A Comparison Model for Agile Web Frameworks

José Ignacio Fernández-Villamor
Departamento de Ingeniería de Sistemas Telemáticos
Univ. Politécnica de Madrid
jifv@gsi.dit.upm.es

Laura Díaz-Casillas
Departamento de Ingeniería de Sistemas Telemáticos
Univ. Politécnica de Madrid
ldcasillas@gsi.dit.upm.es

Carlos Á. Iglesias
Departamento de Ingeniería de Sistemas Telemáticos
Univ. Politécnica de Madrid
cif@gsi.dit.upm.es

ABSTRACT

Nowadays, web development is one of the main activities in software development, with a wide array of tools that make it difficult for developers to deal with its heterogeneity. The appearance of Ruby on Rails has brought a new paradigm to current web development frameworks, and has shown how an agile web development framework can simplify the development process, with a considerable productivity increment. There are several Java-based alternatives to Ruby on Rails, such as Grails, Roma, Trails, JBoss Seam or Sails, with different approaches to the reuse of previous Java frameworks and technologies. This paper proposes a comparison model for agile web frameworks to facilitate developers the selection of the most suitable for each case. This paper reviews the state of the art of agile web frameworks. Afterwards, a comparison model based on a set of evaluation criteria is defined for web framework evaluation. Finally, the model is applied to the most popular web frameworks.

Categories and Subject Descriptors

1.2 [**Relationship between the Web Architecture and other computational areas**]: Frameworks and software architectures for Web-based systems

General Terms

Web frameworks

Keywords

Rails, Grails, Roma, Trails, agile web development, web frameworks

1. INTRODUCTION

Web technologies have a wide acceptance and maturity, which can be observed in the increasing number of services that take the web stack as their base technology. However, web software development is not a mature area, which has caused the proliferation of components, frameworks and tools for the development of web applications. This wide array of tools make it difficult for developers to deal with heterogeneity and keep up to date.

EATIS '08, September 10-12, 2008, Aracaju, Brazil

Web application frameworks (WAF) [8] are defined as a set of classes that make up a reusable design for an application or, more commonly, one tier of one application. This specialization in tiers (persistence, web flow, ...) has led to its classification [20] due to its proliferation. *Agile web frameworks* are defined as full stack web frameworks for developing an application. In contrast with web application frameworks, they are not specialized in one layer, but offer the full stack.

Agile web frameworks started with Ruby On Rails [24], which defined a new approach to web development, based on a single web framework. Ruby on Rails follows the principles of *convention over configuration* and *don't repeat yourself*, providing an agile web development framework that simplifies the development process and increases productivity for prototyping web applications.

However, Ruby on Rails is based on the Ruby language despite the fact that Java is the industry standard for business applications. As a result, a Java-based solution is seeked to let legacy Java systems and technologies be used in future applications while keeping and reusing most libraries, subsystems and technologies already developed in Java.

This has caused the appearance of several Java-based alternatives to Ruby on Rails, such as Grails, Roma, Trails, JBoss Seam or Sails, with different approaches to the reuse of previous frameworks and technologies and the application of agile principles to provide simplicity to web development.

This paper proposes an agile web framework comparison model to assist software project leaders in the selection of a suitable agile web framework for their business goals. The rest of the article is organised as follows. Section 2 reviews the state of the art of agile web frameworks. Then, section 3 presents a model for agile web frameworks comparison and its results on four different web application frameworks. Section 4 shows related work and finally, section 5 presents the main conclusions.

2. AGILE WEB FRAMEWORKS

2.1 Ruby on Rails

Ruby on Rails [24] is a framework that is aimed at agile development of web applications. It was mainly devel-

oped by David Heinemeier Hansson and was extracted out of Basecamp, a production-ready commercial web application. Ruby on Rails' community argues that this extraction is the best proof of the framework's suitability for the development of web applications.

The main principles of Ruby on Rails are *convention over configuration* and *don't repeat yourself*, implying that configuration code should only be used to override default assumptions and that code repetition should be avoided. Rails is written in Ruby [23], a programming language which internally helps to achieve most of Rails' targets due to its abbreviated syntax and dynamic nature.

Rails applications have to follow the Model-View-Controller design pattern [15]. Also, Rails architecture offers the web developer a set of ready available services such as vendor-independent database persistence, automatic code generation for creation, read, update and deletion of resources or integrated testing.

Most Ruby on Rails criticism are due to Ruby's dynamic typing and Rails immaturity. Also, Java is an industry standard for business applications. In spite of initiatives such as JRuby [5], a Java implementation of Ruby that allows the use of Java APIs and application servers, companies still feel reluctant to adopt Ruby on Rails due to the vast number of systems and libraries that are already written in Java.

2.2 Grails

Grails [19] is a Java-based Rails-like development framework that was built in response to Ruby on Rails. As a result, their principles are the same and Grails is heavily Rails inspired. To provide Java integration while offering a dynamic oriented language, Grails is based on the Groovy language [12], a dynamic object-oriented scripting language for the Java virtual machine and with Java-like syntax.

Like Ruby on Rails, Grails is driven by *convention over configuration* and *don't repeat yourself* principles, and therefore most of the development experience is similar to that of Rails. Most differences reside on Grails having a more object-oriented domain definition in comparison with Rails, although Grails offers less community support and application programming interface refinement due to the framework's lower maturity.

Internally, Grails is based on already existing frameworks such as Hibernate [18] and Spring [21], both to take advantage of their maturity and proven stability and to allow easy connection with legacy systems that use those frameworks.

2.3 Trails

Trails [14] is a web development framework that is inspired in Ruby on Rails and Naked Objects [16]. Its target is offering domain driven design by providing a full-stack web application framework based on Tapestry [7], Spring, Hibernate and Acegi [6]. As a result, Trails takes advantage of the stability and maturity of a closed set of already existing frameworks. Trails enhancements to the direct use of the frameworks are tight integration and automatic code generation for common tasks.

2.4 Roma

Roma framework [4] is the latest approach to building an agile Java web development framework. It has been mainly developed by Luca Garulli and is based on the principles of domain driven design [3], model driven architecture paradigm and reusability of previous frameworks. Thus, Roma framework is a metaframework that offers a common application programming interface to a set of pluggable Java frameworks such as Hibernate, Spring or JPOX [9] to transparently provide persistence, presentation or internationalization services. The target is POJO-based development with a minimum coupling with pluggable underlying frameworks and, following the model driven architecture paradigm, provide framework-specific code generation for eventual fine tuning of code.

3. AGILE WEB FRAMEWORKS COMPARISON MODEL

3.1 Framework comparison

Nowadays, there is a wide range of available frameworks and an evaluation is required to determine which is the most suitable for a particular application. Using an inappropiate framework for a system leads to increase development cost and time and reduces software quality.

Web framework comparison is still not a established discipline. The closest area is software architecture comparison, although it is a young discipline. The main contributions are presented in [2]. Some approaches, such as SAAM (Software Architecture Analysis Method) [10] or ATAM (Architecture Tradeoff Analysis Method) [11] allow the assessment about one single architecture and one single design choice. Others like SACAM (Software Architecture Comparison Analysis Method) [22] and DoSAM (Domain-Specific Software Architecture Comparison Model) [1] allow the comparison of different architectures, and follow an approach based on (i) extracting common views of the candidate architectures, (ii) identifying comparison criteria (also so-called quality attributes), (iii) defining indicators in order to measure how a candidate architecture fulfills comparison criteria (so called quality computation weights) and (iv) scoring the candidate architectures.

This article covers all the phases, defining the common views (or blueprint architecture) of agile web development frameworks and a set of parameters to make an evaluation whose results for the four frameworks analyzed are shown at the end of this section.

3.2 Comparison model for agile web frameworks

The goal of the comparison model is facilitating the evaluation of agile web frameworks. This is achieved by a fine-grained characterization of agile web frameworks through an evaluation criteria that is based on the definition of a set of parameters given the common views of the frameworks. Parameters are grouped into a set of aspects shown on figure 1, which summarize general features and issues found in web application development. The aspects are domain description and persistence, presentation, security, usability, testing, service orientation, component orientation and adoption, and are described in the following sections.

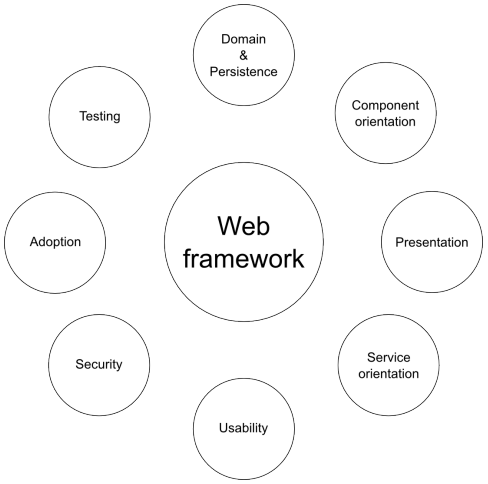

Figure 1: Comparison model.

3.2.1 Domain Description and Persistence

A domain is a conceptualization of the application's field of interest. In software engineering this results in a mapping between a set of classes and the different concepts of the domain, such as, in the case of an e-commerce application, clients, products or bills. In the Model-View-Controller architecture, it is the model what contains the definition of the domain and the used information. Important aspects in domain definition are the following:

- Persistence. When providing database persistence of domain objects, persistence is not an aspect that can be kept transparent. An abstract repository of domain objects that acts as a database is required. As a result, queries for retrieving existing data need to be defined, along with mechanisms for deleting existing objects.
- Data migrations. In the development of a web application, several development iterations produce different domain definitions. Keeping data integrity along the different versions of the domain definition can be handled with migrations. An example of this could be the definition of a class field with a particular type that needs to be changed in a future version of the application without damaging previous stored data. In this case, migrations would help to define how data should be changed when upgrading to the next version of domain definition.
- Constraints. To prevent creation of incorrect domain objects, a model can provide a way for defininining constraints. These constraints will be used for data validation prior to accepting an object as part of the domain.
- Transactions. Some domain changes require complete execution of different subchanges to ensure a coherent final state of the domain. This requires mutual exclusion when accessing data in a context that is separated from the business logic. In the usual case of accessing a database, the mutual exclusion when using this shared resource on performing bulk operations is achieved with transactions. The integration between business logic and the use of transactions can be implemented in a web framework with several approaches, compromising transparency and configurability.

As a result, considering the main comparison aspects that are related to domain description, a set of parameters have been defined:

- *D.1 – Are data migrations built-in in the development process?*: As said, migrations are an important part when developing a web application, allowing preservation of data integrity along different releases.
- *D.2 – Is schema database automatically inferred from domain definition?*: A way of defining a domain is outlining a set of classes that produces a particular database schema to provide persistence.
- *D.3 – Is domain definition automatically inferred from schema data?*: Another way of defining a domain is specifying a database schema that produces a particular class hierarchy and structure.
- *D.4 – Does it support validations?*: Domain model's validations ensure data consistency before storage.
- *D.5 – Does it support transactions?*: The use of transactions enables a coherent state of the domain data when performing bulk operations.

3.2.2 Presentation

The presentation layer, i.e. the view part of the Model-View-Controller design pattern, renders the information of the model. It is not reponsible for incoming data, only manages the information destined to the user, admitting different representations of the same data. Most framewoks enable automatic generation of a basic representation of the model, facilitating the development of the application and encouraging the user feedback. In some cases, the view management is implemented by a external technology, avoiding the use of low-level languages. Finally, internationalization (I18N) and localization manage the adaptation to different cultural environments, enabling the use of several languages depending on the location.

The following presentation-related parameters have been defined:

- *P.1 – Is there a generator of static presentation code for CRUD operations on models?*: CRUD (Create, Read, Update, Delete) operations on models are typical actions that does not have to be hand coded when using generators of static code.
- *P.2 – Is there a generator of dynamic presentation code for CRUD operations on models?*: CRUD operations can also be generated at runtime, avoiding the need of rerunning code generators on domain changes.
- *P.3 – Can presentation layer be defined using an underlying presentation technology?*: Using a specific underlying presentation technology lets the developer use languages that are designed for the specific task of presentation definition.

- *P.4 – Is presentation definition necessarily tied to an underlying presentation technology?*: Getting tied to an underlying presentation technology implies acquiring additional knowledge.
- *P.5 – Does it support internationalization?*: Internationalization and localization are important features on the usual case of providing the presentation layer in different languages.

3.2.3 Security

Applications on the web are under constant attack and it is necessary the use of mechanisms to protect against common problems such as cross site scripting (XSS), injection flaws or malicious file execution [17]. But there is not a clear way to solve these issues, and usually frameworks recommend good practices or provide tools to manage them. Furthermore, most applications deal with multiple aggregated types of users, which often results in the definition of complex business processes. Mechanisms to handle authentication, authorization, and user sessions are necessary to enable users management.

Therefore, considering the security aspect of web application development, a set of evaluation parameters have been defined:

- *S.1 – Does it support static typing?*: Static typing provides security, making it unnecessary to check mistyping or incorrect format.
- *S.2 – Does it support user authentication?*: Managing different types of users is necessary in most applications.
- *S.3 – Does it support automatic escaping to avoid injection attacks?*: Escaping mechanisms increase the security of the application, preventing injection attacks.

3.2.4 Usability

Usability measures a user's ease of interaction with a system. It considers system's functionalities as well as its interface. Usability depends on the type of users who are going to work with the system, what are their goals, and what is the context of use. In the case of web frameworks, usability measures programmers' development experience when usign the framework.

The following usability-related parameters have been defined:

- *U.1 – Does it support automatic generation of code?*: Code generation improves the software development process by avoiding manual coding of typical functionalities.
- *U.2 – Does it support dynamic typing?*: A programming language with dynamic typing improves its readability and development agility at the cost of lacking static typing checks.
- *U.3 – Does it support different deployment environments (development, test, production)?*: Using different environments for each phase in the development process facilitates data management.

3.2.5 Testing

Model-View-Controller-based application testing involve domain, controller and integration testing activities, with performance testing as an essential activity for scalability checking. Although different general testing frameworks are available for several languages, like JUnit or TestNG, these does not take care of specific questions of the web application development:

- Using different environments to development, test and deploy the application faciliates the management of the project. But using separate database for each purpose requires a simple way of populating it with data in the form of database fixtures. Dynamic ways of generating these fixtures (i.e. programatically, e.g. with loops and conditions) allow speeding up the process.
- Sometimes it is difficult to obtain a real scenario to carry out the tests, for instance, when a network connection is necessary. Using mock objects, objects that simulates the behaviour of real objects, facilitates this task and ensures the reliability of the results.
- Also, it is useful to support navigation of XML and HTML elements for assertion of controller responses in functional tests and execution of HTTP requests and sessions to ease the definition of integration tests.
- For performance tests, benchmarking and profiling tools are good helpers to perform scalability and load checks.

The following testing-related parameters have been defined:

- *T.1 – Does it support unit testing?*: Unit testing allows testing of models to ensure proper operation on domain objects.
- *T.2 – Does it support functional testing?*: Functional testing allows testing of controllers to test application business logic.
- *T.3 – Does it support integration testing?*: Integration testing allows testing of the full web application.
- *T.4 – Does it support performance testing?*: Performance testing allows testing of non-funtional performance requirements such as scalability or load testing through benchmarking and profiling.
- *T.5 – Does it support database fixtures?*: Database fixtures are auxiliar definitions of database contents that are load before tests and ease their definition.
- *T.6 – Does it support stub/mock objects?*: Stub/mock objects serve to provide data in tests whenever that data is not available or is very difficult to define.
- *T.7 – Does it support HTTP requests?*: Execution of HTTP requests help to define integration tests and better simulate real use of web applications.
- *T.8 – Does it support HTML/XML parsing?*: HTML/XML parsing allows processing of HTTP responses to test format and content.

3.2.6 Service Orientation

Service Orientation can be implemented through different approaches:

- W3C defines a web service as *"a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other web-related standards".*
- In a Representational Stateless Transfer (REST) approach, a Resource Oriented Architecture (ROA) is proposed, where each resource is referenced using a global identifier (URI) and shares a common interface, consisting of a set of verbs to operate (HTTP methods), for interaction between clients and servers through stateless connections.

Finally, in relation to service oriented support, a set of parameters have been defined:

- *SO.1 – Does it support REST architecture?*: A REST architecture implies using different verbs to perform actions on web resources, along with a specific design style, requiring an appropriate framework support.
- *SO.2 – Does it support web services standards?*: WSDL and SOAP based web services are an industry standard for interoperability that has been widely adopted by companies to support their internal processes and provide business interaction.
- *SO.3 – Does it support semantic web standards?*: Semantic web standards allow the use of ontologies for advanced processing of exchanged data.
- *SO.4 – Does it support automatic generation of RESTful service clients?*: When consuming RESTful services, a client needs to be built to access their data.
- *SO.5 – Does it support automatic generation of web services clients?*: Consuming web services requires building a web service client.

3.2.7 Component Orientation

Component Orientation enables the possibility of using a module or another without code refactoring to complete a task.

The following component orientation-related parameters have been defined:

- *C.1 – Does it support different persistence modules without refactoring?*: Technology independence on domain definition allows using different persistence modules to use framework-specific features or seek compatibility with legacy systems.
- *C.2 – Does it support different testing modules without refactoring?*: Testing technology independence allows taking advantage of framework-specific features for finer testing.
- *C.3 – Does it support different presentation technologies without refactoring?*: Presentation technology independence allows rendering outputs in different formats without additional development effort.
- *C.4 – Does it support generation of underlying framework-specific code?*: Framework-specific code allows fine tuning of framework-specific parameters along with compatibility with legacy systems.
- *C.5 – Does it support connection with systems developed with any of the underlying frameworks?*: If the web framework uses an underlying framework to implement aspects such as persistence or presentation, interoperability would be improved by providing a mechanism to allow connection with other systems developed using any of those frameworks.

3.2.8 Adoption

Resources associated with a technology, such as the number of skilled programmers or existing applications, enable the start up of development projects. Usually, most companies are reluctant to start projects with technologies that are immature or that they are not familiarized with.

Considering these issues, a set of parameters have been defined in relation to adoption:

- *A.1 – Does it have a wide community of users that provide support, documentation and code?*: A big number of active users allows to keep the framework up to date: fixing bugs, developing examples, tutorials, documentation, etc. that help application development.
- *A.2 – Does it have a big amount of programmers in the framework's language?*: The existence of a large number of skilled programmers reduces the risk taken by companies when developing projects.
- *A.3 – Maturity of framework's technologies*: The degree of maturity of the technology has a direct impact on the possible risks in the development of an application. Mature technologies warrant stability, tool support and an experienced community.
- *A.4 – Does it have a big amount of libraries developed in the framework's language?*: The number of available libraries facilitates the reuse of code for the development of new applications.
- *A.5 – Maturity of the application servers that are able to run framework's web applications*: Application servers' maturity warrants application stability.

3.3 Results

As shown on tables 1 and 2, the comparison model has been used to evaluate Ruby on Rails, Grails, Trails and Roma Framework. When applying the model, for each evaluation criteria a set of parameters has been defined. The importance of each parameter is different and it is indicated as

Parameter	Rails	Grails	Trails	Roma
D.1 – Are data migrations built-in in the development process? (20%)	Yes (100%)	No (0%)	No (0%)	No (0%)
D.2 – Is schema database automatically inferred from domain definition? (40%)	No (0%)	Yes (100%)	Yes (100%)	Yes (100%)
D.3 – Is domain definition automatically inferred from schema data? (20%)	Yes (100%)	No (0%)	No (0%)	No (0%)
D.4 – Does it support validations? (10%):	Yes (100%)	Yes (100%)	Yes (100%)	Yes (100%)
D.5 – Does it support transactions? (10%):	Yes (100%)	Yes (100%)	Yes (100%)	Yes (100%)
P.1 – Is there a generator of static presentation code for CRUD operations on models? (20%)	Yes (100%)	Yes (100%)	Yes (100%)	Yes (100%)
P.2 – Is there a generator of dynamic presentation code for CRUD operations on models? (10%)	Yes (100%)	Yes (100%)	No (0%)	No (0%)
P.3 – Can presentation layer be defined using an underlying presentation technology? (20%)	Yes, by embedding Ruby code (100%)	Yes, by embedding Groovy tags (100%)	No, it can only be defined using Tapestry framework (0%)	No, presentation has to be defined through annotated POJOs (0%)
P.4 – Is presentation definition necessarily tied to an underlying presentation technology? (20%)	Yes, requiring redefinition of presentation for each output format (0%)	Yes, requiring redefinition of presentation for each output format (0%)	Yes, being tied to Tapestry framework (0%)	No, presentation is defined through annotated POJOs to allow technology independence (100%)
P.5 – Does it support internationalization? (30%)	No, only through third-party plugins (50%)	Yes (100%)	Yes (100%)	Yes (100%)
S.1 – Does it support static typing? (40%)	No. Ruby language is required (0%)	No. Groovy is required in controllers and models, although using Java classes is straightforward (25%)	Yes, due to the use of Java (100%)	Yes, due to the use of Java (100%)
S.2 – Does it support user authentication? (20%)	No, only through third-party plugins (50%)	No, only through third-party plugins (50%)	No, only through third-party plugins (50%)	Yes (100%)
S.3 – Does it support automatic escaping to avoid injection attacks? (20%)	Yes (100%)	Yes (100%)	Yes (100%)	Yes (100%)
U.1 – Does it support automatic generation of code? (40%)	Yes (100%)	Yes (100%)	Yes (100%)	Yes (100%)
U.2 – Does it support dynamic typing? (20%)	Yes (100%)	Yes (100%)	No (0%)	No (0%)
U.3 – Does it support different deployment environments (development, test, production)? (40%)	Yes (100%)	Yes (100%)	No (0%)	No (0%)
T.1 – Does it support unit testing? (20%)	Yes (100%)	Yes (100%)	No, although any Java testing framework can be used (50%)	No, although any Java testing framework can be used (50%)
T.2 – Does it support functional testing? (20%)	Yes (100%)	Yes (100%)	No (0%)	No (0%)
T.3 – Does it support integration testing? (20%)	Yes (100%)	No (0%)	No (0%)	No (0%)
T.4 – Does it support performance testing? (20%)	Yes (100%)	No (0%)	No (0%)	No (0%)
T.5 – Does it support database fixtures? (5%)	Yes (100%)	No (0%)	No (0%)	No (0%)
T.6 – Does it support stub/mock objects? (5%)	Yes (100%)	Yes, through the use of Spring mock objects (50%)	No (0%)	No (0%)
T.7 – Does it support HTTP requests? (5%)	Yes, integrated in the testing framework (100%)	No, requiring coding or external libraries (0%)	No, requiring coding or external libraries (0%)	No, requiring coding or external libraries (0%)
T.8 – Does it support HTML/XML parsing? (5%)	Yes, integrated in the testing framework (100%)	No, requiring coding or external libraries (0%)	No, requiring coding or external libraries (0%)	No, requiring coding or external libraries (0%)
SO.1 – Does it support REST architecture? (20%)	Yes (100%)	Yes (100%)	No (0%)	No (0%)
SO.2 – Does it support web services standards? (20%)	Through Active Web Service (AWS) plugin (50%)	Through CXF plugin (50%)	No direct support. Spring Web Services features can be used (50%)	No direct support. Spring Web Services features can be used (50%)
SO.3 – Does it support semantic web standards? (20%)	No direct support. Plugins such as SWORD or ActiveRDF can be used (50%)	No direct support. Semantic web frameworks for Java like Jena or Sesame can be used (50%)	No direct support. Semantic web frameworks for Java like Jena or Sesame can be used (50%)	No direct support. Semantic web frameworks for Java like Jena or Sesame can be used (50%)

Table 1: Framework comparison (part 1).

Parameter	Rails	Grails	Trails	Roma
SO.4 – Does it support automatic generation of RESTful service clients? (20%)	Yes (100%)	No (0%)	No (0%)	No (0%)
SO.5 – Does it support automatic generation of web services clients? (20%)	Yes, using AWS plugin (50%)	No (0%)	No (0%)	No (0%)
C.1 – Does it support different persistence modules without refactoring? (20%)	No (0%)	No (0%)	No (0%)	Yes. Current release supports JDO 2.0 standard using JPOX 1.1.4 (100%)
C.2 – Does it support different testing modules without refactoring? (20%)	No (0%)	No (0%)	No (0%)	No (0%)
C.3 – Does it support different presentation technologies without refactoring? (20%)	No (0%)	No (0%)	No (0%)	Yes. Currently Echo2 and JSP available as resulting code (100%)
C.4 – Does it support generation of underlying framework-specific code? (20%)	No (0%)	No (0%)	No (0%)	Yes (100%)
C.5 – Does it support connection with systems developed with any of the underlying frameworks? (20%)	Yes (100%)	Yes (100%)	Yes (100%)	Yes (100%)
A.1 – Does it have a wide community of users that provide support, documentation and code? (20%)	Yes (100%)	Yes (100%)	No (0%)	No (0%)
A.2 – Does it have a big amount of programmers in the framework's language? (20%)	No (0%)	No (0%)	Yes (100%)	Yes (100%)
A.3 – Maturity of framework's technologies (20%)	Growing (50%)	Growing (50%)	Mature (100%)	Mature (100%)
A.4 – Does it have a big amount of libraries developed in the framework's language? (20%)	No (0%)	Yes (100%)	Yes (100%)	Yes (100%)
A.5 – Maturity of the application servers that are able to run framework's web applications (20%)	Growing (50%)	Mature (100%)	Mature (100%)	Mature (100%)

Table 2: Framework comparison (part 2).

a percentage. The value of each parameter is obtained by considering (i) general degree of fulfilment and (ii) degree of integration of the approach (integrated or through third-party plugins). Results are summarized on figure 2.

At first glance, Rails evaluation shows some of its criticised aspects, such as lack of static typing, interoperability with other systems, technology immaturity or shortage of experienced programmers. However, it is a leap ahead in terms of usability against previous web frameworks, and provides a good support for application testing.

In the case of Grails, it broadly follows Rails with testing features and ease of use, being still a work in progress. Technologically, it is a more mature alternative, although the Groovy language is a similar barrier as Ruby, which may not justify the former's adoption.

Trails is based on existing technologies such as Spring or Hibernate. It tries to simplify the development process by taking some ideas from Rails at the cost of losing the interoperability that would be obtained by using its underlying technologies independently.

Finally, Roma framework is a novel approach and mostly a work in progress. This can be noticed due to its lack of features such as testing support. However, its design principles show its strengths over the rest at static typing and interoperability.

4. RELATED WORK

The work presented in this paper proposes a novel approach for characterizing agile web frameworks and its usage for framework selection. The work by Shan et al. [20] presents a taxonomy of Java web frameworks which is related to our work. The main difference is that in this work we are dealing with agile web frameworks instead of Java web frameworks, although this taxonomy is partly applicable. Kong et al. [13] proposes a web application architecture framework (WAAF) which provides different perspectives depending on the user (planner, business owner, architect, etc.) and dimensions. WAAF is intented for analysing a web application, while our work intends to characterize an agile web framework.

5. CONCLUSIONS

In this paper, an approach to agile framework comparison has been proposed. It is based on the definition of a set of parameters given the common views of the frameworks. Parameters are grouped into a set of aspects that summarize general features and issues found in web application development.

The model has been applied on the main agile web frameworks: Rails, Grails, Trails and Roma. Relevant results are obtained, in terms of interoperability, maturity, usability or testing. This way, the comparison model tries to help in choosing among different frameworks before starting an application, which is a key stage in web application development projects.

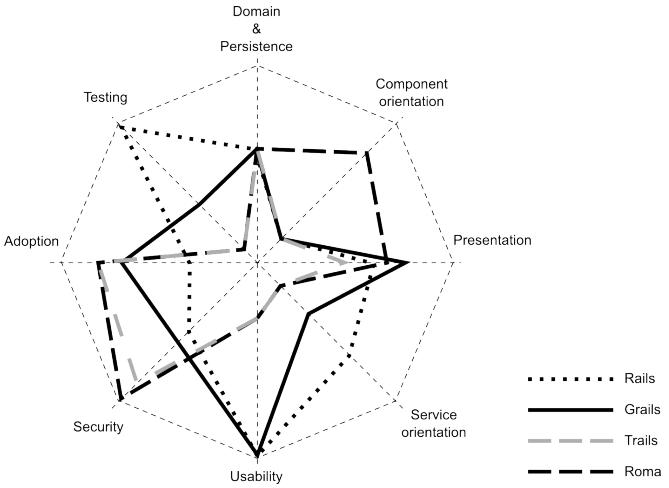

Figure 2: Framework evaluation results.

Acknowledgements

This research project is funded by the European Commision under the R&D project ROMULUS (FP7-ICT-2007-1) and by the Spanish Government under the R&D project Java sobre Ruedas (FIT-350401-2007-8).

6. REFERENCES

[1] K. Bergner, A. Rausch, M. Sihling, and T. Ternité. DoSAM - Domain-Specific Software Architecture Comparison Model. In *Quality of Software Architectures and Software Quality*, Lecture Notes in Computer Science, pages 4–20. Spring Verlag, 2005.

[2] P. Clemens, R. Kazman, and M. Klein. *Evaluating Software Architectures, Methods and Case studies*. SEI Series in Software Engineering, 2002.

[3] E. Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison Wesley, 2003.

[4] L. Garulli. Roma framework's web page. `http://www.romaframework.org`, 2006.

[5] C. Hibbs. JRuby's killer feature. `http://www.oreillynet.com/ruby/blog/2006/11/jrubys_killer_feature.html`, 2006.

[6] Interface21 Inc. Acegi Security's web page. `http://www.acegisecurity.org/`, 2007.

[7] Jakarta. Tapestry web page. `http://jakarta.apache.org/tapestry`, 2006.

[8] R. Johnson. J2EE development frameworks. *Computer*, 38(1):107–110, 2005.

[9] JPOX Team. JPOX's web page. `http://www.jpox.org`, 2008.

[10] R. Kazman, L. Bass, G. Abowd, and M. Webb. SAAM: A method for analyzing the properties software architectures. *Proceedings of the 16th International Conference on Software Engineering*, pages 81–90, May 1994.

[11] R. Kazman, M. Klein, M. Barbacci, H. Lipson, T. Longstaff, and S. Carrière. The architecture tradeoff analysis method. *Proceedings of ICECCS*, August 1998.

[12] D. Koenig, A. Glover, P. King, G. Laforge, and J. Skeet. *Groovy in Action*. Manning publications Co., 2007.

[13] X. Kong, L. Liu, and D. Lowe. Separation of concerns: a web application architecture framework. *Journal of Digital Information*, 6(2), 2005.

[14] K. Korhonen. Trails framework's web page. `http://www.trailsframework.org`, 2006.

[15] A. Leff and J. T. Rayfield. Web-application development using the model/view/controller design pattern. *Fifth IEEE International Enterprise Distributed Object Computing Conference*, 2001.

[16] Naked Objects. Naked Objects's web page. `http://www.nakedobjects.org`, 2007.

[17] OWASP Foundation. The ten most critical web application security vulnerabilities. `http://www.owasp.org/images/e/e8/OWASP_Top_10_2007.pdf`, 2007.

[18] Red Hat. Hibernate's web page. `http://www.hibernate.org`, 2006.

[19] J. Rudolph. *Getting started with Grails*. InfoQ – Enterprise Software Development Series, 2007.

[20] T. C. Shan and W. W. Hua. Taxonomy of java web application frameworks. In *ICEBE '06: Proceedings of the IEEE International Conference on e-Business Engineering*, pages 378–385, Washington, DC, USA, 2006. IEEE Computer Society.

[21] SpringSource. Spring framework's web page. `http://www.springframework.org`, 2006.

[22] C. Stoermer, F. Bachmann, and C. Verhoef. SACAM: The software architecture comparison analysis method. Technical Report CMU/SEI-2003-TR-006, Carnegie Mellon University - Software Engineering Institute, 2003.

[23] D. Thomas, C. Fowler, and A. Hunt. *Programming Ruby: The Pragmatic Programmer's Guide*. The Pragmatic Bookshelf, 2004.

[24] D. Thomas, D. Heinemeier Hansson, L. Breedt, M. Clark, J. Duncan Davidson, J. Gehtland, and J. Schwarz. *Agile Web Development with Rails*. The Pragmatic Bookshelf, 2nd edition, 2006.