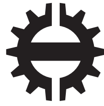


TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JUHANA HARMANEN

POLYGLOT PROGRAMMING IN WEB DEVELOPMENT

Master's thesis

Examiner: Professor Tommi Mikkonen
Examiner and topic approved by the Faculty meeting of the Faculty of Computing and Electrical Engineering on 8 May 2013.

للاستشارات المنارة

www.manaraa.com

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
HARMANEN, JUHANA: Polyglot Programming in Web Development
Master of Science Thesis, 90 pages
September 2013
Major: Software engineering
Examiner: Professor Tommi Mikkonen
Keywords: Polyglot programming, web development, polyglot programming on the JVM, poly-paradigm programming, Java, Groovy, Spring Framework, Grails, Vert.x, AngularJS

Different programming languages are used to solve different problem domains. Front-end code standards and best practices are used to separate presentation, content and behavior. Architectural approaches like three-tier client-server architecture present user interface, business logic and data access as independent modules to develop and maintain.

The idea of polyglot programming is to combine and utilize the best solutions from different programming languages and paradigms. Therefore, polyglot programming has the potential to improve web development in various areas. Web development has always been polyglot.

Polyglot system has two essential aspects, the platform used for the integration and the programming languages supported. The recent rise of non-Java programming languages running on the Java Virtual Machine has created a favorable environment for polyglot programming. The possibility to use more expressive and succinct programming languages with existing solutions has proven to be essential in web development.

An example web project was implemented to study the observations in practice. The project was implemented in both Java and Groovy as a server-side web application and also with Vert.x and AngularJS as a client-side single-page application. Also an additional Groovy implementation with Java legacy domain model was implemented to study programming language interoperability on the Java Virtual Machine. The results were evaluated against related work consisting two project implementations and three case study projects presented also in the context of polyglot programming in web development.

Polyglot programming can enhance web development, because different programming languages and frameworks promise an increase in productivity, reduced amount of code and improved code quality that together promote better maintainability. Although polyglot programming has a steep learning curve that affects on required knowledge, maintainability, and tool support.

www.manaraa.com

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
JUHANA HARMANEN: Monikieliohjelmointi Web-sovelluskehityksessä
Diplomityö, 90 sivua
Syyskuu 2013
Pääaine: Ohjelmistotuotanto
Tarkastajat: professori Tommi Mikkonen
Avainsanat: monikieliohjelmointi, Web-sovelluskehitys, monikieliohjelmointi Java virtuaalikoneessa, moniparadigmaohjelmointi, Java, Groovy, Spring Framework, Grails, Vert.x, AngularJS

Erilaisia ongelma-alueita pyritään ratkaisemaan käyttäen eri ohjelmointikieliä. Esimerkiksi Frontend-ohjelmointikäytäntöjä noudattamalla voidaan erottaa esitys-, sisältö ja toimintalogiikka toisistaan. Lisäksi arkkitehtuuriratkaisut, kuten kolmikerrosarkkitehtuuri, jakavat käyttöliittymän, toimintalogiikan ja tietovarastojen käytön itsenäisiksi, mahdollisesti toisistaan erillään kehitettäviksi ja ylläpidettäviksi moduuleiksi.

Monikieliohjelmoinnin ideana on yhdistää ja hyödyntää ohjelmointikielten ja ohjelmointiparadigmojen parhaat ratkaisut. Tästä syystä monikieliohjelmointi voi mahdollisesti parantaa Web-sovelluskehitystä useilla eri osa-alueilla. Monikielisyys on aina ollut osa Web-sovelluskehitystä.

Monikielisessä järjestelmässä on kaksi olennaista ominaisuutta, käytettävissä oleva sovelluskehitysalusta sekä käytettävissä olevat ohjelmointikielet. Viimeaikainen kehitys ohjelmointikielissä Java-virtuaalikoneella on luonut suotuisan ympäristön monikieliohjelmoinnille. Mahdollisuus käyttää kuvaavampia ja ytimekkäämpiä ohjelmointikieliä olemassa olevien ratkaisujen tukena on osoittautunut tärkeäksi.

Työn yhteydessä toteutettua esimerkkiprojektia käytettiin tarkastelemaan tehtyjä havaintoja käytännössä. Projekti toteutettiin sekä Java- että Groovy-ohjelmointikielellä palvelinpuolen Web-sovelluksena sekä Vert.x ja AngularJS Web-sovelluskehyksiä hyödyntäen asiakaspuolen yhden sivun Web-sovelluksena. Lisäksi ohjelmointikielten yhteentoimivuutta tutkittiin tekemällä Groovy-ohjelmointikielellä toteutus, joka hyödynsi valmista Java-toteutuksen toimialueen mallinnusta.

Monikieliohjelmointi voi tehdä Web-sovelluskehityksestä kannattavampaa. Eri ohjelmointikielet ja ohjelmistokehykset lupaavat lisätä tuottavuutta, vähentää tarvittavan koodin määrää, sekä parantaa koodin laatua, parantaen näin samalla ylläpidettävyyttä. On kuitenkin tärkeää huomata se, että monikieliohjelmointi kasvattaa tarvittavan tiedon määrä, mikä vaikuttaa suoraan ylläpidettävyyteen sekä tarvittavaan työkalutukeen.

www.manaraa.com

PREFACE

This thesis work started over a year ago from my own interest to learn and use multiple programming languages rather than stick with the one I knew the best. At that time a new Vert.x web framework was introduced and it offered new approaches to polyglot programming.

This thesis work started while I was working at Solita Oy. I want to thank everyone contributing to my idea and especially Solita postgraduate group for proofreading this thesis on the long run. My warmest thanks go to my thesis supervisor and examiner Professor Tommi Mikkonen for providing his expertize and finding the time for this thesis.

This thesis proved to be quite demanding, but as it evolved it was satisfactory. This thesis allowed me to pursue my own interest in polyglot programming, thus making myself a better developer, and at the same time hopefully helping others to achieve the same goal.

المنارة للاستشارات

www.manaraa.com

CONTENTS

1. Introduction . . . 1
2. Polyglot programming . . . 3
  2.1 Definition . . . 4
  2.2 Research context . . . 5
  2.3 Associated advantages . . . 5
  2.4 Associated disadvantages . . . 6
  2.5 Polyglot programming in web development . . . 7
    2.5.1 Development . . . 8
    2.5.2 Testing . . . 10
    2.5.3 Deployment . . . 11
    2.5.4 Concurrency . . . 12
    2.5.5 Business rules . . . 14
  2.6 Polyglot software systems . . . 14
  2.7 Polyglot programming pyramid . . . 15
  2.8 Extending the polyglot programming pyramid . . . 18
    2.8.1 Improving the bounded fractal representation . . . 18
    2.8.2 Supporting architectural decision making . . . 20
  2.9 Guidelines for polyglot programming . . . 20
  2.10 Poly-paradigm programming . . . 23
  2.11 Programming language features and tool support . . . 25
3. Polyglot programming on the Java platform . . . 28
  3.1 Java platform . . . 29
  3.2 Evolution of polyglot programming . . . 30
  3.3 Programming languages on the Java Virtual Machine . . . 32
    3.3.1 Java . . . 33
    3.3.2 Groovy . . . 34
    3.3.3 Scala . . . 35
    3.3.4 Clojure . . . 36
  3.4 Vert.x framework for the modern web and enterprise . . . 37
    3.4.1 Effortless asynchronous application development . . . 38
    3.4.2 Verticle and Vert.x instances . . . 39
    3.4.3 Core services and modules . . . 40
    3.4.4 Polyglot programming with Vert.x . . . 42
    3.4.5 Support for new programming languages . . . 44
4. Implementation . . . 45
  4.1 Project structure . . . 45
    4.1.1 Web project with Java using Spring Framework and Hibernate . . . 46

المنارة للاستشارات
www.manaraa.com

4.1.2 Web project with Groovy using Grails framework 47
4.1.3 Web project with Groovy using Grails framework and Java legacy domain model 48
4.1.4 Single-page application with Vert.x framework and AngularJS 48
4.2 Web flow execution, decorators and mapping 50
4.3 Form objects, binding and validation 53
4.4 Model, Repositories and Services 54
4.4.1 Model 55
4.4.2 Repositories 58
4.4.3 Services 59
4.5 View 60
4.6 Observations on Groovy as the programming language 66
4.6.1 Amount of code 66
4.6.2 Code quality 66
4.6.3 Productivity 67
4.7 Observations in web development: traditional methods versus client-side . 67
4.7.1 Amount of code 67
4.7.2 Code quality 68
4.7.3 Productivity 68
4.7.4 Testing 69
5. Evaluation 70
5.1 Related work and previous results 70
5.1.1 Example web project with Java and Scala 71
5.1.2 Buypass: JRuby with existing Java libraries 71
5.1.3 Web based extranet: JRuby with existing Java legacy 72
5.1.4 Au2sys: Java web application with RSpec and Watir tests 72
5.1.5 Required knowledge 72
5.1.6 Amount of code 73
5.1.7 Code quality 73
5.1.8 Productivity 74
5.2 Discussion of the results 74
5.2.1 Amount of code 75
5.2.2 Code quality 75
5.2.3 Productivity 76
6. Conclusion 77
6.1 Recommendations 77
6.2 Future work 78
References 80

المنارة للاستشارات
www.manaraa.com

1. INTRODUCTION

Over the past decade, there has been a prominent focus on using a standard programming language in web development projects. Such programming languages include Java, Microsoft's technologies and PHP. Rationale behind this is that one standard programming language is easier to use, both for developers, management when hiring and tutoring employees, and system administrators. The reasoning is valid, while it fails to reckon that the programming environment is nominally the programming language and more so frameworks and tools. Frameworks for collections, object-relational mapping, XML handling, web development and web services are made to simplify the development process. Despite the fact that all of these frameworks are written in the same programming language, they still introduce new abstractions and require vast knowledge to configure and use.

The rationale to utilize a standard programming language overlooks the fact that humans tend to use different languages to make expressions more effective and succinct. Therefore, if the implementation comes more naturally in another programming language instead of a new framework, it might prove to be a more viable solution. This approach is coined polyglot programming, whose goal is to render simpler, more expressive and fluent solutions by combining the best solutions from different programming languages and paradigms.

Polyglot programming applied in software engineering introduces the use of several programming languages in a single software system. This approach is widely adapted and is in extensive use in web development, for example, embedded HTML and SQL or JavaScript with CSS and HTML. Nevertheless, only limited research has been conducted on this topic, with focus on refactoring polyglot systems or on the possible business benefits of polyglot programming. The support for polyglot programming might prove essential in the future, without the need to rewrite working legacy code.

This thesis conducts thorough research on polyglot programming in web development context and more so on the Java platform. Questions like how polyglot programming is used in modern-day web development, how it can be used to render better solutions, and how to use it on the Java Virtual Machine are covered. Goals are to increase developer productivity, reduce the amount of code, improve the code quality and software maintainability in web development. Polyglot programming on the Java Virtual Machine provides several programming languages and frameworks that differ, for example, in syntax, type system, idioms and programming paradigm, and in coding conventions and best practices.

المنارة للاستشارات
www.manaraa.com

Chapter 2 provides a comprehensive study on the theory and practices behind polyglot programming, and also reveals associated advantages and disadvantages. Polyglot programming in the context of web development is covered with examples. The structure of a polyglot software system is studied and the polyglot programming pyramid introduced. In addition, the polyglot programming pyramid is extended as an architectural pattern to include also frameworks and libraries, and to support decision making when using polyglot programming. Basic guidelines on polyglot programming are also described. In addition, poly-paradigm programming, and programming language features and tool support are covered. This chapter describes why, how and when polyglot programming can be used.

Chapter 3 covers polyglot programming on the Java platform thoroughly. Java platform and its evolution to support multiple programming languages on the Java Virtual Machine is described. In addition, the runtime support of the Java Virtual Machine for different programming languages is explained and several programming languages used in this thesis introduced. Also a new and noteworthy Vert.x framework and its support for polyglot programming is disclosed.

A more practical approach is taken in the Chapter 4. An example web development project is implemented and analyzed in the polyglot programming context. The project is implemented in both Java and Groovy as a server-side web application and also with Vert.x and AngularJS frameworks as a JavaScript client-side single-page application. In addition, a Groovy project with Java legacy domain model is implemented to research possibilities in programming language interoperability on the Java Virtual Machine. This chapter provides observations on Groovy as the programming language, and in web development comparing traditional methods with client-side application architecture.

Chapter 5 summarizes and evaluates the observations against related work and previous results. Observed advantages and disadvantages are generalized from programming language specific subjects to more general results of the polyglot programming approach in the discussion of the results.

Chapter 6 concludes this thesis and presents recommendations on polyglot programming. It also discusses some future directions for later studies in the context of polyglot programming.

المنارة للاستشارات
www.manaraa.com

2. POLYGLOT PROGRAMMING

The separation of languages that is forced in writing can actually be considered quite unnatural in this multilingual world. Writers tend to break this separation because it is too restrictive. For example journalists and historians often mix philosophic prose with statistical facts [1; 2]. A Russian author Leo Tolstoy is an example of a polyglot literature writer. Tolstoy frequently inserted words and phrases from French and German in his novel War and Peace. The purpose of being as expressive and effective in language as possible, not as a device of arrogance or pedantic. Some rare and inventive authors are known even to make up their own languages, for example, J. R. R. Tolkien. [3]

Each language has innumerable amount of nuances that makes it more expressive and distinct from all other languages. An author of good literature considers every word to contribute meaningfully to the whole, meaning that the whole is not complete or correct without that specific word. The decision is based on a careful consideration of definition, context, connotation, sense, length, look, and the author's general attitude towards the word [3]. If no word in the main language of the work is just right, the author will look to another language to provide the necessary depth and exactness.

Tolstoy mixed languages because he understood that in certain situations, a language can outdo another language by being more expressive, effective, succinct, vivid, et cetera. To make this point more valid, consider the novel War and Peace. The novel is originally written in Russian, but despite translation to English, the phrases that Tolstoy inserted from other languages than Russian were not translated to English. This emphasize the definite importance for those pieces of the text. [1; 3]

Tolstoy is not the only author to mix languages. Mixing languages is just another literature tool to exploit polyglot literature. Vladimir Nabakov's novel Lolita is written mainly in English but contains a lot of French phrases. Nabakov used phrases filled with connotation with well placed French colloquialism. A literal translation for "Joie de vivre" is "Joy of living" which loses all the rich connotations that reside in the French phrase. In English it is used to express a cheerful enjoyment of life. Furthermore in French it can be a joy of conversation, joy of eating, joy of anything one might do. It is seen as joy of everything, a comprehensive joy, a philosophy of life. [4]

Another example is the French phrase "Fin de siècle" used by Alex Ross in his first book, The Rest Is Noise: Listening to the Twentieth Century. A literal translation is the "end of the century" which refers to an English idiom "Turn of the century". Turn of

المنارة للاستشارات
www.manaraa.com

the century refers to the transition from one century to another, and it is used to indicate a distinctive time period either before or after the beginning of a century or both before and after. The translation loses the rich connotation that the French phrase encompasses, meaning as it was felt to be a period of degeneration, but at the same time a period of hope for a new beginning. [4]

Programming languages do not differ from natural languages. Each programming language has its own nuances that makes it distinct and expressive in ways that other programming languages are not. This can be seen as a justification for intermingling different programming languages. Watts [3] argues that programming languages should not compete, they should rather be mixed together to form and achieve the perfect programming language, and that no singular programming language can achieve this alone. [3; 5]

Software that is composed of artifacts written in multiple programming languages is pervasive in modern-day software business. The idea of polyglot programming is to render more natural and simpler solutions by combining the best available solutions from different programming languages and paradigms. Thus polyglot programming is also poly-paradigm programming. Polyglot programming identifies the realization that there is “No Silver Bullet”, a tool best at solving all problems [6]. In software development this entail an examination of programming languages, frameworks and development tools most suitable for the task at hand.

Polyglot programming has also been known as multi-language programming [7; 8; 9; 10]. Lately it has also been referred to as cross-language programming merely applied in the field of code analysis and refactoring [11; 12]. Similar ideas are also expressed in language oriented programming which is a development methodology focusing on creation of domain-specific languages [13; 14; 15; 16; 17; 18; 19].

2.1 Definition

The term *polyglot programming* was first introduced in software development context in 2002 [20]. A hypothesis was made for several programming languages within one environment. Later authors tend to use slightly different approaches when they describe polyglot programming [13; 21]. The best description to work with was given by Watts as *“programming in more than one language within the same context”* [1], which postpones the definition onto what the context is. Definition of using multiple programming languages on the same managed runtime was suggested [22]. Managed runtime is definitely polyglot programming, but the definition should not restrict the architecture.

From a developer’s perspective, the context is considered as people working on the project. And more so, the context depends on the number of teams and the way the developed applications are integrated together. Polyglot programming is constituted even if one team uses different programming language regardless of the chosen architecture. If the integration between parts of an application developed by two separate teams using differ-

المنارة للاستشارات
www.manaraa.com

ent programming languages is tight, it is considered as polyglot programming. However, when the separate teams do not need information about programming languages the other teams are using, an application is no longer considered as polyglot. Denoting that the parts of an application could be seen as distinct entities. An example would be a service-to-service application in which knowledge of the interfaces is the only requirement.

Fjeldberg [23] defines and expands the previously described concept of polyglot programming within a context. The formal definition of polyglot programming by Fjeldberg follows:

> *"programming in more than one language within the same context, where the context is either within one team, or several teams where the integration between the resulting applications require knowledge of the languages involved"* [23].

2.2 Research context

A literature study will be conducted to discover how polyglot programming is used in modern-day web development. The key is to scrutinize how polyglot programming can be used to render the best, meaning expressive, effective, succinct, and fluent solution to the problem at hand. A case study will be conducted from the developers perspective to discover how to implement polyglot programming on the Java Virtual Machine. The implementations will be evaluated against related work and previous results from two project implementations by Lähteenmäki [24] and three case study projects by Fjeldberg [23]. The following research questions form the basis for the research:

RQ1: *How polyglot programming is used in modern-day web development?*

RQ2: *How to use polyglot programming to render the best solution?*

RQ3: *How to implement polyglot programming on the Java Virtual Machine?*

2.3 Associated advantages

Definition and measurement of productivity are much debated aspects of programming languages. Two of the most used metrics are lines of code (LOC) [25; 26] and function points [25; 27] per unit time. Regardless of metrics, an additional problem of assessing the productivity of different programming languages exists, although it has been claimed that productivity does not depend on programming language [28]. Delorey [25] presents evidence to the contrary based on the assumption of insufficient data. Due to the nature of the problems within the scope of the productivity measurement, any findings from case studies are hard to generalize. Problems include human factors like motivation, skill and experience, and environmental factors like integrated development environment (IDE) and library support, and also factors on geographical distribution of projects [29; 30; 31; 32; 33; 34].

المنارة للاستشارات
www.manaraa.com

A premise for increased productivity comes from the main idea of polyglot programming to combine and integrate the best solutions from different programming languages thus rendering simpler solution to the problem at hand [5; 23]. A suitable programming language for a particular problem will normally render a shorter solution in terms of LOC because of the built-in primitives and idioms. Following the assumption that developers produce the same amount of lines of code regardless of programming language, they use high-level languages that require less lines of code to be more productive [28]. In addition to reduced LOC, the thought process of a developer will normally be shorter because the solution comes naturally in the appropriate programming language. The work can be done on the problem and without the required low-level plumbing. For example, taking advantage of static polymorphic type-checking in functional programming, a large class of programming errors on race conditions and deadlocks in message passing between processes can be caught at compile time [35; 36]. The nature of interpreted programming languages can further increase productivity because no compile cycles are needed.

A general realization in web development is that developers are more expensive than hardware which means that the importance of a developer's productivity transcends that of runtime performance [23]. This results in shorter development cycle providing faster time-to-market or the possibility of fewer developers working on the same application. However, the development phase of an application is only a part of the life cycle, spanning often from 5 to 10 years. Therefore, the increased productivity from choosing an appropriate programming language would become even more important in the maintenance phase [37]. Furthermore the application written with less lines of code will have fewer lines of code to maintain, as well as fewer instructions to follow. The effort to maintain an application increases exponentially with the number of instructions to follow, and therefore the amount should be kept in minimum [28; 37]. Research also reveals that the number of faults per lines of code increases with the total lines of code in the application [38; 39; 40].

2.4 Associated disadvantages

Knowledge of different programming languages is essential in order to benefit from polyglot and poly-paradigm programming [5; 23]. This results in a problem, because not all developers have vast knowledge over different programming languages and some are not even interested in learning new ones [41]. Although it is suggested that developers should learn at least one new programming language per year to evolve [42], in many cases this has been proven not realistic [43; 44], and also in many situations learning a new programming language takes more than a year [45; 46]. The learning curve becomes even steeper when developers have accustomed to a single programming language and the infrastructure, tools and certifications built around it. Therefore, the required amount of knowledge for developers is increased especially in the hiring process and when selecting

المنارة للاستشارات
www.manaraa.com

programming languages to use [5; 23]. In addition, different problem areas should also be assessed.

Graham [47] presents a conceptual hierarchy with a more expressive and succint programming languages at the top. The so-called blub paradox after a hypothetical programming language of average complexity called "Blub" states that anyone preferentially using a particular programming language knows that it is more powerful than some, but not that it is less powerful than others. Thus writing in some programming language means thinking in that programming language, and that typically programmers are satisfied with whatever programming language they happen to use, because it dictates the way they think about programs.

Administration phase requires a sufficient knowledge of the programming language used in order to conduct maintenance. The administration of a large application with a long life cycle, spanning from 5 to 10 years, is likely conducted by different developers or even by a different company than that who developed the application. This is further enhanced every time a new programming language is added, resulting in decrease in the pool of developers with enough knowledge to maintain the application [41; 48]. In addition, using a new paradigm parallel to a previously used one will make following the application code even harder.

Developers using Java and .NET have accustomed to a diverse and comprehensive IDE support with integrated and plugin features like version control, syntax highlighting, refactoring, debugging et cetera. A support for a new programming language will normally only be implemented if it gains enough traction and popularity, because adding a support requires usually a tremendous amount of work [23; 49]. Therefore, the overhead caused by using different programming languages will increase if the tools do not offer interoperability, and different tools must be configured and used.

2.5 Polyglot programming in web development

Front-end code standards and best practices today separate the presentation, content and behavior. By maintaining consistency in coding styles and conventions the burden of maintaining legacy code can be eased, and the risk of breakages in the future can be mitigated. Optimized page loading, efficient performance and code maintainability can be achieved by adhering the best practices.

Presentation is separated from the content by using cascading style sheets. JavaScript is commonly used to make the web pages more interactive, and to provide more vivid and genuine behavior. JavaScript can also be used to interact with server-side programming languages, and with databases where the information is stored. This means that most web applications are a polyglot system, and that they use at least four different programming or specification languages in development. Thus web development has always been polyglot [21].

المنارة للاستشارات
www.manaraa.com

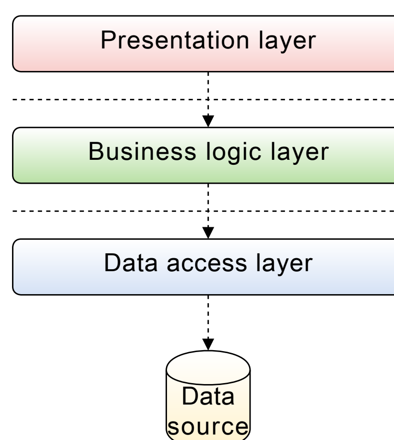


Figure 2.1: *Three-tier architecture represents a polyglot software system.*

Web applications usually follow three-tier architecture, shown in Figure 2.1, or some of its derivatives, which can all be considered as a representation of a polyglot software system. In addition, polyglot programming using a managed runtime is becoming more popular in web development because of the increasing support for polyglot programming on the Java and .Net platforms.

2.5.1 Development

Excellent high-productivity frameworks are the main reason for the increased popularity of polyglot programming in web development. In addition, frameworks created using dynamic programming languages promise even further increase the developer productivity and provide faster turnover. Usually these frameworks include support for generating JavaScript. The most important of these high-productivity web frameworks include Groovy based Grails, Ruby based Ruby on Rails, and Python based Django. The importance of these frameworks is emphasized by the fact that frameworks in other programming languages use same ideas.

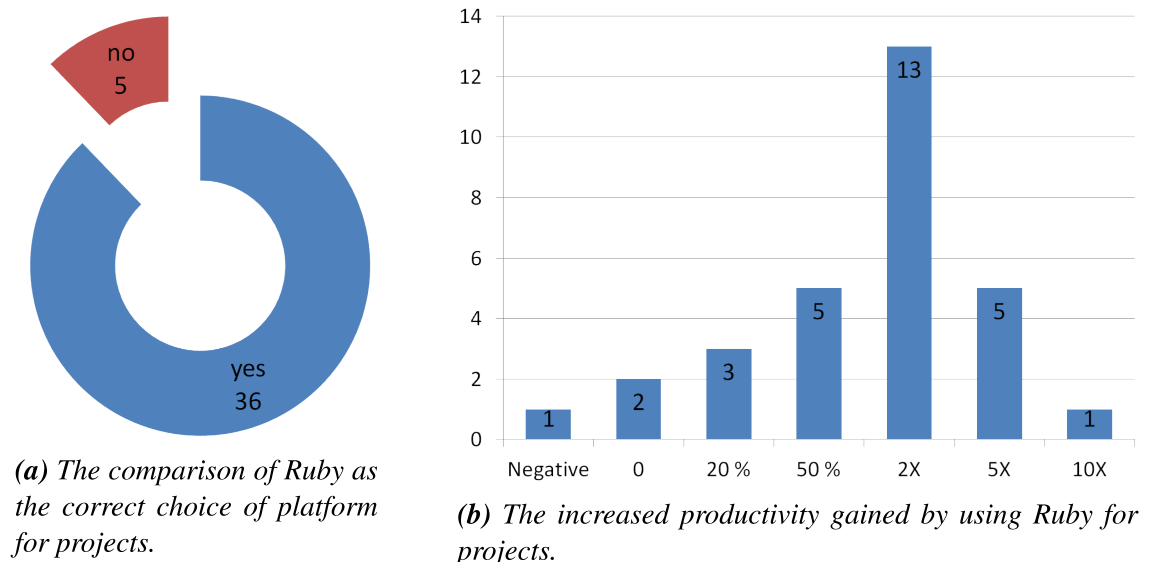
The common nature of web projects is that the web interface is likely to change more often than the services it is built upon. This indicates that a dynamic programming language is a good choice since it offers faster turnover. This is further enhanced in agile methodologies which involve, for example, rapid prototyping [50]. This model allows the web interface to work as a mediator between the request and the core functionality located on the server-side providing the heavy lifting.

Advanced frameworks speed up the productivity, but sometimes at the cost of performance. Although the decrease in performance is not that important in web development to some extent. This is because the bottleneck is usually the users Internet connection and not the application performance. It does not matter if the application uses 100 ms instead of 10 ms, as long as it takes 1 second to send the data. [23]

المنارة للاستشارات
www.manaraa.com

Polyglot programming is only useful if it gives an advantage. For example, an overhead caused by XML file parsing can be removed if a programming language with a literal XML support like Groovy or Scala is used instead of a general-purpose programming language with an XML parser. Groovy and Scala support writing XML directly within the programming language's syntax, providing more natural way of data interaction using the familiar dot notation. The same principle works also with Groovy and JavaScript when handling JavaScript Object Notation (JSON) messages. In addition, there has been a recent rise in next-generation NoSQL data storages with JSON-based protocol like MongoDB and FleetDB that provide implementations in several programming languages.

Global consulting firm ThoughtWorks started 40 % of their projects in 2007 in US with Ruby [51]. In three years period, Ruby was used in 41 projects, most of them web site projects which included Ruby on Rails framework, and the use of Ruby was evaluated as success (Figure 2.2). Although the results are subjective, they reflect the fact that different programming languages can be more productive. Thus Ruby became an important platform beside the major Java and .Net platforms at ThoughtWorks [52]. In addition, polyglot programming is considered to be a part of a growing opportunity worth over 35 billion dollars by 2015 [53].



(a) The comparison of Ruby as the correct choice of platform for projects.

(b) The increased productivity gained by using Ruby for projects.

Figure 2.2: *Subjective, qualitative assessments of Ruby projects at ThoughtWorks between 2006 and 2008. [52]*

Polyglot programming might present some considerable drawbacks depending on tool support and programming languages used. For example, refactoring a program written using a dynamic programming language is difficult, because the IDE cannot resolve the type information of a given variable since it is only revealed at runtime [54]. While developers could use automatic refactoring tools for Java parts, they would manually need to refactor Ruby parts.

المنارة للاستشارات
www.manaraa.com

2.5.2 Testing

Testing is a key concept in many of the new agile development methodologies, including test-driven development (TDD), behavior-driven development (BDD), and extreme programming (XP). The recent increase in popularity of these methodologies have also made automated testing more popular. [55; 56; 57]

Testing is a good way to introduce polyglot programming in software development because the test code is not an integral part of the application. Testing is a continuous activity that should cover the whole application. Automated testing gives developers the confidence to rely on the implementation to work after changes in the code. This is especially critical in dynamic programming languages which lack of static type safety and compile cycle. In addition, polyglot programming does not imply discarding what has already been implemented. The benefits from better suited programming languages can be leveraged even in existing infrastructures.

Testing complex code is a common task which can benefit from polyglot programming. When the code relies on databases and web services the consumption of time increases dramatically, and small changes in database might brake all tests. As a solution, mock and stub objects are created to mimic the behavior of other objects in purpose to test in isolation [58]. Creating mock object expectations can be time consuming due to Java's lack of flexibility required to allow objects to mimic others.

Ford [22] suggests writing the tests, and in this case only the tests, using a more suitable programming language. The example Java code shown in Program 2.1 uses a popular mock object library *JMock* [59] to test that the `Order` class interacts correctly with the

```
1  public class OrderInteractionTester extends MockObjectTestCase {
2     private static String TALISKER = "Talisker";
3
4     public void testFillingRemovesInventoryIfInStock() {
5
6        Order order = new OrderImpl(TALISKER, 50);
7        Mock warehouseMock = new Mock(Warehouse.class);
8
9        warehouseMock.expects(once()).method("hasInventory")
10          .with(eq(TALISKER),eq(50))
11          .will(returnValue(true));
12       warehouseMock.expects(once()).method("remove")
13          .with(eq(TALISKER), eq(50))
14          .after("hasInventory");
15
16       order.fill((Warehouse) warehouseMock.proxy());
17
18       warehouseMock.verify();
19       assertTrue(order.isFilled());
20    }
21
22 }
```

Program 2.1: *JMock test for correct interaction between the* `Order` *class and the* `Warehouse` *class. [22]*

www.manaraa.com

Warehouse class through its interface. Test verifies that the proper methods are called and that the result is correct.

The same example is given in Program 2.2 using a powerful mock object library *Mocha* [60] for Ruby, and thus JRuby programming language on the Java Virtual Machine. The latter implementation is much more concise and fluent due to dynamic nature of the programming language. In addition, the interfaces (Warehouse) can be directly instantiated because JRuby wraps Java objects in proxy classes. Also all the pertinent Java classes on the tests classpath can be easily imported in JRuby by requiring the all-inclusive JAR file require "Warehouse.jar". [22]

```
1  class OrderInteractionTest < Test::Unit::TestCase
2    TALISKER = "Talisker"
3
4    def test_filling_removes_inventory_if_in_stock
5      order = OrderImpl.new(TALISKER, 50)
6      warehouse = Warehouse.new
7      warehouse.stubs(:hasInventory).with(TALISKER, 50).returns(true)
8      warehouse.stubs(:remove).with(TALISKER, 50)
9
10     order.fill(warehouse)
11     assert order.is_filled
12   end
13
14 end
```

Program 2.2: *Mocha test for correct interaction between the* Order *class and the* Warehouse *class. [22]*

The runtime performance of the programming language used in testing is not important because the test code will not be run in production. Therefore, developer's productivity transcends the runtime performance. The developer productivity can be increased, for example, by using interpreted languages, because they enable tests to be run without compilation.

2.5.3 Deployment

Software deployment is the process of making a software available for use. The general deployment process is a series of interrelated activities and possible transitions between them. These activities may take place both at the producer site or at the consumer site.

Deployment should be interpreted as a general process which is customized according to specific requirements and characteristics of every software system. The variability and complexity of software systems makes them unique. Deployment process contains activities like release, installation and activation, deactivation, adaptation, update process, automated built-in activities, version tracking, uninstallation, and software system retirement.

Software systems are deployed on a software platform. The most commonly used software platforms are the .Net and the Java platforms, both of which support polyglot

المنارة للاستشارات
www.manaraa.com

programming extensively. Software platforms dictate which programming languages can be used in software development. In addition to traditional software platforms, there exists several cloud application platforms which provide their own deployment processes and methodologies.

2.5.4 Concurrency

Multicore processors introduced a new challenge for developers, namely how to best utilize these processors with multiple cores using the existing tools. Concurrency has proven to be one of the hardest aspects of imperative and object oriented programming. Concurrency is done using threads which introduce race conditions and risk of deadlocks due to different threads modifying the same variables. Since multicore processors entered the mainstream in software development, the concurrency shifted from being a problem of some developers into an important aspect for all developers. Therefore, managing threads and concurrency should be made easier. [61]

Some of the issues in concurrency can be resolved by shifting programming paradigm. Functional programming does not involve any use of variables as opposed to imperative programming. Since there exists no states to change and there are no shared states between processes, no function can have any side effects, and no data is shared directly, thus risk of deadlocks is removed. This allows referential transparency, meaning expressions can be evaluated in any order, which makes it easier to implement concurrent and transactional software [62; 63].

Erlang [64] is a functional programming language designed for concurrency. It achieves data sharing using message passing which is similar to communication between people, which can be considered a more natural programming idiom [65]. Ghodsi [66] benchmarked *Yaws* [67] which is a web server completely written in Erlang against *Apache* [68] web server, Apache died at about 4000 parallel sessions while Yaws still functioning with more than 80000 parallel sessions [66]. While no benchmark can go unchallenged [69]. Erlang is suitable for the problem domain. Erlang's approach to concurrency is to start a very light weight Erlang process for each state machine requiring concurrency. This approach is more natural in terms of implementation than thread pools, asynchronous input and output, or thread per connection systems. [70]

Functional programming approach is suitable when in need for concurrency, however not when developing user interfaces. Polyglot programming allows parts of the system to be implemented with concurrent functional programming languages and other parts with more suitable general-purpose programming languages. Facebook used this approach in the implementation of their chat client, by integrating existing infrastructure written in C++, PHP and JavaScript with Erlang chat client [71].

Facebook also developed the Apache Thrift [72] software framework to expedite development and implementation of efficient and scalable cross-language back-end services.

المنارة للاستشارات
www.manaraa.com

Primary goal was to enable efficient and reliable communication across different programming languages. Thrift combines a software stack with a code generation engine to build services that work efficiently and seamlessly between C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, JavaScript, Node.js, Smalltalk, OCaml, Delphi, and other programming languages. [73]

Web servers provide different approaches to concurrency in web development. Apache is the most used web server in the world holding more than half of the market (Figure 2.3). Although the main design goal of Apache is not to be the fastest web server, it performs similarly to other high-performance web server. Apache provides several approaches to concurrency by implementing multiple architectures to better match the demands of each particular infrastructure. It provides a variety of MultiProcessing Modules (MPMs) which allow Apache to run in a process-based, hybrid (process and thread) or event-hybrid mode [74]. This implies that it is important to choose the correct MPM and configuration. Apache is designed to reduce latency and increase throughput by ensuring reliable and consistent request processing in reasonable time frame.

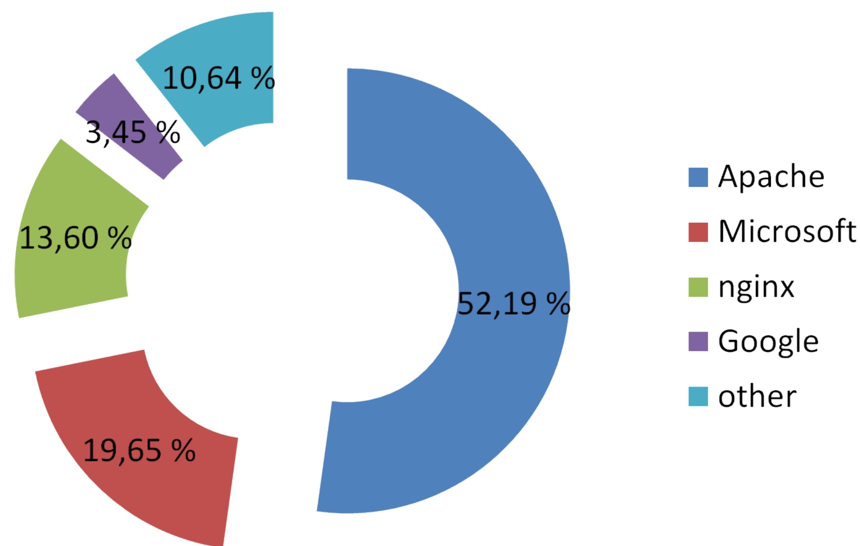


Figure 2.3: *Market share of web server developers based on all sites in July 2013. [75]*

The *nginx* [76] web server processes requests using an asynchronous event-driven approach, instead of the Apache web server model with threaded or process-oriented approach. The nginx outperformed the previous Apache 2.2 series for delivering static pages, although Apache was found to be significantly faster for delivering dynamic pages. The Apache Foundation decided to address this issue by providing a high-performance multithreaded version which uses several processes and several threads per process [74]. This new architectural approach was implemented in the current Apache 2.4 series, which now provides equivalent or even slightly better performance than other event-driven web servers [77].

المنارة للاستشارات
www.manaraa.com

Better concurrency will improve the scalability of the system, because the overhead of spawning new processes is decreased. The independent nature of the processes implies that the implementation of load distribution on multiple cores and machines becomes easier, and thus the hardware is better utilized. [23]

2.5.5 Business rules

Business logic is an integral part of the implementation source code in most modern-day applications. This makes verifying the correctness of the implemented business rules difficult for the domain experts, which also hinders the process of chancing the business rules, related to quite frequent changes in business, difficult for domain experts. [23]

Fields [78] describes a *Business Natural Language* (BNL), a subset of domain-specific languages, as a solution to implement business rules. BNL languages are designed to implement a domain vocabulary similar to the language used by domain experts, thus they could verify and maintain the business rules themselves. Therefore, the intent of the application would be clearer for the domain experts because they understand the rules [79]. [78]

Rules engine is an alternative for BNL. The advantage of rules engine is that there is no need to implement a parser for the language, but on the other hand the rules engine restricts the expressiveness by forcing the syntax. There exist several rules engines for different platforms. Drools [80] and Jess [81] are popular rules engines for the Java platform, InRule [82] for the .Net platform, and BizTalk Server [83]. Programming languages that follow logical programming paradigm are also rules engine languages.

2.6 Polyglot software systems

Fjeldberg [23] propose a degree of polyglotism – usage of multiple programming languages – in an application to differentiate the use of polyglot programming. Proposed levels of polyglotism are integration, organization of code, the processes that run the programming languages, and the data being manipulated. Integration is either networked or non-networked, the organization of code distinguish that is the code written inside same or different files. Either the same or separate processes are used to run the different languages, and the languages manipulate either the same object or the same data. Table 2.1 describes the levels of polyglot programming for different architectures.

Table 2.1: *Levels of polyglot programming in different polyglot systems.*

Architecture	Integration	Organization	Process	Data/object
SOA	Networked	Different files	Different	Same data
Managed runtime	Non-networked	Different files	Same	Same object
HTML++ server	Non-networked	Different files	Different	Same data
HTML++ client	Non-networked	Same file	Same	Same object
CI system	Non-networked	Different files	Different	Same data

المنارة للاستشارات
www.manaraa.com

Example architectures for utilizing polyglot programming are service-oriented architecture (SOA), managed runtime, continuous integration (CI) system and embedded polyglotism where different languages are presented in the same file. HTML in conjunction with CSS, JavaScript and a server-side language is an example of a polyglot program (referred as HTML++ for abbreviation).

2.7 Polyglot programming pyramid

Polyglot environment has two essential aspects, the platform used for the integration and the different programming languages supported by the given platform. The possibility to create a new infrastructure without need to rewrite old legacy code has proven essential [84; 85]. The recent development on software platforms is one of the reasons why polyglot programming has the means for success.

Polyglot programming pyramid is used to describe and categorize the programming languages and specification languages used in a polyglot software system [13; 86; 87]. Polyglot programming pyramid presents three potential layers for different programming languages as shown in Figure 2.4 [13].

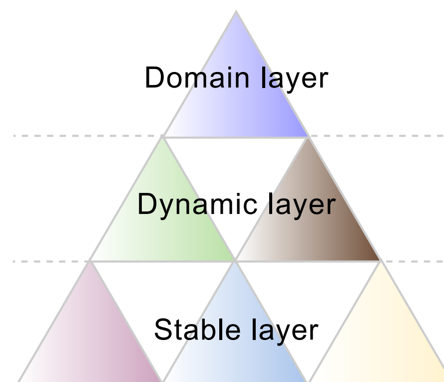


Figure 2.4: *Polyglot programming pyramid.*

Statically typed powerful programming languages tend to gravitate towards the needs of the stable layer. The less powerful general-purpose technologies tend to ascend to the top layer, and the dynamic layer in the middle consist a rich variety of programming languages with the most flexibility. However, in many cases the dynamic layer tends to overlap with the adjacent layers. In addition, the three layers of the polyglot programming pyramid represent polyglot programming as a form of separation of concerns, where the layers represent patterns as described in Table 2.2.

The layers presented in the polyglot programming pyramid are organized with the stable layer as a wide base [87]. Bini [13] argues that the domain-specific layer should be the largest and that the dynamic layer quite often includes more than one programming language. The polyglot programming pyramid can be inverted to present the stable layer

المنارة للاستشارات
www.manaraa.com

Table 2.2: Three layers of the polyglot programming pyramid.

Layer	Description	Examples
Domain-specific	Domain-specific language; tighly coupled to a specific part of the application domain	HTML, CSS, Web templating, SQL
Dynamic	Rapid, Productive, Flexible development of functionality	Groovy, Clojure, Jython, JRuby, JavaScript
Stable	Core functionality, stable, well-tested, performant	Java, Scala

as the tip of the pyramid providing the base as shown in Figure 2.5. This allows the dynamic layer in the middle to be divided into smaller parts, for example, based on programming language or functionality. Also the different domain-specific languages are now represented as smaller pyramids standing upside down. Bini [13] names this strategy as bounded fractal representation. [13]

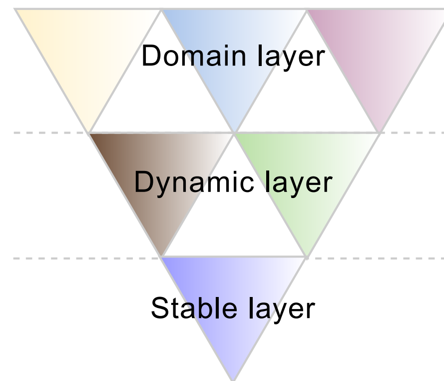


Figure 2.5: Polyglot programming pyramid is inverted to reflect the amount of used programming languages in each layer.

The bounded fractal representation forms the polyglot system which consists of these smaller pyramids. The pyramids have no restrictions, and they can all be the same programming language and system, or multiple different ones. Organization of the smaller pyramids depends heavily on the application or system being developed. Figure 2.6 represents an example polyglot system with a combination of Clojure, Scala and JavaScript. Any imaginable combination can be used, although it is important to keep in mind that the combination should be best suited for the problem at hand [13].

Domain layer defines the actual domain rules, which in general means using one or more domain-specific languages. This model sees the DSLs as the same layer regardless if they are internal or external. Domain layer is a part of a system which needs to be adaptive enough, so that it is possible to change rules in production. Domain layer should allow domain experts to manage it. The programming languages in domain layer

المنارة للاستشارات
www.manaraa.com

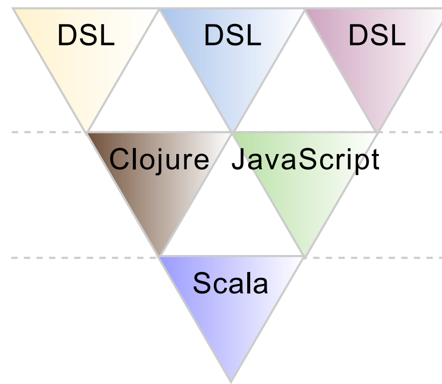


Figure 2.6: Bounded fractal representation of an example organization of a polyglot system with a combination of Clojure, Scala and JavaScript.

are mostly external DSLs like HTML and SQL, but it can also include general-purpose programming languages like Groovy, Ruby or Python which support building internal DSLs [23; 79]. [13]

Ford [21] argues that the dynamic layer is more about essence and not so much about dynamic. Problem is that even with a programming language like Scala which is usually classified as an essential programming language, it requires compilation. Bini [13] considers compilation as a ceremony, meaning that it is one extra thing developers do not want to care about when writing most of the application code. This is the main reason why this layer needs to be dynamic. The dynamic layer includes programming languages like Clojure, Groovy, Ruby, Python and JavaScript. [13]

Stable layer is the core set of axioms, the hard kernel or the thin foundation that the system can be developed in. It thrives for performance with static type checking, although there would be advantages having this layer written in an expressive programming language. The main idea of this layer is to make the trade-off in giving up static typing smaller. The stable layer on the bottom provides the resources and services for dynamic layer to utilize. [13]

Stable layer encapsulates another important feature, it is where all external APIs are defined. The performance of the APIs need to be solid since other clients rely on them. This way the implementations for the APIs live in the dynamic layer, not in the stable one. This approach allows developers to take advantage of static type information for APIs while retaining full flexibility in implementation. The stable layer provides the necessary services needed for everything to function, and it should be fairly small compared to the rest of the application. Stable layer include programming languages like Java and Scala. [13]

المنارة للاستشارات
www.manaraa.com

2.8 Extending the polyglot programming pyramid

Polyglot programming pyramid represents the programming languages used in a polyglot system. Stable layer at the bottom represents the implementation of the core functionality and is usually written with a single programming language, for example, to provide efficient Java Hibernate database interaction. Stable layer provides its services for the dynamic layer to use, thus benefiting from static typing and good type safety. Dynamic layer is to maximize productivity in web development by using several programming languages side by side, for example, Groovy and JavaScript. Domain layer represents domain-specific programming and specification languages which provide solutions and techniques to certain problem domains, for example, HTML for markup, CSS for presentation semantics, and SQL for managing data.

2.8.1 Improving the bounded fractal representation

The bounded fractal representation by Bini [13] has been criticized to represent only a layered structure of a polyglot software system instead of a fractal structure [13]. Fractal typically involves a self-similarity pattern, where fractals are the same from near as from far [88]. The definition of fractal states that fractals exclude trivial self-similarity and include the idea of a detailed pattern repeating itself [89]. Thus the idea of polyglot programming pyramid representing different programming languages in a layered manner requires improvement.

Programming environment is nominally the programming language and more so frameworks, libraries and tools. Excellent frameworks and libraries are the main reason for the increased popularity of using polyglot programming in web development. The fractal representation can thus be improved by introducing frameworks and libraries, and more so by assessing their contents. Although some principles of the layer separation has to be neglected when going deeper into the levels of fractal representation. Therefore, the separation of programming languages to stable and dynamic layer based on compilation, and static versus dynamic typing, is to be considered only when representing solely programming languages of a complete polyglot system.

Consider an example web development project shown in Figure 2.7 with a stable layer providing a well tested Java Hibernate database interaction as a core functionality. The dynamic layer is used for rapid web development to provide fast turnover. Therefore, the front-end of the application is implemented using Groovy and its high-productivity web framework Grails which embraces the convention over configuration paradigm. In addition, JavaScript and its powerful jQuery library is used to provide enhanced and more dynamic web content. The domain layer is used to identify specific areas of the application like the concatenated HTML markup and CSS presentation semantics, and SQL used in collaboration with Hibernate.

المنارة للاستشارات
www.manaraa.com

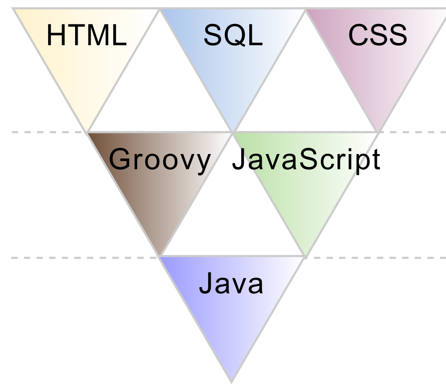
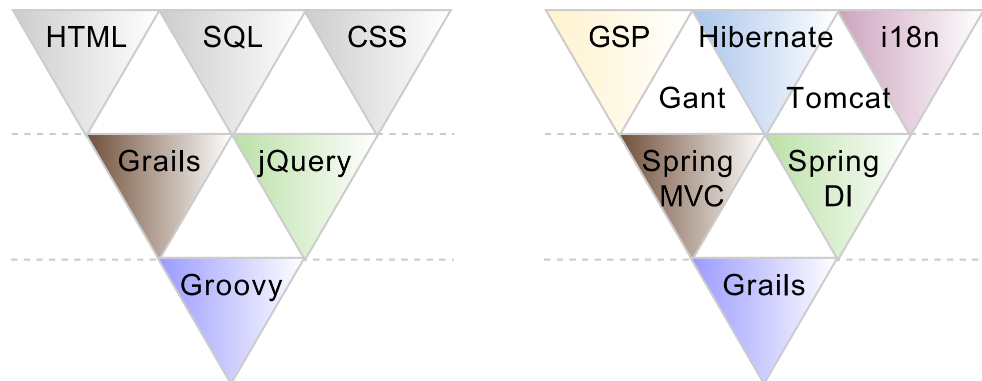


Figure 2.7: Polyglot programming pyramid of an example project.

Polyglot programming pyramid by Bini [13] clearly lacks information in documentation of an polyglot system, because only programming languages are represented. Important decisions like using Hibernate in conjunction with SQL for object-relational mapping and database interaction, or Grails framework for rapid web development is not documented. In addition, the domain layer gives only a vague abstraction with none or little value by specifying DSLs like HTML and CSS. Therefore, by including frameworks and libraries as shown in Figure 2.8a, and opening them up when necessary as shown in Figure 2.8b, it is possible to get much more detailed information about the polyglot system and its structure.



(a) Including frameworks to polyglot programming pyramid.

(b) Polyglot programming pyramid opening up frameworks.

Figure 2.8: Improving polyglot programming pyramid fractal representation by supporting and opening up frameworks.

Thus the enhancement on the polyglot programming pyramid makes it possible to document polyglot software systems more specifically. In addition, the separation of concerns is utilized in a deeper manner.

المنارة للاستشارات
www.manaraa.com

2.8.2 Supporting architectural decision making

Patterns capture existing, well-proven experience in software development. They promote good design practice build on the collective experience of skilled software engineers. Patterns are used to construct software architectures with specific properties dealing with certain, recurring problem domain in the design or implementation of a software system. Experts working on a particular problem, often tackle it by recalling a similar problem they have already solved. The essence of the previous solution is reused to solve the new problem at hand. It is quite uncommon to invent a new approach completely distinct from existing solutions. [90; 91; 92]

Web softwares are prone to change requests, whether updating the look and feel or extending the functionality of an application. Even upgrading to a new release of used frameworks can imply changes to the source code of an application. Web development is usually fast phased and the application is always evolving. Building a software system with the required flexibility may prove to be ineffective and laborious when using a single programming language. The selected programming language will dictate the programming paradigm and idioms, thus restrict to certain approaches.

Web development allows and practically recommends to embrace several different programming languages. Different programming languages are used to solve different problem domains of the web application. Best practices and front-end code standards separate the presentation, content and behavior of the software system. In addition, web applications usually follow the three-tier client-server architecture in which the user interface, business logic, data access are developed and maintained as independent modules.

When developing such an interactive and polyglot software system the changes to the different layers of the polyglot system should be easy, and non-interactive to adjacent layers. To solve the problem, the tree layers of the polyglot programming pyramid further represent polyglot programming as a form of separation of concerns. Therefore, the polyglot programming pyramid can be considered as an architectural pattern to support decision making and documentation when developing polyglot software systems using several programming languages.

2.9 Guidelines for polyglot programming

After making the decision to experiment with multiple programming languages, the project should be revised against the layers of the polyglot programming pyramid [93]. Table 2.3 highlights project areas suited for these three layers in a project. Identifying the scenario that could be resolved with an alternative programming language is just the beginning. It is more important to evaluate whether utilization of an alternative programming language is expedient. Evans and Verburg [87] propose five useful criteria to question when evaluating a technology stacks. Risk exposure of the project area should be revised and the ease

المنارة للاستشارات
www.manaraa.com

of interoperability with Java considered. Tooling support for an alternative programming language should be charted. The learning curve for a new programming language should be evaluated and the ease of hiring experienced developers in that alternative programming language considered. [87]

Table 2.3: *Project areas suited for domain-specific, dynamic, and stable layers.*

Layer	Example problem domains
Domain-specific	Build, Continuous Integration, Continuous Deployment, Devops, Enterprise Integration, Pattern modeling, Business rules modeling
Dynamic	Rapid web development, Prototyping, Interactive administrative and user consoles, Scripting, Tests (such as for test- and behavior-driven development)
Stable	Concurrent code, Application containers, Core business functionality

Consider a stable piece of core payment-processing Java software handling millions of transactions per day. Furthermore the code has plenty of dark corners with low test coverage. This reveals definitely a high-risk area for a new language to be added, especially due to the lack of test coverage and a pool of developers who understand it in detail. But in consideration of a complete system, there is more into it than just the core processing. This situation clearly cries out for better tests and test coverage which in contrast is low-risk. Scala with its supreme ScalaTest framework would remove the boilerplate generated with JUnit still enabling to produce familiar JUnit-like tests. Improved developer productivity and test coverage would be achieved after the initial learning curve. ScalaTest also provides a way to introduce behavior-driven development process. Comprehensive and advanced testing comes extremely pragmatic when the core needs refactoring or functional changes, whatever the chosen programming language is. [87]

Another low-risk area for a programming language try out would be developing a noncritical web console for administering the static data behind the payment-processing system. The development team already knows Struts and JSF but do not have any enthusiasm for either technology. An obvious choice would be Grails web framework which is backed up by a developer buzz and recent studies suggesting it as the best-available web framework for productivity [94]. Focusing on a limited pilot in a low-risk area would allow ease of termination of the project or switching the technology stack without much disruption. [87]

One of the main reasons why organizations hesitate to introduce a new programming language into their technology stack is the fear of loosing the existing value of all previously written Java source code. In contrast, by selecting an alternative programming language that runs on the Java Virtual Machine (JVM) it turns out to be about maximizing the existing value of the codebase without discarding any working code. By using an

المنارة للاستشارات
www.manaraa.com

alternative programming language on the JVM as a part of the system, the expertise of those developers can be used in supporting the existing environment. This can be used to help reduce risk by alleviating any worries that production managers might have about supporting the new solution. [87]

Good development environment is often taken for granted since Java developers have for years benefited from great tooling support due to its maturity. Most developers underestimate the time they save once they get comfortable with the powerful integrated development environments and build and test tools, which enable rapid development to produce high quality software. Some of the alternative programming languages like Groovy have also had longstanding IDE support for compiling, testing and deployment. However, not all of the alternative programming languages have reached the same level of maturity when it comes to the development environment support. Despite that, for example, the fans of Scala feel that the power and conciseness of the programming language itself more than makes up for the imperfections of the unpolished generation of IDEs. Another related issue appears when alternative programming languages develop powerful tools only for their own use. These tools might not be well adapted to another programming languages, for example, the powerful build tool Leiningen for Clojure. This reveals a need for careful consideration when dividing up the project, especially in deployment of distinct but interoperable components. [87]

Even though a pragmatic developer should learn a new programming language per year [42], it is always time consuming [45; 46], and more so when introducing a new paradigm. Most Java developers are familiar only with the object-oriented architecture and C-like syntax which would benefit in favor of Groovy. When shifting a paradigm away from the familiar the learning curve gets definitely steeper. Scala as a hybrid programming language is an example in bridging the gap between object-oriented and functional programming paradigms. An extreme alteration from the object-oriented Java towards completely functional Clojure with Lisp like syntax may represent substantial retraining requirements for the developers. A viable alternative is to introduce JVM incarnations of existing well-established programming languages like Ruby and Python, providing thorough insight into a non-Java programming language interpreted on the JVM. However, reimplemented programming languages might pose a threat since many existing packages and applications are only tested against the original implementation. [87]

Similarly to developers, organizations have to be pragmatic. Inside an organization, the developers and development teams change during the course of a year. Thus the programming language choices reflect directly on the developers working but also on to the hiring process. When programming language enjoys a well-established social and technical infrastructure this is not a problem, since there will be lots of developers with enough knowledge. Programming languages like Ruby and Python offer a large pool of developers that can readily adopt into their JVM reincarnation.

المنارة للاستشارات
www.manaraa.com

2.10 Poly-paradigm programming

Sebesta [95] classifies programming languages according to programming paradigms. Different programming paradigms have distinct strengths and weaknesses, and domain areas where the programming paradigm is best suited.

The *imperative programming paradigm* provides a good performance, because it offers little abstraction. The lack of abstraction is also one of the limitations of the paradigm, because without abstraction, the management and organization of a large software system becomes hard. In imperative programming, the execution of the instructions is done in order of appearance [61]. The imperative paradigm follows the fundamental computer architecture of the von Neumann-Eckert model, meaning both data values and program instructions are stored in memory, thus almost all programming languages possess imperative properties.

The *object-oriented programming paradigm* focuses on object-oriented decomposition instead of data abstraction and functional decomposition. Objects are represented using a class concept, which encapsulates constants, variables and functions, also supporting inheritance, visibility and information hiding [61]. The higher abstraction provides a more natural organization of related aspects, which is especially beneficial in large software systems, although at the cost of performance.

The *functional programming paradigm* adopts mathematical thinking, and shifts the focus from how something should be computed to what should be computed [62]. Recent interest in functional programming is caused by the emerge of multicore processors, thus requirements for better concurrency. Functional programming does not involve any use of variables as opposed to imperative programming. It considers everything as a function with an input and a result. Functions interact with each other through functional composition, conditionals and recursion [61]. In addition to concurrency, other important characteristics of functional programming include lazy evaluation and higher order of functions. Lazy evaluation ensures that functions are called and values evaluated only when necessary, therefore infinite expressions can be created and evaluated using only the values needed. Higher order functions provide the basis of functional programming by allowing functions to be passed on, in addition to values, to other functions [63]. The major problem with functional programming is that it introduces a steep learning curve for developers accustomed to object-oriented programming languages, because completely different mindset is needed. Although many of the new versions of object-oriented programming languages provide hybrid functionality by mixing object-oriented paradigm with functional paradigm.

The *concurrent*, *parallel*, and *reactive programming paradigms* are increasingly important paradigms. Asynchronous message-passing can be used to implement elegant communication in a client-server application or to parallelize computation. Functional

المنارة للاستشارات
www.manaraa.com

nature of programming languages facilitate the writing of concurrent, parallel, and reactive programs. [96; 97]

The *logical programming paradigm* adopts declarative thinking, and shifts the focus from declaring how something should be accomplished to what should be accomplished. Logical programming often gives a collection of assertions, or rules about the constraints and outcomes, thus it is also called rule-based programming. Logical programming has two distinct properties which are nondeterminism and backtracking. Nondeterminism allows multiple solutions to a problem, and backtracking means that the decisions can be reproduced and reasoned about [61]. The ability to specify what should happen enables the machine to decide how to accomplish the task, and optimize the performance. The disadvantage of logical programming is that the paradigm is very specialized and somewhat limited to the field of artificial intelligence and database information retrieval [61].

A programming language can be either *statically* or *dynamically typed* independently of programming paradigm. Programming language that is statically typed has to declare the types of all variables before compilation and the types cannot change, while dynamically typed programming languages declare the types at runtime and the types can change. An exception is typecasting, which is checked at runtime in many programming languages [61]. Statically typed programming languages usually provide better performance because the compiler can make optimization based on the known type. Dynamically typed programming languages are often interpreted, thus they allow new features to be tested without laborious compilation. Because of the type of the variable is implicit in dynamically typed programming languages, the implementations require less typing in contrast to statically typed programming languages where the type of each variable must be specified. However, dynamically typed programming languages add these features at the cost of type safety and performance.

Software development and domain engineering uses *domain-specific languages* (DSL), which are a type of programming languages or specification languages which are dedicated to particular problem domains, representation techniques, and solution techniques. DSLs are in extensive use in web development and examples include CSS, regular expressions and Ant [79]. DSLs are also essential in language oriented programming [19], and apparent in the polyglot programming pyramid representation [13; 14; 15]. Since DSLs are designed specifically for the problem domain, they increase productivity. However, there is a huge extra effort needed when creating a new domain-specific language.

Scripting languages are programming languages that supports the writing of scripts. A script is a program that automates the execution of tasks which could alternatively be executed one at a time by human operator [98]. Scripting is used, for example, to automate build and compilation process of software applications, web page interactions within web browsers, and in several general-purpose and domain-specific programming languages. Scripting languages offer ease of use through relatively simple syntax and

المنارة للاستشارات
www.manaraa.com

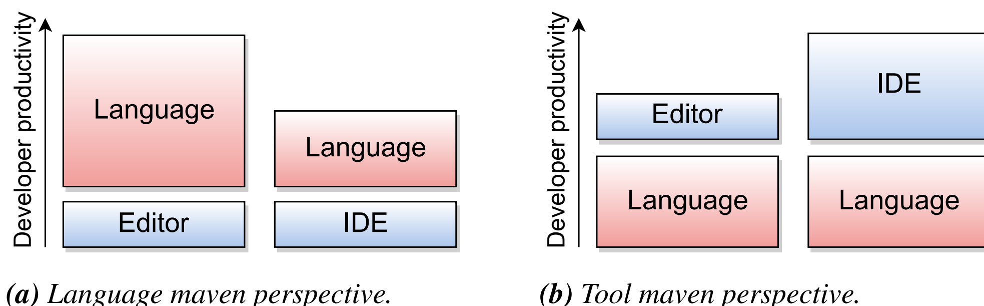
semantics, and good operating system facilities with built-in interfaces. The source code is interpreted to provide fast turnaround from script to execution [99]. In comparison, non-scripting programming languages are usually compiled for superior performance. The vast spectrum of scripting languages varies from general-purpose programming languages to very compact and highly domain-specific languages.

2.11 Programming language features and tool support

Steele [49] proposes that all developers are divided into two categories known as *language mavens* and *tool mavens*. A maven refers to a trusted expert in a particular field, who seeks to pass knowledge on to others. Language mavens are enthusiastic about the power and possibilities of higher-level programming, for example, first-class functions, staged programming, aspect-oriented programming, reflection et cetera. Tool mavens are vastly skilled with integrated development environment features like build and debug tools, integrated documentation, code completion, refactoring and code comprehension. Whereas language mavens tend to use more unsophisticated text editors which are more likely to work with new programming languages and features. [49]

A new programming language typically lacks in support from integrated development environments and is usable only with plain text editors. Whenever a programming language or a new feature proves to be successful and establishes a solid ground, the tools catch up. Steele [49] argues that programming language expertise and tool expertise are alternatives to certain extent, since both tend to strengthen themselves to the exclusion of the other.

Figure 2.9a shows the developer productivity from the language maven perspective. The choice of a programming language can make a huge difference, because language maven knows how to apply each programming language feature to a variety of situations. The IDE on the other hand does not reflect that much on the productivity, since it is used mainly as a plain text editor with few robust features like compilation support. [49]



***(a)** Language maven perspective.*   ***(b)** Tool maven perspective.*

***Figure 2.9:** Different perspectives on developer productivity.*

A tool maven has an inverse perspective on developer productivity shown in Figure 2.9b. A tool maven spends most of his time mastering the development tools and enjoy

المنارة للاستشارات
www.manaraa.com

the occasional developer's flow state of working with an integrated editor and debugger with application scope refactoring and code comprehension capabilities at disposal. An IDE provides a wide variety of advantages compared to a simple plain text editor. Therefore, the choice of a programming language as long as it is supported by IDE matters less. A tool maven is likely to work with the same classes and methods, statements and expressions where the real development power comes from the IDE and other development tools. [49]

Developers have limited time especially for learning new skills. Any allocated time can be used to master a programming language or to master the development tools. Both of which accumulate knowledge. A language maven gathers knowledge of and how to leverage from programming language features. A tool maven on the other hand familiarizes with the development tools to take advantage on their powerful features. Both of these have a positive feedback cycle, but Steele [49] argues that they are competing cycles, and thus divided categories.

Developers willing to learn programming language features are more likely to appreciate the features of new programming languages. These developers have the knowhow to adopt a new programming language before the development tools support it, and may do so, since their productivity does not rely on the tools. In addition, a language maven sees these new features worth to adopt early, because using them is where the expertise lies. [49]

In contrast, a developer who masters his development tools is likely to be unwilling to try a new programming language, because he could lose a major part of his productivity without the support of the development tools. In addition, a new programming language provides far less advantage over another programming language for a tool maven, because he does not have the knowhow of the features to enhance his productivity. [49]

Therefore, the more invested in learning programming language features, the bigger is the benefit, although to the exclusion of tool features and vice versa. Relative merits of programming language features and tool support divide the perspectives in two distinct categories shown in Figure 2.10. [49]

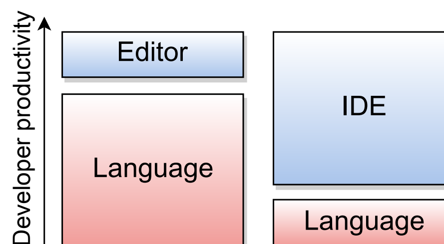


Figure 2.10: *Relative merits of programming language features and tool support divide the two categories.*

المنارة للاستشارات
www.manaraa.com

Fjeldberg [23] proposes that polyglot programmers are more likely to think of the programming languages as tools over the IDEs. Therefore, polyglot programmers are supposed to be language mavens.

المنارة للاستشارات
www.manaraa.com

3. POLYGLOT PROGRAMMING ON THE JAVA PLATFORM

The recent rise of non-Java programming languages on the Java Virtual Machine (JVM) has led to a cross-fertilization between Java and other programming languages on the JVM. The emergence of polyglot programming in projects involving programming languages such as Groovy, Scala and Clojure has been a major factor in the evolution of the current Java platform [100; 101].

As mentioned previously, polyglot environment has two essential aspects, the platform used for the integration and the different programming languages supported by the chosen platform. The recent development on the Java platform is one of the reasons why polyglot programming has the means for success.

Polyglot programming on the JVM is a relatively new concept. It is coined to describe software systems and projects that utilize a non-Java programming language on the JVM alongside a core of Java source code [87]. Although this is contradictory since polyglot programming should not restrict to any specific programming languages. Therefore, polyglot programming on the JVM describes software systems and projects that utilize more than one programming language on the Java Virtual Machine.

Java's nature makes it a prominent choice for implementing functionality in the stable layer. A mature general-purpose, statically typed and compiled programming language provides many advantages. Conversely these same advantages tend to become a burden in the upper layers. For instance recompilation is laborious, and deployment is a time consuming heavyweight process. Static typing can lead to long refactoring times due to its inflexibility and Java's syntax is not naturally fit for producing domain-specific languages.

The fact that the recompilation and rebuild time of a Java project quickly reaches the 90 seconds to 2 minutes mark will definitely break the developer's flow [87; 102]. In addition, it is a bad practice for developing code that may live only few weeks in the production. A pragmatic solution is suggested to take advantage of Java's rich application programming interface (API) and library support to do the groundings in the stable layer. Similarly if a particular feature such as a superior concurrency support is required a pragmatic choice would be to choose another stable layer language with such vantage like Scala. However, working stable layer code should not be discarded and rewritten in a different stable layer programming language [84; 85].

المنارة للاستشارات
www.manaraa.com

The distinction of duality between the programming language and the platform is a critical concept to comprehend. It is essential to distinct what constitutes the programming language and platform to understand how the polyglot programming on the Java Virtual Machine can thrive. The Java programming language is a general-purpose, concurrent, strongly typed and class-based object-oriented language. The human-readable Java source files are compiled by the `javac` Java compiler into bytecode class files which are not human-readable. The Java platform, on the other hand, is the software that provides a runtime environment, the Java Virtual Machine to link and execute the bytecode in the form of the class files.

3.1 Java platform

Java platform [103] is a software platform that provides a runtime environment, the Java Virtual Machine to link and execute the Java bytecode in the form of the class files. In technical terms of compiler theory the bytecode is a form of intermediate language rather than a true machine code. This implies that the process of turning Java source code into bytecode is really a class file generator. The actual compiler in the Java platform is the just-in-time (JIT) compiler which performs runtime compilation of the class files as shown in Figure 3.1. [87]

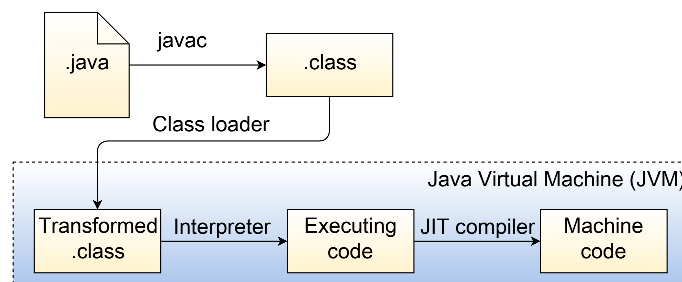


Figure 3.1: *Java source code is transformed into class files, which are manipulated at load time before just-in-time compilation.*

The most important specifications that influence the Java platform are the *Java language specification* (JLS) [104] and the *Java Virtual Machine specification* (VMSpec) [105]. Java 7 takes this distinction seriously by separating the VMSpec entirely from the JLS. This indicates how seriously the support for non-Java programming languages is taken on the JVM in Java 7.

Java Virtual Machine was originally built only for Java programming language but nowadays it allows code to be cross-platform. The link between the programming language and the platform is the shared format definition of the class files as shown in Figure 3.2. Java platform supports any other programming language as long as they are targeted to run on the JVM, and that they obey the same rules and idioms.

المنارة للاستشارات
www.manaraa.com

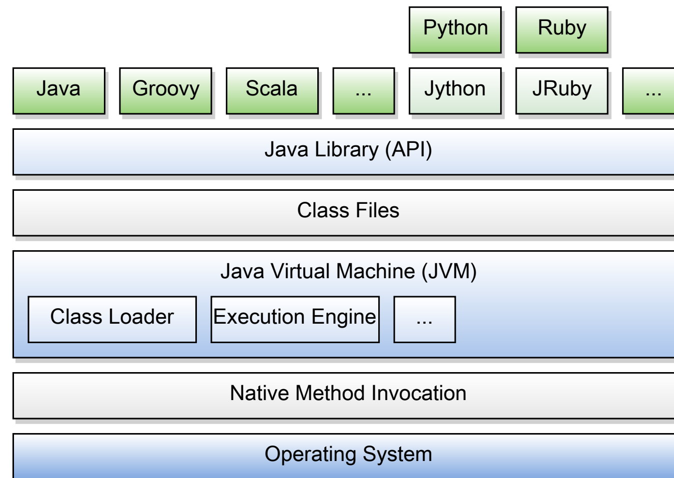


Figure 3.2: *The Java platform and programming language interoperability.*

Portability and the fact that JVM is available for almost any platform has led to its widespread use. This has constituted the continuous evolutions and that libraries exist for almost any task which has made JVM even more attractive and popular.

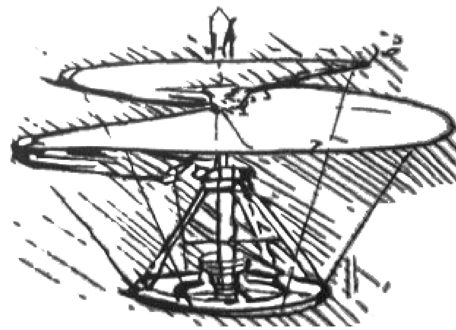
3.2 Evolution of polyglot programming

Java 6 release with JSR 223 [106] was a revolutionary step on the polyglot programming frontier. The JSR 223 presented scripting for Java platform to address the needs of Java community to take advantage of the benefits of the Java technology in a variety of programming and scripting languages. The specifications described a mechanism which allowed scripting language to be used in Java server-side applications and also allowed scripting language programs to access information developed in the Java platform.

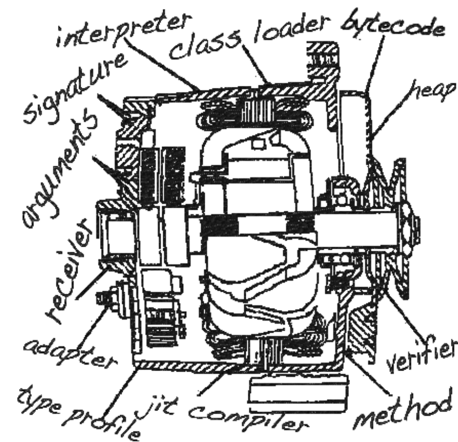
Java 7 introduces more evolutionary rather than revolutionary changes on the programming language. The major difference between previous releases is that Java 7 is explicitly released in preparation for the next version. The groundwork for major language changes following in Java 8 is contained in Java 7 release. Java 7 is also the first Java version developed in an open source manner. In addition, polyglot programming practically lives in open source [107].

Java community driven OpenJDK [108] introduced *the da Vinci Machine project* [109] as a multi-language renaissance for the Java Virtual Machine architecture (Figure 3.3). The idea was to extend the JVM first-class architectural support for programming languages other than Java, especially dynamic languages. The graceful co-existence of the new programming languages with Java on the JVM was pursued. The goal was to allow other programming languages to take advantage of the powerful and mature Java technologies. This was later on adapted and coined as JSR 292 [110] to pursuit polyglot programming.

المنارة للاستشارات
www.manaraa.com



(a) *The da Vinci Helicopter.*



(b) *The first JVM as da Vinci machine.*

Figure 3.3: *The da Vinci Machine project, a multi-language renaissance for the Java Virtual Machine architecture [109].*

The Java community had increasingly recognized the value of dynamically typed programming languages, especially scripting languages. Java 7 with JSR 292 added support for dynamically typed programming languages on the Java platform making them equal class citizens with Java. Programming languages such as Groovy, Scala, JavaScript, Python, Ruby, Perl et cetera running directly on the JVM facilitated their interoperability with the existing Java programming language. The key concept was to make the implementations of such programming languages more efficient and to make it easier to create such implementations. The JSR 292 added a new bytecode called invokedynamic to support efficient and flexible execution of method invocations in the absence of static type information. As of Java 7 all the programming languages on the JVM are now equal [111].

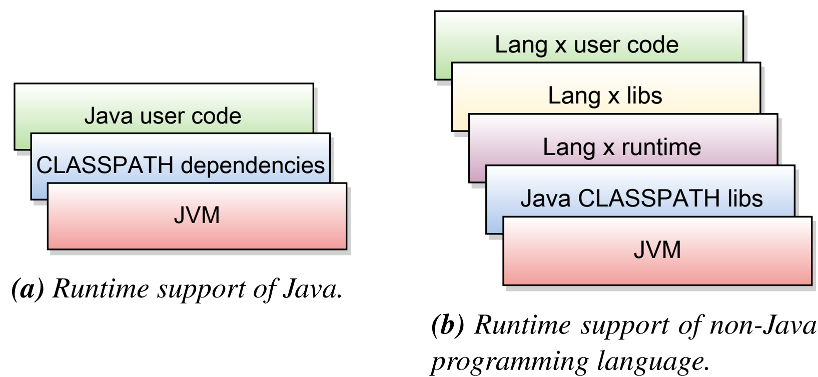
Some of the programming languages require interpreters to be written to be able to evaluate code written in other programming languages from the Java. For example JRuby is such an interpreter for Ruby and similarly Jython for Python. These interpreters enable the usage of Java libraries from the respective programming languages and also utilize the possibility to use best features of any supported programming language.

The postponed JSR 335 [112] in the upcoming Java 8 will introduce lambda expressions also known as closures for the Java programming language. The JSR 335 will enable the creation and consumption of easy-to-use, multicore-ready libraries to address the challenge posed by multicore processors. Closures are welcomed addition to the Java programming language since they remove much of the excess verbosity in Java source code. The closures will also benefit poly-paradigm programming because they allow Java programmers to deviate from the imperative object-oriented paradigm towards the functional paradigm.

المنارة للاستشارات
www.manaraa.com

3.3 Programming languages on the Java Virtual Machine

Java platform provides two possible ways of running different programming languages on the Java Virtual Machine. Either the programming language has a compiler that emits class files or that it has an interpreter which is implemented in Java bytecode. In any case, it is common to have a runtime environment that provides a programming language specific support for executing programs. The complexity of the runtime support for a programming language varies depending on the amount of guidance required at runtime. The implementation of the runtime environment support is a set of JAR files on the classpath of the program that are bootstrapped before the program execution starts as shown in Figure 3.4.



(a) *Runtime support of Java.*

(b) *Runtime support of non-Java programming language.*

Figure 3.4: *Runtime support of programming languages on the JVM.*

The need for a runtime environment support is required based on the fact that programming languages tend to differ even in basic programming concepts. The runtime environment is designed to help the type system and other aspects of a non-Java programming language to achieve the desired semantics. For example Ruby differs from Java in its object-oriented approach. In Ruby it is possible to have differently defined instances of the same class. Individual object instances can have methods included at runtime that were not known when the class was defined. This property called "open class" needs to be replicated with advanced support from the JRuby runtime.

The JVM supports functional programming through first-class functions. The key concept of functional programming can be stated as "functions are first-class values". This means that functions can be passed as variables into methods, and manipulated as if ordinary values. On the other hand, the JVM handles classes as the smallest unit of code and functionality. Therefore, a trick of creating an anonymous classes to hold the functions is needed. Java does not offer any special syntax for it, although this might change with Java 8 [87]. Programming languages like Groovy, Scala and Clojure all provide a special syntax for writing out these "function literals" or "anonymous functions", which is a major pillar of the functional programming style.

المنارة للاستشارات
www.manaraa.com

Multiple inheritance can not be expressed directly in Java or on the JVM. This can be done only through interfaces that do not allow any method bodies to be specified. The trait mechanism of Scala provides a different view of inheritance by allowing method implementations to be mixed into a class definition. The JVM has no provision for it and the behavior has to be synthesized by the Scala compiler and runtime.

3.3.1 Java

Java [113] is a general-purpose, statically typed, concurrent, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. Java derives much of its syntax from C and C++ though the object-oriented features are modeled after Smalltalk and Objective-C. Java follows the principle *write once, run anywhere* (WORA), meaning that no recompilation is needed regardless of platform changes.

Primary goals of Java are that programming language should be simple, object-oriented and familiar. Secondly it should be robust and secure. In addition, it should be architecture-neutral and provide high portability, and execute with high performance. The programming language itself should be interpreted, threaded and dynamic. [114]

An example implementation of a standard deck of cards in Java is shown in the Program 3.1. Java 7 improves type inference for generic instance creation [115]. It allows to invoke the constructor of a generic class with an empty set of type parameters <> if the compiler can infer the type arguments from the context. New "diamond syntax" is used at line 15 to create an `ArrayList<>()` which compiler will infer as `ArrayList<Card>()`.

```
1  List<String> ranks = Arrays.asList("Ace", "2", "3", "4", "5", "6", "7", "8", "9", "10",
2      "Jack", "Queen", "King");
3  List<String> suits = Arrays.asList("Clubs", "Diamonds", "Hearts", "Spades");
4
5  public class Card {
6      private String rank;
7      private String suit;
8      public Card(String rank, String suit) {
9          this.rank = rank;
10         this.suit = suit;
11     }
12     public String value() { return rank + " of " + suit; }
13 }
14
15 List<Card> deck = new ArrayList<>();
16 for (String rank : ranks) {
17     for (String suit : suits) {
18         deck.add(new Card(rank, suit));
19     }
20 }
21
22 for (Card card : deck) {
23     System.out.println(card.value());
24 }
```

Program 3.1: *Making and printing a deck of cards with Java.*

www.manaraa.com

Java offers a very simple memory model where all variables of object types are references to the objects allocated on the heap. Java also eliminates certain low-level constructs such as pointers. Therefore, providing a simple memory management which is handled through integrated garbage collection performed automatically by the JVM.

The fact that Java is standardized has contributed to its success as a software system. Due to standardization it has specifications that describe how it is supposed to work. In practical terms it allows different project groups and companies to produce interoperable and compatible implementations, although without any guarantees on the performance of the different implementations on the same task. Specifications can be used to provide assurances about the correctness of the implementation.

Java is used in a wide variety of computing platforms from mobile phones and embedded devices, to enterprise servers and supercomputers. Today Java is one of the most popular programming languages, with a reported 10 million users [116; 117]. Originally Java was released in 1995 as a core component of Java platform.

3.3.2 Groovy

Groovy [118] is an agile, dynamic, compiled programming language for the Java Virtual Machine with a Java like syntax with enhanced flexibility. It builds upon the strengths of Java by enhancing it with powerful features inspired by programming languages like Python, Ruby, Perl and Smalltalk. Groovy is widely adapted as a scripting and rapid prototyping language, and it is often the first non-Java programming language that is investigated on the JVM due to almost-zero learning curve for Java developers. In addition, it integrates seamlessly with all existing Java classes and libraries.

Groovy is a superset of Java leveraging the Java's enterprise capabilities but also adding productivity features like closures, builders and dynamic typing. Groovy also supports building DSLs meaning that code becomes easy to read and maintain. It provides a native support for XML as well as many existing modules for various types of tasks. It also offers powerful processing primitives for shell and build scripting. Groovy is designed to increase developer productivity by reducing scaffolding when developing. It also simplifies testing by including supporting for unit testing and mocking.

An example implementation of a standard deck of cards in Groovy is shown in the Program 3.2. Groovy introduces `def` which is a replacement for a type name. It is used to indicate in variable definition that the type is irrelevant. Although in variable definition it is mandatory to either provide a type name explicitly or replace it with `def`. A trivial Groovy object contains only properties, and it can be created using the default constructor or with named parameters. In addition, Groovy supports closures which are similar to Java's inner classes, except they are a single method which is invokable, with arbitrary parameters. Closures allow collections to be processed in a clean way, for example, looped through with the `each` closure shown between lines 17 to 19.

المنارة للاستشارات
www.manaraa.com

```
1  def ranks = ['Ace', 2, 3, 4, 5, 6, 7, 8, 9, 10, 'Jack', 'Queen', 'King']
2  def suits = ['Clubs', 'Diamonds', 'Hearts', 'Spades']
3
4  class Card {
5     def rank
6     def suit
7     String value() { println rank + ' of ' + suit }
8  }
9
10 List deck = new ArrayList()
11 for (rank in ranks) {
12    for (suit in suits) {
13       deck.add(new Card(rank: rank, suit: suit))
14    }
15 }
16
17 deck.each { card ->
18    card.value()
19 }
```

Program 3.2: *Making and printing a deck of cards with Groovy.*

Groovy 2.1 is the latest major and stable version. Current version adds support for static type checking so that the compiler can report about the correctness of the source code. Also a support for static compilation is provided to enhance the performance of the critical parts in an application. Groovy 2.1 supports a special annotation to assist documentation and type safety of domain-specific languages, thus providing beyond conventional static type checking capabilities. It also introduces feature-oriented modularity, and takes full advantage of the new Java 7 invokedynamic [110] support for dynamic programming languages on the JVM.

The fact that alternative programming languages on the JVM can purely interoperate with Java also implies that they can be deployed into a preexisting environment. Groovy is placed on the dynamic layer since it is widely used as a scripting and rapid prototyping language, and it is known for being great for building DSLs [87].

3.3.3 Scala

Scala [119] is a general-purpose, object-oriented programming language that also supports aspects of functional programming paradigm. It is designed to express common programming patterns in a concise, elegant and type-safe way. Scala increases the developers productivity by smoothly integrating the features of object-oriented and functional programming languages. Amount of lines of code is typically reduced by a factor of two to three compared to an equivalent Java application.

Scala is a statically typed, compiled programming language like Java, but unlike Java it performs a vast amount of type inference. Therefore, it can be described as a statically typed dynamic programming language due to its dynamic language resembling syntax. Scala provides an XML integration which makes it an interesting option also for web environment.

المنارة للاستشارات
www.manaraa.com

An example implementation of a standard deck of cards in Scala is shown in the Program 3.3. Scala allows multiple ranges to be used in a single loop, in which case all the possible computations of the given ranges are iterated as shown between lines 8 to 11. In Scala the variables are created either with the val keyword or with the var keyword. Variables instantiated with the val keyword are immutable read-only variables. Scala constructors differs from Java. In Scala the primary constructor is the class body followed by its parameter list. In addition, Scala supports closures which are similar to Groovy. Closures allow collections to be processed in a clean way, for example, looped through with the foreach closure as show between lines 13 to 15.

```
1  val ranks = List("Ace", "2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King")
2  val suits = List("Clubs", "Diamonds", "Hearts", "Spades")
3
4  class Card(val rank: String, val suit: String) {
5     def value() = println(rank + " of " + suit)
6  }
7
8  var deck = for {
9     rank <- ranks
10    suit <- suits
11 } yield new Card(rank, suit)
12
13 deck.foreach { card =>
14    card.value()
15 }
```

Program 3.3: *Making and printing a deck of cards with Scala.*

The language design of Scala has learned a great deal from Java as it overcomes several long-term annoyances that Java developers have had to cope with. Scala is located on the same stable layer with Java offering features like superior concurrency and powerful testing support [87]. Some developers argue that Scala might one day challenge Java as the “next big language” on the JVM.

3.3.4 Clojure

Clojure [120] is a dynamically typed, functional programming language from the Lisp family from where it inherits many of its syntactic features. Clojure implements the code-as-data philosophy and a powerful macro system. It is designed as a general-purpose programming language to combine the approachability and interactive development of a scripting language with an efficient and robust infrastructure for multithreaded programming.

Clojure features a rich set of immutable, persistent data structures, and an explicit progression of time constructs which are intended to facilitate the development of more robust programs, especially multithreaded ones. The need for mutable data is provided both through a software transactional memory system and a reactive Agent system that together ensure clean and correct multithreaded design.

المنارة للاستشارات
www.manaraa.com

An example implementation of a standard deck of cards in Clojure is shown in the Program 3.4. The philosophy behind Clojure states that most parts of most programs should be functional, based on the assumption that programs that are more functional are more robust. Clojure uses immutable data structures to avoid mutating state. In case some changes needs to be done with an immutable collection, a new collection based on the old one is created with the additional changes. Keyword `def` is used to create and bind a data to a *Var*. Vars provide a mechanism to refer to a mutable storage location that can be dynamically rebound to a new storage location on per-thread basis.

```
1 (def ranks ["Ace" 2 3 4 5 6 7 8 9 10 "Jack" "Queen" "King"])
2 (def suits ["Clubs" "Diamonds" "Hearts" "Spades"])
3
4 (def deck (for [rank ranks suit suits] [rank suit]))
5
6 (doseq [card deck] (println (first card) "of" (second card)))
```

Program 3.4: *Making and printing a deck of cards with Clojure.*

Programming languages from the Lisp family are traditionally referred as expert-only. Clojure differs by being somewhat easier, yet still providing developers its formidable power. Clojure has been adopted well within test-driven development. Clojure's advantages and features are seen significant only for enthusiasts and specialized tasks like financial applications. It is likely to remain outside the mainstream of programming languages. Clojure is located on the dynamic layer, but due to its concurrency support and other features it can also be used as a fully fledged stable layer programming language [87].

3.4 Vert.x framework for the modern web and enterprise

Vert.x [121] is an asynchronous, multi-language, event-driven web application framework build on top of the JVM. It provides a new approach to polyglot programming in web application development. Vert.x offers an additional wrapper around its Java core foundation, and also provides its own runtime environment. The core Java API is exposed through the wrappers, thus making it feel like idiomatic framework for every programming language it supports. Currently the API is exposed in Java, Groovy, CoffeeScript, JavaScript via Rhino framework, Ruby via JRuby and Python via Jython, and with programming languages like Scala and Clojure on the road-map.

Vert.x offers simplicity through convention over configuration by enabling to write real, scalable applications in just a few lines of code without excess XML configuration files. The core scalability is utilized efficiently using message passing and immutable shared data. Also a simple concurrency model is offered to get rid of the superfluous hassle with the traditional multithreaded programming. Vert.x enables polyglot programming so that application components can be written solely or as a combination of supported programming languages in a single applications.

المنارة للاستشارات
www.manaraa.com

Vert.x provides a different approach to the contemporary expectations of a web framework. It faces the fact that an increasing amount of the web application logic is implemented in JavaScript and run in the browser. Therefore, rendering the HTML pages and processing form data is no longer the focus of the framework [122]. The server is to output static HTML and JavaScript pages, and process connections to JavaScript clients through WebSockets. Furthermore, Vert.x supports JavaScript communication library SockJS to enable direct communication between the server and the JavaScript run in the browser. Data is handled through a REST interface which utilizes the HTTP protocol functionality, thus making the abstraction of HTTP less important for web frameworks.

3.4.1 Effortless asynchronous application development

Vert.x consists of Netty for much of its network input and output paired with Hazelcast for group management of cluster members. *Netty* [123] is an asynchronous event-driven network application framework which enables rapid development of maintainable high-performance network applications such as protocol servers and clients. *Hazelcast* [124] is an open source, peer-to-peer, in-memory data grid for Java.

Vert.x provides a succinct API which reduces the excess amount of an inordinate of factories, providers, and thread pools that needs to be created with Netty just to do simple things. Vert.x achieves a simple and clean single responsibility encapsulation by leveraging Hazelcast to provide a high-performance network and in-memory event bus. [125]

Vert.x has an efficient concurrency model that works well, for example, with Scala and Clojure, both of which are programming languages that were designed for multicore processors. Development is greatly simplified since developers can write their implementation as single threaded. This is a relief with multithreaded programming in Java, Scala, or even in Ruby, since developers do not have to synchronize the access to state. This eliminates a whole class of race conditions and operating system thread deadlocks. It also removes the need for `synchronized` methods, `volatile` variables and explicit locking.

Vert.x provides a high-performance API implementation for every programming language it supports. This way Vert.x enables developers to write high-performance code on the Java Virtual Machine without actually requiring much knowledge of the JVM or the Java platform ecosystem at all. Since Vert.x is a framework library, any programming language on the JVM can leverage from it. Therefore, the entire universe of JVM libraries, concurrency APIs, and developments tools are available to developers. [125]

Vert.x provides an in-depth website and a set of documentations. Programming language independent installation instructions and core concepts are covered in general. An API reference manual goes into detail about each feature. There exists a version of the API manual for each officially supported programming language. Also a generated HTML documentation is provided for the related APIs. In addition, a manual to create Vert.x modules and a guide to implement new programming language support is given.

المنارة للاستشارات
www.manaraa.com

3.4.2 Verticle and Vert.x instances

Verticle is the unit of deployment in Vert.x, think of a particle for Vert.x. Verticles can be written in any of the supported programming languages. A verticle is defined by having a *main* or class in case of Java as shown in Program 3.5 which is actually a script to run to start the verticle.

```
import org.vertx.java.core.Handler;
import org.vertx.java.core.http.HttpServerRequest;
import org.vertx.java.deploy.Verticle;

public class Server extends Verticle {
   public void start() {
      vertx.createHttpServer().requestHandler(new Handler<HttpServerRequest>() {
         public void handle(HttpServerRequest req) {
            String file = req.path.equals("/") ? "index.html" : req.path;
            req.response.sendFile("webroot/" + file);
         }
      }).listen(8080);
   }
}
```

Program 3.5: *A simple Vert.x Java API implementation of a web server verticle, which serves files from the* webroot *directory.*

The Java API implementation is rather verbose. Consider a more compact example of a simple Vert.x Groovy API implementation of a highly scalable web server, which serves files from the *webroot* directory:

```
vertx.createHttpServer().requestHandler { req ->
   def file = req.uri == '/' ? 'index.html' : req.uri
   req.response.sendFile 'webroot/$file'
}.listen(8080)
```

In addition, a verticle may contain other scripts which are referenced from the main, and also any JAR files, and other resources that are used by the verticle. An application may be a single verticle or it may consist of a set of verticles communicating with each other through the event bus.

Verticles run inside a *Vert.x instance* and a single Vert.x instance runs inside its own JVM instance. Many verticles can be run inside a single Vert.x instance. Vert.x isolates all verticles by giving them their own classloader. This way separate verticles cannot interact by sharing static members, global variables or by other means. A Vert.x instance of the previously presented web server verticle can be run with:

```
vertx run Server.groovy
```

Multiple Vert.x instances can be run on the same host, or on different hosts on the network at the same time. Separate instances can be configured to cluster with each forming a distributed event bus over which verticles can communicate. Separate instances for multiple cores on the server can be run with specifying the amount of instances:

المنارة للاستشارات
www.manaraa.com

```
1 vertx run Server.groovy -instances 32
```

Vert.x ensures that all requests are distributed amongst the instances by implementing a multi-reactor pattern, a reactor pattern with more than one event loop. The reactor design pattern is an event handling pattern for handling service requests delivered concurrently to a service handler by one or more inputs. Internally, a Vert.x instance manages a small set of threads, one for every available core on the server. Basically each one of these threads implements an event loop. When a Vert.x instance is run, meaning that a verticle instance is deployed, the server selects an event loop which the verticle instance will be assigned to. Any subsequent work for the particular verticle instance is guaranteed to be executed by the same thread. Since potentially there are thousands of verticles running at any one time, a single event loop is assigned to many verticles at the same time.

Vert.x allows to specify a particular verticle instance as a *worker verticle*. The difference is that the worker verticle is executed in a thread from an internal thread pool called the background pool. Worker verticles are never executed concurrently by more than one thread. Normally worker verticles communicate with other verticles through the event bus, for example, receiving work to process. They are not allowed to use TCP or HTTP clients or servers, and the amount of worker verticles should be kept to a minimum, since the blocking approach does not scale when dealing with many concurrent connections.

3.4.3 Core services and modules

Vert.x core includes a set of services which can be directly called from code in a verticle. The API for the core is provided in each of the supported programming languages. The Vert.x core is considered fairly static and it is not intended to grow much over time.

Vert.x makes it easy to package applications or reusable resources as modules. These modules can be written in any of the supported programming languages. Modules communicate by sending and receiving JSON messages over the event bus. This eliminates the need to write API adaptor for each programming language.

Vert.x includes a module system and provides a public module repository. While the Vert.x core is fairly static, the public module repository is intended to grow, and to offer a wide range of modules. The repository already contains several modules including a persistor, a mailer and work queues. As of Vert.x 2.0, modules can be placed into any Maven repository or Bintray [126] and registered with the Vert.x module registry. This creates an eco-system of Vert.x modules managed by the community. The community is encouraged to create and contribute their own modules for others to use. [127]

As previously shown, setting up a web server with Vert.x takes just a few lines of code. In addition, Vert.x ships with an out-of-the-box web server module. Consider example of Vert.x Groovy API implementation which starts the web server module and also contains the configuration:

المنارة للاستشارات
www.manaraa.com

```
1  def webServerConf = [
2      port: 8080,
3      host: 'localhost'
4  ]
5  container.deployModule('io.vertx~mod-web-server~2.0.0-final', webServerConf)
```

The call to deployModule instructs Vert.x to deploy an instance of the specified module io.vertx˜mod-web-server˜2.0.0-final. Vert.x will download and install the module automatically from the public module repository if the module is not already installed. In addition, scaling up the web server is trivial. It is simply done by starting more instances of the web server. This can be achieved by specifying the amount of instances to start in the call to deployModule:

```
1  container.deployModule('io.vertx~mod-web-server~2.0.0-final', webServerConf, 32)
```

Vert.x notices that multiple servers on the same host and port are being started. It will internally maintain a single listening server and round-robin schedule all connections between the various instances.

A Vert.x bus module *busmod* is a specific type of module that communicates with other verticles only on the event bus by sending JSON messages as shown in Figure 3.5. Server component is a module which works as a mediator allocating requests from the clients to the respective verticles handling them. A verticle communicates on the event bus in purpose of handling requests to other verticles and modules. Busmods are listening to the event bus and will work accordingly based on the request.

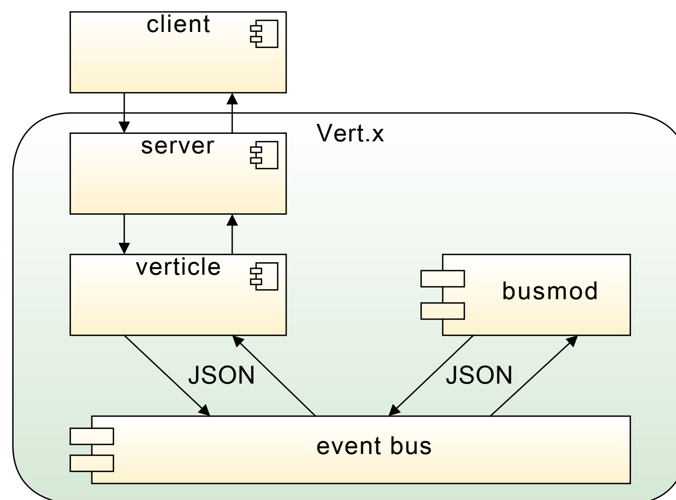


Figure 3.5: *Propagating JSON messages between verticles and busmods on the event bus.*

An example busmod, such as a MongoDB persistor io.vertx˜mod-mongo-persistor˜2.0.0-final can be instantiated out-of-the-box from the command line like any other verticle:

المنارة للاستشارات
www.manaraa.com

```
def persistorConf = [
   address: 'test.persistor',
   db_name: 'testdb'
]
container.deployWorkerVerticle('io.vertx~mod-mongo-persistor~2.0.0-final', peristorConf)
```

Since it uses the event bus to communicate, the busmod is instantly usable by other verticles irrespective of the programming language they are written in. In this particular case the verticle is deployed as a worker verticle by invoking `deployWorkerVerticle` to ensure that it is never executed concurrently by more than one thread to ensure consistency in the database.

3.4.4 Polyglot programming with Vert.x

Vert.x allows developers to write verticles in a choice of programming languages, currently supporting Java, Groovy, CoffeeScript, JavaScript, Ruby and Python, and aiming to support Scala and Clojure. Deployed verticles can seamlessly interoperate with other verticles irrespective of what programming language they are written in. A verticle or multiple verticles that form an entire application or a reusable resource can be easily packaged as a Vert.x module.

Vert.x builds on event-driven programming (EDP) model which is similar to approach in frameworks such as Node.js [128]. In EDP the flow of the program is determined by events, for example, messages from other programs or threads. Basic concept of Vert.x involve setting event handlers. For example, to receive data from a TCP socket a handler must be set, this handler is then called when the data arrives. Consider an example of a simple TCP server that echoes everything it receives on the socket:

```
def server = vertx.createNetServer()
server.connectHandler { sock ->
   sock.dataHandler { buffer -> sock << buffer }
}.listen(1234, 'localhost')
```

Similarly, handlers are set to receive messages from the event bus, to receive HTTP requests and responses, to be notified when a connection is closed, or to be notified when a timer fires. All other operations that do not involve handlers are guaranteed never to block, for example, writing data to a socket.Writing data to a socket can be done by invoking the `write` method or using the left shift operator:

```
sock.write('hello')
sock << 'world'
```

In case Vert.x API allowed a blocking read on a TCP socket, and a verticle called that blocking operation and no data would arrive in a period of time, the thread running that blocking operation would be fully occupied. This means that the thread could not work for any other verticle during that period of time. In such system, all verticles would

www.manaraa.com

need to be assigned their own thread. Consider what would happen with thousands, tens of thousands, or hundreds of thousands of verticles running. It is clear that this kind of blocking model would not scale because the overhead due to context switching and stack space would be immense.

Verticles can communicate with other verticles using an event bus. Vert.x event bus resembles the actor model popularized by the Erlang programming language, where verticles are the actual actors. In response to a message that a verticle receives, it can make a local decision, create more verticles, send more messages, and determine how to respond to the next message received. The system is able to scale well over available cores without need for multithreaded execution of any verticle code. This is achieved by having multiple verticle instances in a Vert.x server instance and allowing message passing through the event bus. The following snippet is an example where a handler registered to address `test.address` echoes the body of the message it receives:

```
1 def eb = vertx.eventBus
2 eb.registerHandler('test.address') { message -> println message.body }
```

Event bus messaging supports two main styles of asynchronous messaging. Message queue, also known as point-to-point messaging, is used when sending a message that will result in at most only one handler registered at the address `test.address` receiving the message. Publish subscribe messaging is also supported. Publishing assures that the message will be delivered to any handlers registered against the given address `test.address`:

```
1 eb.send('test.address', 'Sent message')
2 eb.publish('test.address', 'Published message')
```

Although message passing is extremely useful, it is not always the best approach to concurrency. An example would be an application providing an in-memory web cache. When a resource request arrives, the server looks up the resource in the cache and returns it if the data is present, otherwise loading it from disk and placing it in the cache for the next time. Application should also scale across all available cores. Modeling this using message passing proves problematic. Implementing a single verticle that manages the cache would mean that all requests to the cache would be serialized through that single threaded verticle instance. This could be improved by having multiple instances of the verticle managing separate parts of the cache, although implementing this would quickly get ugly and complicated.

As a solution, Vert.x provides a shared map structure that can be accessed directly by separate verticle instances in the same Vert.x instance. The shared map and shared set facility allow only immutable data to be shared between verticles:

```
1 def map = vertx.sharedData.getMap('test.sharedmap')
2 map['some-key'] = 123
3 def set = vertx.sharedData.getSet('test.sharedset')
4 set << 'some-value'
```

المنارة للاستشارات
www.manaraa.com

With a single line of code, the requested data from the server can be looked up in the cache and returned to the user. It is good to remember that shared data is only dangerous if the data shared is mutable.

The polyglot programming approach of Vert.x has its concerns. The foremost concern is that taking programming languages like Ruby, Python or JavaScript from their normal environments can be confusing for developers, as the developers need to internalize alternative configurations for systems and installing packages. In addition, sufficient knowledge of Java is required to take an advantage of the JVM libraries when native wrappers for alternative programming languages do not exist. This reason alone indicates the future success of native JVM programming languages like Clojure, Groovy and Scala with the Vert.x framework. [125]

3.4.5 Support for new programming languages

The Vert.x core distribution is implemented in Java. Support for alternative programming languages is provided in the form of modules. Vert.x 2.0 made it possible for the community to implement support for new programming languages, or alternative programming language engines for already supported programming languages. New programming language implementations are pushed to any Maven repository or to Bintray and registered with the Vert.x module registry.

A general rule is to encapsulate the Vert.x Java API inside idioms common in the programming language in question. For example, the Java API uses an interface to represent an event handler, whereas Groovy API uses a closure, and Ruby API uses a block. For this reason, the programming language module needs to implement an API wrapper which converts the raw Java API to a form that follows the conventions and best practices of the programming language in question.

Programming language implementations are marked as `'resident': true` in the module configuration file `mod.json` to annotate that the module will not get unloaded until the JVM exits. In addition, the modules are marked as `'system': true` which means that they will be installed into the `sys-mod` directory of the Vert.x installation so that they will not be downloaded for every application that requires them.

المنارة للاستشارات
www.manaraa.com

4. IMPLEMENTATION

This chapter describes four polyglot programming project implementations of a simple web application that were produced as a part of this thesis to experiment with polyglot programming in a real-world situation. The project was implemented in both Java and Groovy as a server-side application, and also with the Vert.x and AngularJS frameworks as a JavaScript client-side application to allow direct comparison of several polyglot systems. Also an additional Groovy project with Java domain model was made to explore interoperability and usage of legacy code. In addition, observations on Groovy as the programming language of choice, and in web development comparing traditional methods with client-side architecture are presented.

Example application consist of a small domain model with only simple business logic to add, remove and list departments and employees. These project implementations serve as proof of concept to demonstrate the feasibility behind polyglot programming and polyglot programming pyramid. Simplicity of the application limits its ability to demonstrate some cases where different programming languages and paradigms might prove useful. However, proof of concept design is given to verify that the concept has the potential of being used. All of the project implementations are fundamentally identical.

4.1 Project structure

Java and Groovy are both widely used programming languages on the Java platform. In addition, interoperability of programming languages provided by Java Virtual Machine can be leveraged to reduce risks and benefit from legacy code in some parts of the project. The Vert.x project provides a different approach with its JavaScript client-side application. A simple web application is often split into four layers: domain model, data access, business logic and web layer for binding other layers to the web environment.

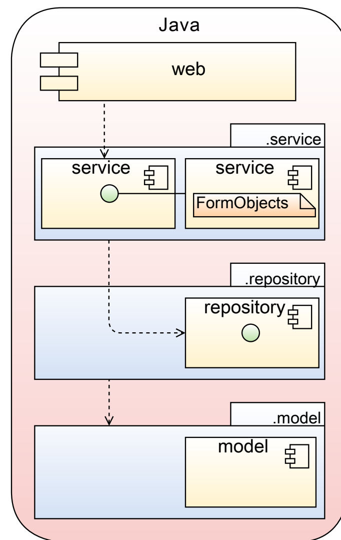
Java project uses *Maven* [129] for managing the project and its dependencies. Groovy projects benefit from the convention over configuration provided by the Grails framework by using the built-in Grails dependency resolution DSL. Vert.x project is also managed with Maven, and its own module repository is used to fetch required modules.

All of the example projects utilize *jQuery* [130] JavaScript library to simplify the client-side scripting of HTML and to produce more dynamic web content. Another common element is the sleek and intuitive *Twitter Bootsrap* [131] front-end framework that is used to ease and speed up the content presentation and web development in general.

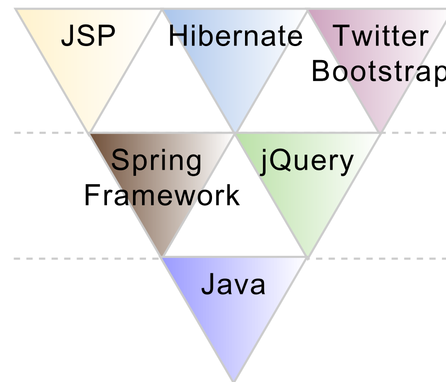
المنارة للاستشارات
www.manaraa.com

4.1.1 Web project with Java using Spring Framework and Hibernate

Structure of the project shown in Figure 4.1a follows the four layer architecture of a simple web application. Domain model represent the real-world problem, repository provides the data access abstraction and service implements the business logic, whereas web module binds the whole application to the web environment.



(a) Project structure



(b) Polyglot programming pyramid

Figure 4.1: *Project structure and polyglot programming pyramid of the Java project.*

Java provides the basis for the polyglot system in this project. It is located in the stable layer of the polyglot programming pyramid shown in Figure 4.1b. *Spring Framework* [132] contributes to the dynamic layer by providing an efficient and versatile web application development framework for Java. Spring is an essential part of the web application implementation and its technologies contribute to all layers.

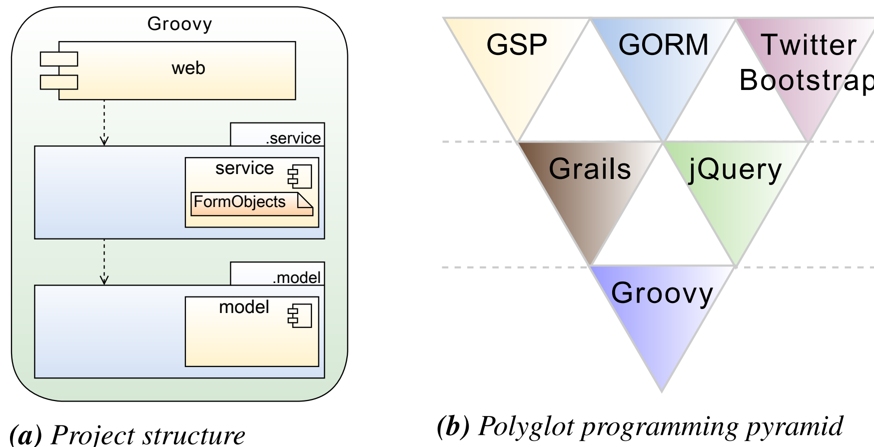
Domain layer contains different technologies targeting certain and concrete problem domains. Technologies located in the domain layer are usually utilized inside the other two layers to provide problem domain specific solutions. *Hibernate* [133] annotations are used to create the object-relational mapping and validation of the domain model objects which is used in conjunction with Spring JPA Repository to provide the data access.

The web module contains web application specific components implemented on top of the Spring Framework and also the HTML view layer implemented with JSP. Twitter Bootstrap is used to enhance the content presentation and jQuery is utilized to provide dynamic features on the client-side.

المنارة للاستشارات
www.manaraa.com

4.1.2 Web project with Groovy using Grails framework

The project structure of the Groovy project shown in the Figure 4.2a follows the principles of the four layer architecture of a simple web application, although the repository layer is omitted because data access is encapsulated in the domain model objects. Domain model represent the real-world problem, service provides implements the business logic and uses the domain model data access abstraction, whereas web module binds the whole application to the web environment.



(a) Project structure

(b) Polyglot programming pyramid

Figure 4.2: *Project structure and polyglot programming pyramid of the Groovy project.*

Figure 4.2b represents the polyglot programming pyramid for the project. Groovy is the programming language of choice in this project and is located in the stable layer. *Grails* [134] is a project to extend Groovy to web application development. It provides a fully fledged agile web development environment built upon existing technologies like Spring Framework and Hibernate leveraging the features of Groovy. Having principles like convention over configuration, Grails strives for quick and simple development with less configuration providing many common features out of the box. Grails is an essential part of a web application contributing to all layers, although foremost to the dynamic layer by providing a very versatile web application development framework.

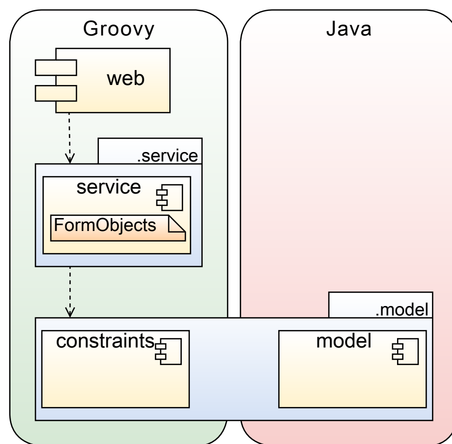
The domain layer contains different technologies targeting certain and concrete problem domains. Technologies located in the domain layer are usually utilized inside the other two layers to provide problem domain specific solutions. Grails provides its own object-relational mapping build on top of Hibernate to implement data access on domain classes.

The web module contains web application specific components implemented on top of the Grails framework and also the HTML view layer implemented with GSP. Similarly to the Java project, Twitter Bootstrap is used to enhance the content presentation and jQuery is utilized to provide dynamic features on the client-side.

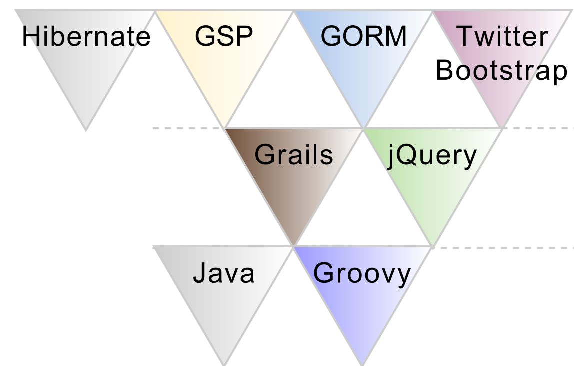
المنارة للاستشارات
www.manaraa.com

4.1.3 Web project with Groovy using Grails framework and Java legacy domain model

The project structure of the Groovy project with a Java legacy domain model is shown in the Figure 4.3a. Figure 4.3b represents the polyglot programming pyramid for the project. This project was made to study programming language interoperability and as a proof of concept on how to utilize Java implementation of the domain model in a Grails web application. The project is otherwise identical with the previously described Groovy project.



(a) Project structure



(b) Polyglot programming pyramid

Figure 4.3: *Project structure and polyglot programming pyramid of the Groovy project with Java legacy domain.*

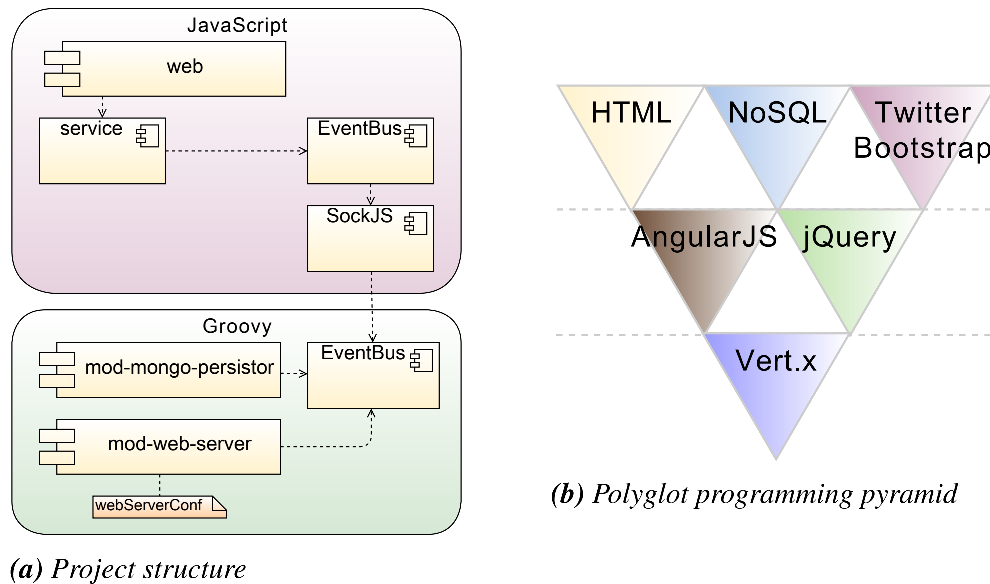
The difference is that this project uses the Hibernate and JPA annotated Java legacy domain model implementation from the previous Java project in conjunction with the Groovy project. How this is achieved is covered subsequently.

4.1.4 Single-page application with Vert.x framework and AngularJS

A *single-page application* (SPA) is a web application within a single web page with a goal of providing more fluid user experience similar to a desktop application. Only a single page load is required to retrieve all necessary HTML, JavaScript and CSS. Additional resources are dynamically loaded and added to the page on request, generally in response to user interactions. A single-page application does not load at any point in the process, neither the control transfers to another page. Modern web technologies – in this case Twitter Bootstrap and jQuery – are used to provide the perception and navigability of separate logical pages in the application.

المنارة للاستشارات
www.manaraa.com

The project structure of the client-side project shown in the Figure 4.4a follow the principles of a *thin server architecture* to implement a single-page application. Figure 4.4b represents the polyglot programming pyramid for the project.



(a) *Project structure*

(b) *Polyglot programming pyramid*

Figure 4.4: *Project structure and polyglot programming pyramid of the Vert.x project.*

Groovy is used as the programming language of choice on the server-side and JavaScript on the client-side. Both of them utilize the native Vert.x API implementations. An out-of-the-box MongoDB persistor bus module is used as a NoSQL document database that provides the necessary data access over the event bus.

The implemented SPA uses dynamic communication to interact with the web server behind the scenes. The Vert.x event bus is utilized in conjunction with *SockJS* [135] to bridge the client-side with the server-side application. The server-side application deploys the web server, enables the SockJS bridge, and also deploys the MongoDB persistor bus module which instantly registers on the event bus.

Requests to the server over the event bus result in raw data in JSON representation being returned. A client-side *AngularJS* [136] JavaScript implementation uses the returned JSON to update the partial area of the *Document Object Model* (DOM). AngularJS is web framework used at the client-side to implement the web application and to provide declarative templates with data-binding.

The web module contains web application specific components implemented on top the AngularJS framework and also the pure HTML view layer enhanced with AngularJS directives. Twitter Bootstrap is used to enhance the content presentation and jQuery is utilized to provide dynamic features on the client-side.

www.manaraa.com

4.2 Web flow execution, decorators and mapping

An execution flow of a web application in Java and Groovy projects are identical, since the Grails framework builds on top of the Spring Framework. The execution flow of an application shown in Figure 4.5 corresponds roughly to a basic web application. An incoming HTTP request traverse through some servlet filters and the decorator filter after which it is passed to Spring Framework. Spring maps the incoming request to a controller binding and validates request parameters when appropriate. The controller uses service methods to interact with a database, after which the control is returned back to the controller which prepares and returns an appropriate view to Spring. Spring then renders the view producing the resulting HTML page that is returned through the decorator to the client.

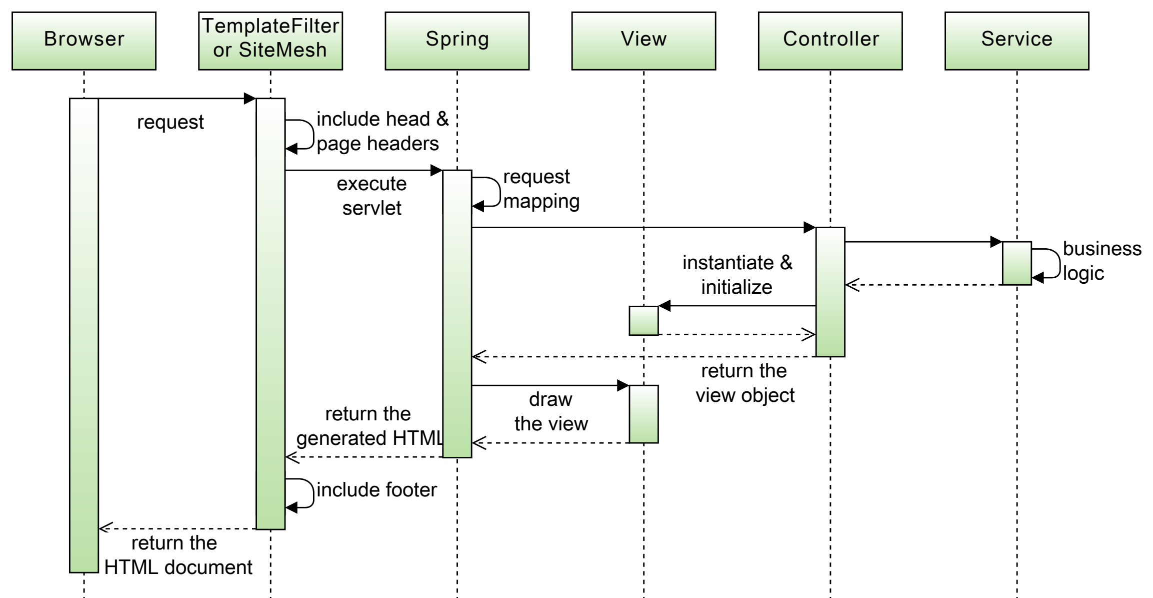


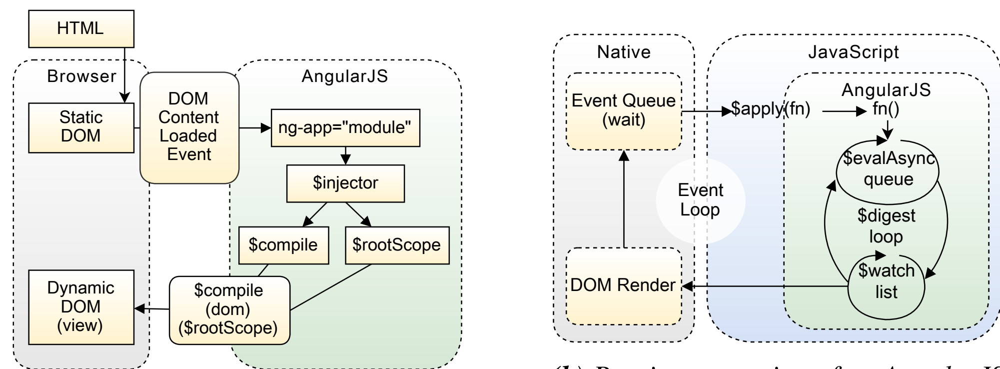
Figure 4.5: *Execution flow of the web application.*

Decorators are used to wrap additional data around a regular view. Decorators allow references to common elements to be omitted from the code of the actual pages. A common use case is to include a header, a footer and a menu to every page of an application. Both Java and Groovy projects use *SiteMesh* [137] decoration framework. The Vert.x project implements a single-page application, and thus there is no need for a separate decorator to include common elements to several pages – there is only a single page.

The execution flow of the single-page application in the Vert.x project is rather different because of the thin server architecture and the client-side single-page application approach that uses JavaScript AngularJS web framework. The server-side only serves an HTML file and deploys a MongoDB persistor on the Vert.x event bus to provide necessary data access for the client-side application.

المنارة للاستشارات
www.manaraa.com

Startup of an AngularJS web application is shown in the Figure 4.6a. The browser loads the HTML file from the server and parses it into the DOM, after which it loads the `angular.js` script file. AngularJS waits for `DOMContentLoaded` event before it starts. AngularJS searches for the application boundary designated by `ng-app` directive. The `ng-app` directive can be used to specify a module containing declarative specifications on how the application is bootstrapped. The module specified in the `ng-app` directive is used to configure the `$injector`, which creates the `$compile` service and the `$rootScope`. The `$compile` service compiles the DOM and links it with the `$rootScope`. All AngularJS directives are matched against the HTML and executed during the DOM compilation. This way directives can register behavior or transform the DOM.



***(a)** Startup of an AngularJS web application.*

***(b)** Runtime execution of an AngularJS web application.*

***Figure 4.6:** Startup and runtime execution of the AngularJS web application.*

Runtime execution of an AngularJS web application is shown in the Figure 4.6b. The event loop in the browser waits for an event to arrive. Example events include user interactions, timer or network events such as a response from the server. Execution of the event's callback function enters the JavaScript context, where the DOM structure can be modified. The browser leaves the JavaScript context after the callback is executed, and the view is rendered based on DOM changes.

AngularJS interacts with the event loop in the browser by modifying the normal JavaScript flow. AngularJS splits the JavaScript execution context into classical and AngularJS context by providing its own event processing loop. Operations applied in AngularJS execution context will benefit, for example, from data-binding, exception handling, and property watching. The `$apply()` can be used to enter the AngularJS execution context from JavaScript. Controllers, services and other methods call the `$apply()` automatically by the directive handling the event. An explicit call to `$apply()` is only needed when implementing custom event callbacks, or when working with third-party library callbacks.

A function is run in the AngularJS execution context by calling `scope.$apply(stimulusFn)` on the work to be processed. AngularJS executes the `stimulusFn()`, which typically alters

المنارة للاستشارات
www.manaraa.com

the application state. AngularJS then enters the `$digest` loop and iterates until it stabilizes, which means that the `$evalAsync` queue is empty and the `$watch` list does not detect any changes. The `$evalAsync` queue schedules all of the work that needs to occur outside of the current stack frame, but before the browser renders the view. The `$watch` list contains a set of expressions that may have changed since last iteration, and if any detected the `$watch` function is called typically updating the DOM with a new value. Execution leaves the AngularJS and JavaScript context once the `$digest` loop has finished. Browser then re-renders the DOM to reflect any changes.

Spring framework supports multiple ways for request mapping including a Java annotation based used in the Java project and an XML based mapping. Grails framework extends the underlaying Spring by introducing some Django like features. Django is a web framework written in Python. Grails provides request mapping based on regular expressions and by capturing groups as arguments for the target method. The mappings are reverse matched so that URLs can be inserted to a page by simply referring a mapping by specifying controller name and action. This way URL handling and refactoring is considerably improved. Program 4.1 shows a Grails configuration file `UrlMappings.groovy` which contains the URL request mappings specified in the Groovy projects.

```
1  class UrlMappings {
2    static mappings = {
3      "/departments"(controller: "department", action: "departments",
4          view: "department/departments")
5      "/departments/add"(controller: "department", action: "add", view: "department/departments")
6      "/departments/delete/$departmentId?"(controller: "department", action: "delete",
7          view: "department/departments")
8
9      "/employees"(controller: "employee", action: "employees", view: "employee/employees")
10     "/employees/add"(controller: "employee", action: "add", view: "employee/employees")
11     "/employees/delete/$employeeId?"(controller: "employee", action: "delete",
12         view: "employee/employees")
13     "/employees/change/$employeeId?/$departmentId?"(controller: "employee",
14         action: "changeDepartment", view: "employee/employees")
15
16     "/api/employees/"(controller: "employee", action: "listEmployeesJson")
17     "/api/departments/"(controller: "department", action: "listDepartmentsJson")
18     "/api/municipalities/"(controller: "municipality", action: "listMunicipalitiesJson")
19     "/api/municipalities/$term?"(controller: "municipality", action: "checkMunicipalitiesJson")
20
21     "/"(controller: "overview", action: "overview", view:"overview")
22     "500"(view:'/error')
23   }
24 }
```

Program 4.1: *Grails URL request mapping.*

Single-page application approach of the Vert.x project omits the need for URL request mapping. Twitter Bootstrap and jQuery are used to provide similar perception and navigability of separate logical pages. AngularJS controllers use services as directives which can be utilized directly from the HTML. The Vert.x event bus is configured on the server-side to allow interaction with the MongoDB persistor from the AngularJS services.

المنارة للاستشارات
www.manaraa.com

4.3 Form objects, binding and validation

Most web applications modify stored data. HTML forms are used to achieve this, however forms themselves provide no binding or validation logic. Web frameworks provide ready-made components to simplify the process of reading the data from the request parameters, and to perform form validation and binding of the data into specified program components, for example, to JavaBeans.

Spring allows any object to be used as a form object responsible for handling the data in the forms. A form object is a representation of an existing domain model object containing only the needed properties. In general, a domain model object should not be used directly as a form object, since the form data rarely corresponds one-to-one with the domain model properties. Form objects usually follow the principles of *Data Transfer Objects* (DTO) which are *Plain Old Java Objects* (POJO) that are used to contain only the needed properties for the form in question. DTOs require manual initialization from the corresponding model properties. After binding the request parameters the properties are manually validated and finally the values are manually copied to the corresponding domain model objects.

A disadvantage of this approach is the missing relation between the properties in the form object and in the corresponding domain model object. Often the properties are congruent even though the whole classes are not. The missing relation of the properties render any information provided in the model object for its properties unavailable to validation, thus no automatic binding in either direction is possible. But then again, this approach entail simplicity and clear separation of concerns as an advantage.

The relation between the properties in the domain model and in the form object can be achieved by encapsulating the required information in a separate object which is then used to replace the related properties in the form object. Thus the type information and annotations in the model object becomes available to the framework to perform automatic validation. This also enables the generation of rules for the client-side validation. A more common approach is simply to duplicate the necessary properties and annotations specified in the domain model object into the form object.

Groovy and its high-productivity web framework Grails makes the relation between domain models and form objects more easier to maintain by enhancing the underlaying Spring Framework Validator API and data binding. Grails provides an unified way to declaratively specify validation rules known as constraints. A common way is to create constraints for properties in domain classes and then import the required ones into corresponding form objects as shown at line 11 of Program 4.2. All constraints that map with the specified properties in both classes will be imported, if not stated otherwise. Also additional constraints can be specified for required fields as shown at line 12.

المنارة للاستشارات

www.manaraa.com

```
package info.harmia.polyglot.grails.service
/* imports */

@Validateable
class EmployeeForm {
  String name
  String email
  Long departmentId

  static constraints = {
     importFrom Employee
     departmentId nullable: false
  }
```

Program 4.2: *Validation constraints can be specified or imported from the model object.*

AngularJS client-side applications provide a different role for the forms. JavaScript logic is triggered to handle the form submission in application specific way, in contrast to classical round-trip applications that translate the form submission into a full page reload that sends the data to the server.

AngularJS prevents the default action of form submission to the server unless an `action` attribute is specified in the `form` directive element. AngularJS applications use form submission to trigger a JavaScript method specified via `ng-submit` directive on the form element or via `ng-click` directive on the first button or via `input` directive by specifying field of type submit `input[type=submit]`.

AngularJS uses the form and form controls to provide client-side validation services. Client-side validation provides better user experience, because of the instant feedback and instructions on how to correct pending errors. Server-side validation is still necessary for a secure application, since client-side validation can easily be circumvented and thus can not be solely trusted.

AngularJS provides two-way data-binding with an `ng-model` directive which is responsible for synchronizing the model to the view, as well as the view to the model. The `ng-model` directive provides an API for other directives to augment its behavior.

4.4 Model, Repositories and Services

A model is used to describe a real-world problem domain that shapes and forms the basis of an application. Services are used to implement the business rules, thus considered to be the most important components. Repositories wrap a facade, an additional layer of abstraction over the *data access objects* (DAO) that are used to abstract out the actual implementation of data acquisition and persistence.

The difference between repository and DAO is that the repository deals with domain objects and DAOs return data as in object state. In fact, repositories use DAOs to retrieve the data from the storage and use them to restore the domain object, or to extract the data from the domain object to be persisted.

المنارة للاستشارات
www.manaraa.com

Web applications often implement a programmatically simple domain model, services, and data access that together form the basis of the web application, although not web-related in any way. Due to programmatic simplicity the gain from advanced programming languages in domain modeling is vapid.

4.4.1 Model

Program 4.3 shows a Java implementation of a simple domain model object representing an employee. The example model uses *Java Persistence API* (JPA) [138] annotations to utilize a POJO persistence model for object-relational mapping and Hibernate Validator annotations for strengthened typing.

```
1  package info.harmia.polyglot.springapp.mvc.core.model;
2  /* imports */
3
4  @Entity
5  public class Employee {
6    @Id
7    @GeneratedValue(strategy = GenerationType.AUTO)
8    private Long id;
9
10   @Basic
11   @NotBlank
12   private String name;
13
14   @Basic
15   @Email
16   @NotBlank
17   private String email;
18
19   @ManyToOne(fetch = FetchType.EAGER)
20   @NotNull
21   private Department department;
22
23   /* basic bean getters and setters for all properties */
```

Program 4.3: *Java implementation of an employee model.*

Line 21 specifies a `department` as a reference to another domain model object. The type information including validator annotations can be used to deduce that `department`, `name` and `email` properties are required fields that may never be `null` or more so `blank`. In addition, the value of the `email` field has to match the `@Email` annotation pattern validation.

Java provides a rather clean domain model class implementation except for the required property getters and setters. Although they are easily generated. Optional fields in model objects may prove to be problematic upon use, since Java has no built-in mechanism to describe fields with possible `null` value. Thus users are expected to know and check for possible `null` value before dereferencing objects:

```
1  String departmentName = null;
2  if(employee.getDepartment() != null) {
3    departmentName = employee.getDepartment().getName();
4  }
```

المنارة للاستشارات
www.manaraa.com

The Java syntax is quite cumbersome. Java 7 was proposed to implement a safe-dereference operator with the following syntax:

```
departmentName = employee.getDepartment()?.getName();
```

The proposal was not included in Java 7 and has been discontinued since it does not remove the problem, although it would simplify the syntax. Groovy provides this kind of safe-dereference syntax to reduce some of the Java verbosity:

```
departmentName = employee?.department?.name
```

Upcoming Java 8 introduces a container object `Optional<T>` which may or may not contain a non-null value. An optional field will return `true` if a value is present, `isPresent()`. The value can be returned with a `get()` method. Following example fetches an optional `department` of an employee:

```
Optional<Department> found = employee.getDepartment();
if(found.isPresent()) {
  Department department = found.get();
  String departmentName = department.getName();
}
```

The adoption of this functional idiom is to render implementations less vulnerable to null dereferencing problems and as a result more robust and less error-prone. Beside the increase in readability, it forces developers to actively think about the absent case, since the optional property has to be unwrapped before usage.

Program 4.4 shows a similar domain model object implementation in Groovy. Grails framework omits the Java annotations and provides the enhanced constraints mechanism. Constraints are defined in a `constraints` property that is assigned a code block.

```
package info.harmia.polyglot.grails.model
/* imports */

@Validateable
class Employee {
  String name
  String email

  static belongsTo = [
    department: Department,
  ]

  static mapping = {
    department lazy: false
  }

  static constraints = {
    name blank: false, minSize: 1, maxSize: 255
    email blank: false, minSize: 1, maxSize: 255, email: true
  }
```

Program 4.4: *Groovy implementation of an employee model.*

المنارة للاستشارات
www.manaraa.com

Line 10 specifies a reference to another domain model object `department`. Constraints declare that the `name` and `email` properties cannot be `blank` and must be between 1 and 255 characters long, and that the `email` property is validated against an email pattern. By default, all domain class properties have an implicit `nullable: false` constraint, which means that properties cannot be `null` unless specified otherwise.

Groovy provides a clean and robust domain model object that reduces much of the Java boilerplate. A property name declared without an access modifier generates a private field with a public getter and setter methods. Java persistence and validation annotations are replaced with Grails enhanced mapping and constraints mechanisms. In addition, Groovy's enhanced syntax allows constructors to be invoked with lists and maps to create objects, in addition to a classic Java way:

```
1  def employee = new Employee(params)
2  def employee = new Employee(name: params.name, email: params.email, department: params.department)
```

Groovy and Grails makes it also possible to reuse JPA and Hibernate Java legacy domain models. The existing domain model implemented in Java can be packaged in a JAR and included in the application or by simply copying the source files into the `src/java` directory and used from there. A `mapping` entry is needed for every JPA annotated domain model class in the `/grails-app/conf/hibernate/hibernate.cfg.xml` file as shown in Program 4.5.

```
1  <?xml version='1.0' encoding='UTF-8'?>
2  <!DOCTYPE hibernate-configuration PUBLIC
3          '-//Hibernate/Hibernate Configuration DTD 3.0//EN'
4          'http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd'>
5
6  <hibernate-configuration>
7          <session-factory>
8      <mapping package="info.harmia.polyglot.springapp.mvc.core.model"/>
9      <mapping class="info.harmia.polyglot.springapp.mvc.core.model.Department"/>
10     <mapping class="info.harmia.polyglot.springapp.mvc.core.model.Employee"/>
11     <mapping class="info.harmia.polyglot.springapp.mvc.core.model.Municipality"/>
12         </session-factory>
13 </hibernate-configuration>
```

Program 4.5: *Hibernate configuration file mapping the Java domain model objects.*

Grails introduces GORM which is Grails' object-relational mapping (ORM) implementation built on top of Hibernate 3 open source ORM solution. GORM uses the dynamic nature of Groovy with its static and dynamic typing, along with Grails convention over configuration principle to reduce excess the amount of configuration involved in domain class creation.

GORM validation can be used with a JPA annotated Java domain model by specifying constraints. The constraint metadata can be attached to a class by adding a `package.DomainClassConstraints.groovy` script file to `src/java`. Constraints block needs to be defined in the same package as the Java class file:

المنارة للاستشارات
www.manaraa.com

```
package info.harmia.polyglot.springapp.mvc.core.model

constraints = {
  name blank: false, minSize: 1, maxSize: 255
  email blank: false, minSize: 1, maxSize: 255, email: true
}
```

As described, it is rather easy to incorporate a Java domain model in a Grails web application, although some of the elegance and custom mappings of standard GORM domain classes are lost. However, even with Java domain model classes, GORM is still able to provide dynamic finders, criteria queries, validation and scaffolding. Dynamic scaffolding enables auto-generation of a complete application for a given domain class including necessary views and controller actions for create, read, update and delete (CRUD) operations.

The single-page application project with Vert.x framework has a different approach to domain model objects due to AngularJS and MongoDB. In contrast to Spring, Grails and many other frameworks, AngularJS sets no restrictions or requirements on the domain model object. There is no required inheritance from base classes or any special accessor methods to implement for accessing or modifying the model state. AngularJS accepts any plain JavaScript object as a model, and thus the model can be a primitive, an object hash or a full object type.

AngularJS application uses a model to represent the data which is rendered into a view. Merging the model with a template to produce the view requires only that the model can be referenced from the scope.

4.4.2 Repositories

Lean programming practices are used to eliminate some waste effort and boilerplate code in proof of concept development by favoring “pull” design over “push” design. Therefore, infrastructure concerns like persistence is built only to satisfy the needs of business requirements instead of building a data access layer thought to be needed later on by the application. Repository pattern is used to mediate between the domain model and data mapping layers using a collection-like interface for accessing domain objects. It practically lays a facade over the persistence system and shields the rest of the application code from having to know how the persistence works.

Spring Data provides a repository abstraction to significantly reduce the effort to implement data access layers for various persistence stores. Spring provides `JpaRepository` interface that can be extended to create a domain model repository providing automatically generated CRUD operations for the domain model object:

```
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
}
```

المنارة للاستشارات
www.manaraa.com

The `JpaRepository` extends a `CrudRepository` which in turn extends a basic `Repository`. Utilizing repositories does not restrict to simple CRUD operations. Custom implementations and queries can still be created using the standard JPA queries and with the Hibernate Criteria API.

Grails' object-relational mapping also generates automatically the CRUD operations for domain model objects. GORM also supports number of powerful ways to query. Dynamic finders, where queries, criteria queries and Hibernate's object-oriented query language (HQL) can be used.

Vert.x project uses a *MongoDB* [139] data storage which provides a completely different approach other than the object-relational mapping. MongoDB is a document-oriented database designed to facilitate development and scalability. MongoDB provides a flexible schema for data. Data is stored as collections which do not enforce document structure. Therefore, documents in the same collection are not enforced to have the same structure or set of fields, and that common fields might hold different types of data. Although in practice, most documents in a collection share a similar structure. It is considered as a good practice that each entity or object only contains relevant fields required to represent the document. Schema flexibility is a powerful tool which allows documents in MongoDB to closely resemble and reflect the application-level domain objects.

Vert.x project uses a MongoDB persistor bus module to provide the necessary CRUD operations for the MongoDB instance over the event bus. MongoDB is a practical choice for persisting Vert.x and JavaScript client-side application data since it has native support for JSON documents. Although it is good practice to validate any data on the server-side before persisting, this was omitted from this proof of concept project, since the data access with the MongoDB persistor was also possible from the client-side via the event bus propagating JSON messages.

4.4.3 Services

Because of a simple domain model in the example project, the service implementations in Java and Groovy projects do not differ much. Although Groovy provides a few lines shorter implementation in general, but essentially the content is the same. In addition, the service implementations in the Vert.x project are also fundamentally identical.

Business logic, which is possibly the most important code of an application, is implemented in the service methods. It is essential that the actual logic is isolated from any excess or additional code, thus Groovy can help with its syntax. One of the big advantages Groovy has over Java is its natural and simple way of handling and producing JavaScript Object Notation messages. Generation of JSON messages in Java is extensively verbose and laborious:

المنارة للاستشارات
www.manaraa.com

```
public JSONArray listDepartmentsJson() throws JSONException {
  JSONArray departmentArray = new JSONArray();
  for (Department department : departmentRepository.findAll()) {
    JSONObject departmentJSON = new JSONObject();
    departmentJSON.put("id", department.getId());
    departmentJSON.put("name", department.getName());
    departmentArray.put(departmentJSON);
  }
  return departmentArray;
}
```

Groovy can achieve this same JSON generation in just one line with Grails automatic support for marshalling to JSON:

```
def listDepartmentsJson() {
  Department.list() as JSON
}
```

Grails also supports automatic marshalling to XML. In addition, Groovy provides an extensive and straightforward XML and JSON builders, and familiar dot notation.

The business logic in the Vert.x project is implemented on the client-side using AngularJS services and the Mongo persistor on the event bus when data access is required. Groovy's ability to handle JSON contributed also to the selection of the server-side programming language in the Vert.x project.

4.5 View

View represents the front-end of a web application, an interface for the user to interact with. An average user with vague or no knowledge on how the application works behind the view might argue that the view represents the whole application. Therefore, a bug in the view seems like a bug in the whole application even though it is only related to representation semantics. Thus the view undoubtedly forms the feeling of quality over the whole application. A view representing a slightly distorted data table might induce a doubt over the correctness of the whole application.

View of a web application is most often an HTML document or its variant. These documents are usually built using error-prone string concatenation which is likely to cause problems, for example, with character escaping. String concatenation poses also security issues including JavaScript injection in any structured data.

JavaServer Pages (JSP) technology is the standard way to build HTML documents in Java EE environment. JSP technology separates the user interface from content generation. JSP Standard Tag Library (JSTL) provides a collection of tag libraries that implement general-purpose functionality common to many web applications.

The view which is rendered from the JSP file is actually created by concatenating strings. Therefore, it is inherently bad to implement views with technologies that use string concatenation if the correctness of the view has any requirements. All views that are

المنارة للاستشارات
www.manaraa.com

implemented using a structured document should be created by serializing structural data with ready-made tools to ensure well-formed and correct structure. View technologies like JSP and GSP can only be used safely in conjunction with such a preemptive tool that evaluates all programmer output against any possible error conditions and warns the programmer of any potential defects.

AngularJS template system provides a different approach since it works on DOM objects and not on strings. The template is written in a standard HTML string which the browser parses into the DOM. The DOM is used as an input for the AngularJS template engine known as the compiler. The compiler looks for directives to sets up watches on the model. This approach results in a continuously updating view based on the model, with no need for template model re-merging.

Program 4.6 shows a JSP file composed mostly of HTML with some JSP tags and EL expressions. The tag libraries used are introduced after the page header in the beginning of the file. Localization messages are fetched from resource files with fmt:message tag which can handle message parameters. Core library provides conditional formatting with c:if tag and looping with c:forEach tag. Text escaping is done with c:out tag.

```
1  <%@ page contentType="text/html;charset=UTF-8" language="java" %>
2  <%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
3  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
4  <%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
5  <html>
6  <head>
7    <meta name="activePage" content="DEPARTMENTS"/>
8    <title><fmt:message key="departments.title"/></title>
9  </head>
10 <body>
11 <div class="container">
12   <div class="row">
13     <div class="span8 offset2">
14       <c:if test="${!empty departments}">
15       <h3><fmt:message key="departments.title"/></h3>
16       <table class="table table-bordered table-striped">
17         <thead>
18         <tr>
19           <th><fmt:message key="departments.table.id"/></th>
20           <th><fmt:message key="departments.table.name"/></th>
21         </tr>
22         </thead>
23         <tbody>
24         <c:forEach items="${departments}" var="department">
25           <tr>
26             <td><c:out value="${department.id}"/></td>
27             <td><c:out value="${department.name}"/></td>
28           </tr>
29         </c:forEach>
30         </tbody>
31       </table>
32       </c:if>
33     </div>
34   </div>
35 </div>
36 </body>
37 </html>
```

Program 4.6: *JSP page to list departments.*

www.manaraa.com

Groovy and Grails provides a similar view technology called *Groovy Server Pages* (GSP) designed to resemble JSP, but with far more flexible and intuitive syntax. Program 4.7 shows a similar GSP file composed mostly of HTML with GSP tags and EL expressions. Convention over configuration omits the need to specify the used tag libraries. Localization messages are fetched from resource files with g:message tag which can handle message parameters with more compact syntax. Core library provides conditional formatting with g:if tag and looping with g:forEach tag.

```
<%@ page contentType="text/html;charset=UTF-8" %>
<html>
<head>
  <meta name="activePage" content="DEPARTMENTS"/>
  <title><g:message code="departments.title"/></title>
  <meta name="layout" content="main">
</head>
<body>
<div class="container">
  <div class="row">
    <div class="span8 offset2">
      <g:if test="{!empty departments}">
      <h3><g:message code="departments.title"/></h3>
      <table class="table table-bordered table-striped">
        <thead>
        <tr>
          <th><g:message code="departments.table.id"/></th>
          <th><g:message code="departments.table.name"/></th>
        </tr>
        </thead>
        <tbody>
        <g:each in="${departments}" var="department">
          <tr>
            <td>${department?.id}</td>
            <td>${department?.name}</td>
          </tr>
        </g:each>
        </tbody>
      </table>
      </g:if>
    </div>
  </div>
</div>
</body>
</html>
```

Program 4.7: *GSP page to list departments.*

Grails provides additional approaches to text escaping. When using a default configuration of a Grails application all of the potentially harmful strings need to be explicitly escaped with a encodeAsHTML() method that Grails provides on the String class. This approach provides a very clear, although a bit verbose syntax. Since this is a method embedded in the String class it is available everywhere in the application, not just in the GSP files:

```
department?.name?.encodeAsHTML()
```

Another approach is to enable auto-escaping in GSP files with a default codec option in the Grails configuration file Config.groovy. A configuration property grails.views.default.codec

المنارة للاستشارات
www.manaraa.com

is used to set the default codec to encode data when using an `${}` expression. When the default codec property is set to `html`, utilizing the `${}` expression in any GSP file will invoke the `encodeAsHTML()` method automatically:

```
grails.views.default.codec = "html"
```

Program 4.8 shows an AngularJS template file composed mostly of standard HTML along with AngularJS directives and markup which defaults to double curly brace notation `{{ }}` to bind expressions to elements. Localization messages are fetched from resource files using a custom filter `i18n` registered with the AngularJS application module.

```
<div class="tab-pane" id="DEPARTMENTS">
  <div class="row">
    <div class="span8 offset2">
      <span ng-show="departments.length">
      <h3>{{'departments.title' | i18n}}</h3>
      <table class="table table-bordered table-striped">
        <thead>
          <tr>
            <th>{{'departments.table.id' | i18n}}</th>
            <th>{{'departments.table.name' | i18n}}</th>
          </tr>
        </thead>
        <tbody>
          <tr ng-repeat="department in departments | orderBy:'id'">
            <td>{{department.id}}</td>
            <td>{{department.name}}</td>
          </tr>
        </tbody>
      </table>
      </span>
    </div>
  </div>
</div>
```

Program 4.8: *AngularJS template tab pane to list departments.*

The `i18n` filter utilizes a jQuery plugin *jquery-i18n-properties* for providing internationalization from `.properties` files as in Java and Groovy projects:

```
angular.module('polyglotVertxModule', []).filter('i18n', function () {
    return function (key, params) {
      if (params) {
        params.unshift(key)
        return jQuery.i18n.prop.apply(this, params)
      }
      return jQuery.i18n.prop(key)
```

AngularJS directives provide conditional formatting with an `ng-show` and similar directives, and looping with an `ng-repeat` directive. The `ng-repeat` directive at the line 14 uses a ready-made `orberBy` filter to return a sorted copy of the array. Text escaping can be done with `ng-bind` directive or with the template markup `{{ }}` which is less verbose.

GSP tags reduce some of the verbosity compared to corresponding JSP tags. For example, a JSP tag `ftm:message` requires multiple lines when parameters are needed:

المنارة للاستشارات
www.manaraa.com

```
<fmt:message key="overview.total">
  <fmt:param value="${departments.size()}"/>
  <fmt:param value="${employees.size()}"/>
</fmt:message>
```

In contrast, Grails equivalent provides a one line solution with args property:

```
<g:message code="overview.total" args="[departments.size(), employees.size()]"/>
```

AngularJS provides even shorter solution with a clear and syntactically simple way for application internalization by using filters. Filters can be chained and additional parameters provided. Following example uses the previously described i18n filter with additional parameters:

```
{{'overview.total' | i18n:[departments.length,countEmployees()]}}
```

All of the above internationalization examples produce the same message to be rendered on the view. AngularJS template directives and markup appear all around less verbose than JSP and GSP technologies, although the comparison remains with these too, since there are exceptions where JSP tags are less verbose compared to GSP tags.

Program 4.9 describes an example JSP form to add a department to the system. The form definition is wrapped with a Spring form:form tag which takes care of the data binding of the form object. Spring provides elegant data binding with form:input and form:errors tags.

```
<form:form method="post" action="add" commandName="department" class="form-horizontal">
  <div class="control-group">
    <form:label cssClass="control-label" path="name">
      <fmt:message key="departments.form.name"/>
    </form:label>
    <div class="controls">
      <form:input path="name"/>
      <form:errors path="name" cssClass="alert alert-error" />
    </div>
  </div>
  <div class="control-group">
    <div class="controls">
      <button class="btn btn-primary"><fmt:message key="departments.form.submit"/></button>
    </div>
  </div>
</form:form>
```

Program 4.9: *Spring form to add a department.*

Program 4.10 shows a similar GSP example wrapping the form definition with a Grails g:form tag. Grails data binding proves to be a bit more verbose concerning error handling, compared to the one line solution in the Spring example. All of the form field errors have to be explicitly checked with the g:hasErrors, since the g:fieldError has no attribute to provide necessary presentation semantics, forcing it to be placed inside an additional span or a similar HTML element.

المنارة للاستشارات
www.manaraa.com

```
<g:form method="post" controller="department" action="add" class="form-horizontal">
  <div class="control-group">
    <label class="control-label" id="name"><g:message code="departments.form.name"/></label>
    <div class="controls">
      <g:textField name="name" value="${department?.name}"/>
      <g:hasErrors bean="${department}" field="name">
        <span class="alert alert-error">
          <g:fieldError field="name" bean="${department}"/>
        </span>
      </g:hasErrors>
    </div>
  </div>
  <div class="control-group">
    <div class="controls">
      <button class="btn btn-primary"><g:message code="departments.form.submit"/></button>
    </div>
  </div>
</g:form>
```

Program 4.10: *Grails form to add a department.*

AngularJS provides data binding of form controls through directives. An example form is shown in Program 4.11. The form definition is wrapped with AngularJS form directive that creates an instance of a FormController. The FormController monitors all control directives and nested forms as well as their state, such as field being $valid or $invalid and $dirty or $pristine. A novalidate attribute is used to indicate that the form is not to be validated on submit by the native form validation support of the browser.

```
<form name="departmentForm" class="form-horizontal" novalidate>
  <div class="control-group">
    <label class="control-label" for="name">{{'departments.form.name' | i18n}}</label>
    <div class="controls">
      <input type="text" id="name" name="name" ng-model="department.name" required/>
      <span class="alert alert-error" ng-show="departmentForm.name.$error.required">
        {{'NotEmpty.department.name' | i18n}}
      </span>
    </div>
  </div>
  <div class="control-group">
    <div class="controls">
      <button class="btn btn-primary" ng-disabled="!departmentForm.$valid"
          ng-click="addDepartment(department)">
        {{'departments.form.submit' | i18n}}
      </button>
    </div>
  </div>
</form>
```

Program 4.11: *AngularJS form to add a department.*

An ng-model directive is used to bind form data to a model object in scope via an input directive control field as shown at line 5. A required attribute enforces the client-side validation. A validation error message can be customized as shown between lines 6 to 8. An $error is an object hash that contains references to all invalid controls or forms. The keys are validation tokens corresponding to the validation attributes such as required, url or email. Values are control arrays or forms that are $invalid with the given error.

المنارة للاستشارات
www.manaraa.com

In addition, most of the HTML markup with CSS presentation semantics enhanced with Twitter Bootstrap, and JavaScript including jQuery implementations used in the views were reusable as whole. Little or none refactoring was required when reusing these in all projects.

4.6 Observations on Groovy as the programming language

Groovy programming language has the potential to improve web development, and it is an easy acquaintance for any Java developer. Most valid Java files are also valid Groovy files. Since Groovy does not require all the elements that Java requires, it is possible for Java developers to gradually amend the familiar Java syntax with Groovy idioms. Thus reducing the boilerplate of Java. As seen in Subsection 4.4.1, it is also possible to integrate a complete Java legacy domain model with Groovy and Grails web application.

Groovy can be used to implement the same as Java with less boilerplate and reduced verbosity, but it is still seen as a controversial programming language, although backed up with developer buzz. Question remains, should it be used in enterprise applications or just as a powerful scripting language on the Java Virtual Machine.

4.6.1 Amount of code

In comparison to Java, Groovy provides a less verbose and more expressive syntax which reduces the amount of code needed to implement the same functionality. Groovy offers some functional programming style advantages over Java, for example, closures can remove the excess use of loop structures. In addition, Grails web framework uses convention over configuration and do not repeat yourself (DRY) principles that reduce a significant amount of configuration required.

4.6.2 Code quality

Code quality does not directly imply anything about the amount of defects in an application. Architectural choices and design decisions are important factors that reflect on the code quality. These prove to be especially important guidelines when something needs to be changed or extended, or when introducing new developers.

Syntactic advantages can improve code quality by making the implementations more clear and intuitive. Groovy has the potential to reduce errors by making the code more fluent. This is because of the increased expressiveness and reduced amount of supportive code. Since Groovy implementations are more compact and have less boilerplate and irrelevant code, the actual business logic becomes more visible. In addition, functional programming style closures can be used to remove loops which tend to be error-prone sections in Java application.

المنارة للاستشارات
www.manaraa.com

4.6.3 Productivity

Groovy can increase developers productivity because the advanced syntax allows to write the intent more clearly without the necessary loops and temporary variables required in Java. Although this requires more comprehensive knowledge of Groovy and its features.

Grails follows the convention over configuration and do not repeat yourself principles which are powerful tools to increase productivity. In addition, the resulting learning curve with a new programming language and framework did not hinder the development process, if more so enhanced it because learning something new was taken as a challenge to improve oneself.

4.7 Observations in web development: traditional methods versus client-side

Vert.x project uses the principles of a thin server architecture to implement a client-side single-page application. Groovy programming language is used to implement the server-side necessities and JavaScript with AngularJS web framework to implement the client-side SPA. Vert.x framework provides seamless interaction over the event bus and is used to integrate the client-side and server-side.

Vert.x provides a lightweight, high-performance alternative to the Java EE programming model described in Section 3.4. Vert.x is designed for modern mobile, web, and enterprise applications. In addition, Vert.x applications should ensure good scalability and be very fast. Vert.x provides a highly-optimized runtime environment utilizing the strengths of the JVM.

AngularJS provides a powerful framework to implement web applications with pure client-side JavaScript. It offers declarative templates with data-binding, dependency injection, comprehensive testability, and an architectural approach named Model View ViewModel (MVVM) pattern based profoundly on the Model-view-controller (MVC) pattern.

4.7.1 Amount of code

Vert.x provides many advantages by following convention over configuration principle and supporting modular application structure. Packaging applications or reusable resources as modules is made easy. Out-of-the-box modules like web servers and database persistors can be configured and deployed in just few lines of code. In addition, an excess amount of configuration is removed by following the principles of convention over configuration.

The Vert.x event bus and bus modules simplify the request mapping significantly client-side applications. Deployed bus modules register on the event bus and provide,

المنارة للاستشارات
www.manaraa.com

for example, necessary CRUD operations for data access working instantly both on the server-side and client-side.

AngularJS provides a fully fledged web framework with reduced verbosity and increased expressiveness compared to many web frameworks. It is designed for web application development all the way to its core. AngularJS conventions and directives reduce large amount of excess jQuery JavaScript code and Twitter Bootstrap presentation semantics.

4.7.2 Code quality

Architectural patterns and design principles reflect on the code quality, and as mentioned before, code quality is not an inverse of the amount of faults in the application. Vert.x and AngularJS frameworks both provide syntactic advantages which improve the code quality by making the implementations more fluent and intuitive. The increased expressiveness and reduced amount of supportive code has the potential to reduce errors.

The Groovy implementation on the server-side is more compact and has less boilerplate and irrelevant code compared to a Java alternative. Similarly AngularJS reduces lines of irrelevant code otherwise required by the JavaScript and jQuery implementations, thus making the implementations more compact, and the actual business logic more visible.

Web application view is often an HTML file or its variant. The HTML page is built using an error-prone string concatenation which is likely to cause problems, for example, with character escaping, and has security issues like JavaScript injection. AngularJS has a different approach since it works on DOM objects and not on strings. The template is written as a standard HTML from which the browser parses the DOM. AngularJS template engine uses the DOM as an input and performs necessary transformations on it directly. This approach provides a continuously updating view based on the model, with no deed for re-merging the model with the template.

4.7.3 Productivity

Client-side single-page applications are quickly becoming de facto standard in web development. Vert.x and AngularJS web frameworks are powerful technologies with advanced practices that support productive web application development.

The choice of programming languages and frameworks should be made based on the complexity of the web application and on the assumption of an increased developer productivity. Selected tools and techniques should make the development experience as enjoyable and as productive as possible. Although implementing a new architecture with new programming languages and frameworks definitely requires a learning process which can hinder the development at the beginning.

المنارة للاستشارات
www.manaraa.com

4.7.4 Testing

AngularJS was designed to be testable from the beginning. It encourages behavior-view separation and comes with bundled mocks, and also takes full advantage of dependency injection. AngularJS also provides an end-to-end scenario runner which eliminates test flakiness by understanding how the AngularJS works inside. In addition, applications using Vert.x framework can be tested using any programming language that is supported.

المنارة للاستشارات
www.manaraa.com

5. EVALUATION

Polyglot programming has the potential to enhance web development in various areas. Different programming languages and frameworks promise an increase in productivity, reduced amount of code and improved code quality that together promote better maintainability. It is also important to realize that polyglot programming and the use of polyglot programming pyramid is not constrained only to web development.

Polyglot programming is a powerful tool for developers with an open mindset. Increase in the amount of required knowledge is seen as a challenge to improve oneself as a developer rather than an obligation to barely overcome in an attempt to finish the task at hand.

This chapter summarizes and evaluates the results of this thesis. This includes a comparison of the observations of this thesis presented in Chapter 4 with the related work and previous results described subsequently. The context of the results are generalized from programming language specific subjects to more general advantages and disadvantages on the polyglot programming approach in the subsequent discussion of the results section.

5.1 Related work and previous results

Related work and previous results are presented from five projects in the context of polyglot programming in web development. These projects are used to provide a more extensive research context and in-depth discussion of results.

In addition, some of the implementations were used to provide more specific technical comparison on different programming languages on the JVM and interoperability with Java. Some of the projects only provide a brief technical description since the technical documentation was omitted due to confidentiality reasons. However, they still provide valuable discussion of results.

Lähteenmäki [24] describes a simple web application to experiment with Scala as a replacement to Java in a real-world situation. A direct comparison is possible since the domain model and the business logic corresponds more or less to the proof of concept project implemented in this thesis. Polyglot programming is represented on the managed runtime where different programming languages interoperate directly.

Fjeldberg [23] provides a case study in polyglot programming context. He describes two web development projects which introduced JRuby with Ruby on Rails into an existing Java environment. Also a web application testing project was evaluated in polyglot

المنارة للاستشارات
www.manaraa.com

programming context. Technical aspects and documentations of the solutions as well as interview transcripts were omitted due to confidentiality reasons.

All of the projects represent a different degree of polyglot programming. Managed runtime is used in the Buypass project where the different programming languages interact directly. A web based extranet solution is also deployed at managed runtime, but the interaction between the application and legacy services is networked. In the last project, the web browser is the integration between the web application and the tests implemented using a different programming language. Last two of the projects are considered polyglot only because the same team was responsible for the implementation of the applications to be integrated.

5.1.1 Example web project with Java and Scala

Lähteenmäki [24] uses Maven to manage projects and their dependencies. All of the project modules are implemented in Java and Scala, and also made interchangeable. Maven handles the build process and can be utilized to choose which implementations to use when building the project. This way parts of the project, for example, the model, services and web application specific parts could be implemented in Java and the view and DAO layer in Scala.

This highlights interoperability as the main polyglot programming feature in his project. However, Lähteenmäki [24] emphasizes an issue in interoperability between Scala and Java. He notes that interfaces have to be Java-compatible if used also from Java, which seems to restrict and complicate the interface implementations in Scala.

Lähteenmäki [24] uses the powerful syntax of Scala to implement solutions to several problem domains with clear risk factors to benefit from the static typing in courtesy of the Scala compiler. In addition, Scala's implicit conversions, operator overloading and type inference is used to enhance the syntax. Scala implementation shows similar advantages over the Java implementation as did the Groovy implementation described in this thesis.

5.1.2 Buypass: JRuby with existing Java libraries

Fjeldberg [23] provides a case study on a web project that experimented with an existing Java solution interoperability with a JRuby and Ruby on Rails high-productivity web framework. Buypass is a company which provides web based identification and payment systems mainly implemented with Java technologies. Goal was to evaluate whether by introducing Rails framework they could increase productivity and reduce the overall cost of web development. In addition, could the framework and the programming language be introduced in their existing architecture.

The first prototype proved that existing Java libraries could be directly accessed with JRuby by making use of the programming language interoperability on the JVM. The

المنارة للاستشارات
www.manaraa.com

second prototype verified that Ruby on Rails could be utilized in their existing architecture without jeopardizing security aspects. However some additional work was required to overcome problems with the architecture using a centralized Java server which provided all data access. [23]

Fjeldberg [23] notes that developers experienced an increase in productivity and ease of learning when using Rails framework. A programming language without need for a compilation, utilizing the convention over configuration and do not repeat yourself principles enhanced the development process significantly. However, some of the productivity gain was counteract by the lack of automatic refactoring and the increased need for tests.

5.1.3 Web based extranet: JRuby with existing Java legacy

Fjeldberg [23] describes another case study on a project to provide an updated solution for an outdated nearly ten years old Java legacy extranet solution. Based on the success of the Buypass project, a cheaper and faster Ruby on Rails solution was offered compared to an original Java solution estimate.

Challenge was to build a new solutions on top of an existing legacy database with hundreds of tables. In addition, the client was reluctant to discard any of their existing Java code. [23]

Fjeldberg [23] states that the interoperability of JRuby with existing Java infrastructure was used as the major selling point for Ruby on Rails. However, the interoperability was not used and the interaction between existing services was achieved with servlets. Again, learning the Ruby on Rails framework was found easier than equivalent in .Net or Java.

5.1.4 Au2sys: Java web application with RSpec and Watir tests

Fjeldberg [23] provides yet another case study. Au2sys is a complex Java web application on the public sector. The challenge was to implement an extensive automated web testing on the web application with complex business rules and a web interface.

Fjeldberg [23] notes that RSpec and Watir were evaluated as best tools available at that time. Watir framework was used to interact with the browser, and RSpec framework to implement the tests. This architectural approach allowed a new programming language and frameworks to be introduced into the project independently from the web application Java code.

5.1.5 Required knowledge

Java is intended for an average programmer [140], whereas Lähteenmäki [24] argues that Scala developers are required more effort and dedication. For example, the shift in

المنارة للاستشارات
www.manaraa.com

paradigm presents a huge step for an average developer. Balancing between expressiveness and verbosity can produce unreadable code if not skilled and careful. In addition, more extensive knowledge is required since more is happening behind the scenes, for example, automatic getters and setters. [24]

Lähteenmäki [24] states that Scala is a powerful tool for a capable programmer to rethink existing structures and design principles with concepts like mixins and implicits. He notes that Java requires supportive libraries, frameworks and technologies, while similar implementations in Scala would be straightforward providing static type safety and no need to learn new tools or syntax. [24]

Fjeldberg [23] states that the increase in knowledge requirements for both the developers and management is perceived as disadvantage. However, developers who participated in the projects reported that learning a new programming language and framework was easier than learning an equivalent framework in Java. Fjeldberg [23] also noted that every time a new programming language is introduced into an application, the amount of developers with enough knowledge to maintain decreases. [23]

5.1.6 Amount of code

Scala reduces the amount of code needed to implement the same functionality because it is less verbose and more expressive. Lähteenmäki [24] suggest that higher level functions with more convenient syntax for anonymous inner classes and functional language structures can remove the excess use of loop structures in imperative programming languages. Similarly, Fjelberg [23] states that dynamic programming languages offer higher level of abstraction and reduce lots of repetition. [23; 24]

Lähteenmäki [24] notes that in order to make a bigger impact, Scala has to be applied in a more advanced level. A smaller codebase can be achieved by re-thinking the whole application architecture with the help of higher level functions, traits, implicits et cetera. Lähteenmäki [24] states that a simple web project cannot demonstrate these ideas at least when the architecture is based mainly on traditional frameworks like Spring and Hibernate. [24]

5.1.7 Code quality

Syntactic advantages can improve code quality by making the implementations more clear and intuitive. Lähteenmäki [24] argues that Scala has the potential to reduce errors by making the code easier, thus reducing the lines of code needed because of the increased expressiveness and reduced amount of supportive code. Therefore, Scala implementations have less boilerplate and irrelevant code, which makes it easier to focus on the relevant parts, thus making the actual business logic more visible. Code quality also benefits from stronger typing, functional paradigm and clean syntax to reduce programming errors

المنارة للاستشارات
www.manaraa.com

like null pointer or class cast exceptions. Thus programmers working with Scala can implement more reliable programs when compared to Java. Lähteenmäki [24] states that immutability and stateless programming removes errors in many common places where Java might pose errors. In addition, functional programming removes loops which tend to be error-prone sections in Java applications. [24]

5.1.8 Productivity

Lähteenmäki [24] argues that developers productivity increases with Scala since the advanced syntax allows to write the intent, without the necessary loops and temporary variables needed in Java. Although this requires comprehensive knowledge of Scala and IDE support. Fjeldberg [23] provides similar results. Convention over configuration and do not repeat yourself principles included in selected programming languages and frameworks presented an increase in productivity. Start developing at once and no compilation cycle was seen as approaches that also increased productivity. [23; 24]

In addition, Fjeldberg [23] highlights that developers had fun while developing with new programming languages and with changed perspective. The fact that the development methodology changed, the developers were forced to write extensive unit testing to ensure type safety, since they could no longer rely on static type safety and compile cycle to run through the whole application. [23]

5.2 Discussion of the results

A repeating pattern is visible in the results. Selecting an appropriate programming language for the task at hand proves to be more productive and better in any way possible over and over again. It is important to realize that this does not imply that one should always choose a new programming language. It simply states that if any programming language is more appropriate for the task at hand, it should be selected or at least considered regardless of the possible increase in required knowledge or the decrease of developers with enough knowledge in hiring or maintenance.

Vert.x represents a new alternative to the Java EE programming model and includes its own runtime environment. Its programming model with asynchronous approach is completely different to any approaches taken elsewhere in the Java ecosystem. This is particularly useful when many open connections needs to be maintained, as WebSockets necessitate. In addition, Vert.x applications should ensure good scalability and be perform very fast. Vert.x utilizes the strengths of the JVM as a highly-optimized runtime environment.

المنارة للاستشارات
www.manaraa.com

5.2.1 Amount of code

Programming languages provide different levels of verbosity and expressiveness which can reduce the amount of code required to implement the same functionality. For example, higher level functions with more convenient syntax for anonymous inner classes and functional language structures can remove the excess use of loop structures in imperative programming languages. Similarly, dynamic programming languages offer a higher level of abstraction and reduce lots of repetition.

Programming languages that follow convention over configuration and do not repeat yourself principles can reduce a significant amount of configuration and boilerplate code. In addition, separation of concerns is a powerful tool when packaging applications or reusable resources as modules. Out-of-the-box modules and applications can provide powerful functionalities in just few lines of code.

Lean programming practices can be used to eliminate waste effort and boilerplate code by favoring "pull" design over "push" design. Since implementations are built only to satisfy the needs of business requirements the necessary code is usually shorter. For example, repository pattern can be used to abstract the data access layer and shield the rest of the application implementation from having to know how the persistence works.

In order to make a bigger impact, the chosen programming language has to be applied in a more advanced level. A smaller codebase can be achieved by re-thinking the whole architecture of the application to correspond with the best practices of the chosen programming language and frameworks.

5.2.2 Code quality

Code quality does not directly imply anything about the amount of defects in an application. Architectural patterns and design principles are important factors that reflect on the code quality. These represent important guidelines when creating, changing or extending an application, or when introducing new developers.

Different programming languages and paradigms present syntactic advantages that can improve code quality by making the implementations more clear and intuitive. This has the potential to reduce errors by making to code more readable. Also different frameworks provide syntactic advantages which can improve the code quality. In addition, the increased expressiveness and reduced amount of supportive code reduces the possibility of erroneous code.

Implementations that are more compact and have less boilerplate and irrelevant code enable developers to focus on the relevant sections, thus making the actual business logic more visible. In addition, stronger typing, functional paradigm and clean syntax can help to reduce programming errors like null pointer or class cast exceptions. Immutability and stateless programming are also powerful tools that remove errors in many common

المنارة للاستشارات
www.manaraa.com

places where imperative programming languages might pose errors. For example, functional programming removes loops which tend to be error-prone sections in imperative programming languages.

5.2.3 Productivity

Developers tend to divide roughly into two categories presented in Subsection 2.11. The language mavens concentrate their effort on gathering knowledge on new programming languages and features, whereas tool mavens familiarize with development tools to enhance productivity. These two perspectives, or more so traits are competitive. Therefore, the more invested in learning programming language features, the bigger is the benefit, although to the exclusion of tool features and vice versa. Polyglot programmers are though as language mavens, although polyglot programming does not imply the use of new and unsupported programming languages and features.

Advanced syntax in several programming languages increase developer productivity, since it allows developers to write the intent without the necessary loops and temporary variables. Although this requires a comprehensive knowledge of the programming language and frameworks used as well as a good IDE support.

When choosing programming languages and frameworks for web development, the decisions should always be made based on the complexity of the web application and by following the assumption of an increased developer productivity. Chosen tools are supposed to make the development experience as enjoyable and productive as possible.

In addition, learning a new programming language and framework seems to enhance developer productivity since learning something new is taken as a challenge to improve oneself. Therefore, the increased amount in learning is countered by the results.

Architectural approaches like thin server architecture and single-page application in client-side web application development are becoming de facto standard. Thus there exists powerful web frameworks and technologies with best practices to make web application development more productive.

المنارة للاستشارات
www.manaraa.com

6. CONCLUSION

This thesis has provided a comprehensive research on polyglot programming in web development and on the Java platform. This includes a literature study on polyglot programming covering associated advantages and disadvantages and how polyglot programming is used in modern-day web application development, testing, deployment, concurrency and in business rules modeling. This thesis also studies how polyglot programming can be utilized on the Java Virtual Machine and describes cases how, when and where it might prove useful. In addition, a new and noteworthy Vert.x framework with high promises on polyglot programming is described.

This thesis enhances the polyglot programming pyramid to work as an architectural pattern to solve and document problem domains suitable for polyglot programming. In addition, four implementations were made to research how polyglot programming can be utilized on the Java Virtual Machine. The observations were compared against related work and previous results from two example web projects and three case study projects presented in polyglot programming in web development context.

6.1 Recommendations

Polyglot programming has the potential to improve web development in various areas. Perceived advantages are increased productivity, reduced amount of code and improved code quality that together promote better maintainability. Perceived disadvantages are a steep learning curve that affects on required knowledge, maintainability, and tool support. Although increase in the amount of required knowledge is also perceived as a challenge to improve oneself as a developer rather than an obligation to barely overcome.

Support for alternative programming languages on the Java Virtual Machine have come a long way. While the compatibility with existing software systems and investments on the Java technology can be retained, alternative programming language can be used provide better solutions to certain problems. This implies that even for a Java technology centered organizations, the automatic choice for every programming task is not always Java. It is fundamentally important to understand the different ways how programming languages can be classified to be able to select the most appropriate programming language for the task at hand.

Polyglot programming divides the different programming languages roughly into three layers. The particular layers are fit for different problem domains and programming chal-

www.manaraa.com

lenges. The stable layer consist of programming languages like Java and Scala to do the heavy lifting, whereas programming languages like Groovy and Clojure are more suitable for tasks in the dynamic layer providing rapid web development. The domain layer consists mainly on domain-specific languages targeting specific problem domains like HTML markup and CSS presentation semantics or SQL.

The core business functionality of an existing production software is almost never the correct place to introduce a new programming language. The core requires a high-grade support with comprehensive test coverage in addition to the proven stability of the programming language. A low-risk area should be chosen for the first deployment of an alternative programming language.

The unique characteristics of each team and project will impact on the decision of which programming language to choose. The nature of the projects and teams should always be assessed by the managers and senior developers when considering to introduce a new programming language. There are no universal right answers, only guidelines. An agile web company could choose a Groovy on Grails for its productivity and relatively deep pool of developers to attract young developers and grow the team quickly. Meanwhile, a small project group of experienced enthusiasts could choose Clojure for its clean design, sophistication and power, and disregard the conceptual complexity and possible difficulties when hiring.

6.2 Future work

The enhanced polyglot programming pyramid describes a working solution to document and solve different problem domains in polyglot programming context. How ever it should be further studied that in which phase of the software development process it would prove to be most beneficial for the project and developers in it.

Case studies in real-world projects should be conducted to evaluate the advantages and disadvantages more comprehensively to be able to draw proper conclusions and to generalize. Further work should be conducted to compare the effects of different approaches and alternatives, although such a comparison might prove difficult unless simply comparing the quality and other aspects of the projects. It would also be beneficial to study the effects of the steeper learning curve in polyglot programming, and more precisely to find factors that define different types of developers that are willing to try out and actually benefit from polyglot programming.

In addition, research should be conducted to gather information on the thought process and decisions that define the chosen programming languages, frameworks and libraries that contribute in polyglot programming context of the project. It would be most beneficial to find a polyglot programming project that has had more disadvantages and negative effects due to the chosen approach or, in other words, to study where did the polyglot programming approach of the project go wrong and why. For example, was the process

المنارة للاستشارات
www.manaraa.com

of choosing the programming languages valid and were the programming languages and tools evaluated properly at the beginning.

This thesis concentrated more on the perceived advantages and disadvantages from the developers perspective. Additional research should also be conducted from the application performance perspective since it plays a huge part in enterprise applications. In addition, this thesis researched polyglot programming mainly on the Java platform. Other platforms like .Net should also be assessed.

المنارة للاستشارات
www.manaraa.com

REFERENCES

[1] Watts, N. 2008. Even more than polyglot programming. [WWW]. [Accessed on 02.12.2012]. Available at: http://thewonggei.wordpress.com/2008/01/22/even-more-than-polyglot-programming/

[2] Beardsmore, H. 1978. Polyglot Literature and Linguistic Fiction. International Journal of the Sociology of Language. Volume 15. Issue 1. Published online 29.07.2009. 12p.

[3] Watts, A. 2008. Mixing Programming Languages. [WWW]. [Accessed on 06.04.2013]. Available at: http://thewonggei.wordpress.com/2008/01/22/mixing-programming-languages/

[4] Ford, N. 2008. Polyglot Programming Parallels Perched Precariously per Patron Participation. [WWW]. [Accessed on 06.04.2013]. Available at: http://memeagora.blogspot.fi/2008/01/polyglot-programming-parallels-perched.html

[5] Morgan, J. 2013. What Is a Polyglot Programmer? [WWW]. [Accessed on 07.08.2013]. Available at: http://www.jeremymorgan.com/blog/programming/what-is-a-polyglot-programmer/

[6] Brooks, F. 1987. No silver bullet: Essence and accidents of software engineering. IEEE Computer. Volume 20. Number 4. 9p.

[7] Kullbach, B., Winter, A., Dahm, P. and Ebert, J. 1998. Program comprehension in multi-language systems. IEEE Reverse Engineering. Fift Working Conference on Proceedings. 9p.

[8] Bini, O. 2008. Connecting languages (or polyglot programming example 1). [WWW]. [Accessed on 02.12.2012]. Available at: http://olabini.com/blog/2008/04/connecting-languages-or-polyglot-programming-example-1/

[9] Schink, H. and Kuhlemann, M. and Saake, G. and Lämmel, R. 2011. Hurdles in multi-language refactoring of hibernate applications. International Conference on Software And Data Technologies (ICSOFT). 5 p.

[10] Schink, H. and Kuhlemann, M. 2010. Hurdles in Refactoring Multi-Language Programs. Technical report, Otto-von-Guericke-Universität, Magdeburg/Germany.

المنارة للاستشارات

www.manaraa.com

[11] Mayer, P. and Schroeder, A. 2012. Cross-Language Code Analysis and Refactoring. IEEE 12th International Working Conference on Source Code Analysis and Manipulation. 10 p.

[12] Jackson, A. and Clarke, S. 2004. Sourceweave.net: Cross-language aspect-oriented programming. Generative Programming and Component Engineering. Springer. 25p.

[13] Bini, O. 2008. Fractal programming. [WWW]. [Accessed on 02.12.2012]. Available at: http://olabini.com/blog/2008/06/fractal-programming/

[14] Bini, O. 2008. Viability of Java and the stable layer. [WWW]. [Accessed on 02.12.2012]. Available at: http://olabini.com/blog/2008/01/viability-of-java-and-the-stable-layer/

[15] Bini, O. 2008. Language Explorations. [WWW]. [Accessed on 02.12.2012]. Available at: http://olabini.com/blog/2008/01/language-explorations/

[16] Saluja, N. and Dhiman, P. 2008. Language Oriented Programming: The Next Programming Paradigm. Proceeding. 7p.

[17] Fowler, M. 2005. Language workbenches: The killer-app for domain specific languages. [WWW]. [Accessed on 28.11.2012]. Available at: http://martinfowler.com/articles/languageWorkbench.html

[18] Dmitriev, S. 2004. Language oriented programming: The next programming paradigm. JetBrains onBoard. Volume 1. Number 2.

[19] Ward, M. 1994. Language-oriented programming. Software Concepts and Tools. Volume 15. Number 4. 15p.

[20] Meyer, B. 2002. Polyglot Programming. [WWW]. [Accessed on 02.12.2012]. Available at: http://www.drdobbs.com/polyglot-programming/184414854

[21] Ford, N. 2006. Polyglot Programming. [WWW]. [Accessed on 02.12.2012]. Available at: http://memeagora.blogspot.com/2006/12/polyglot-programming.html

[22] Ford, N. 2008. Polyglot Programming. In The ThoughtWorks anthology: Essays on software technology and innovation. The pragmatic Programmers. 10p.

[23] Fjeldberg, H. 2008. Polyglot Programming: A business perspective. Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway. 75 p.

[24] Lähteenmäki, J-M. 2010. Using Scala to Boost Web Development. Master's thesis, Tampere University of Technology. Tampere, Finland. 60 p.

المنارة للاستشارات
www.manaraa.com

[25] Delorey, D., Knutson, C. and Chun, S. 2007. Do programming languages affect productivity? a case study using data from open source projects. IEEE. FLOSS'07 First International Workshop on Emerging Trends in FLOSS Research and Development. 5p.

[26] Maxwell, K., Van Wassenhove, L. and Dutta, S. 1996. Software development productivity of European space, military, and industrial applications. IEEE Transactions on Software Engineering. Volume 22. Number 10. 13p.

[27] Maxwell, K. and Forselius, P. 2000. Benchmarking software development productivity. IEEE Software. Volume 17. Number 1. 9p.

[28] Brooks, F. 1995. Calling the shots. The mythical man-month: essays on software engineering. Addison-Wesley. Anniversary edition. 8p.

[29] Cataldo, M., Herbsleb, J. and Carley, K. 2008. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement. 10p.

[30] Cataldo, M., Bass, M., Herbsleb, J. and Bass, L. 2007. On coordination mechanisms in global software development. Second IEEE International Conference on Global Software Engineering, 2007. ICGSE 2007. 10p.

[31] Cataldo, M., Wagstrom, P., Herbsleb, J. and Carley, K. 2006. Identification of coordination requirements: implications for the Design of collaboration and awareness tools. Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work. 10p.

[32] De Souza, C. 2005. On the relationship between software dependencies and coordination: field studies and tool support. Doctoral dissertation, University of California. 186p.

[33] Sullivan, K., Griswold, W., Cai, Y. and Hallen, B. 2001. The structure and value of modularity in software design. ACM SIGSOFT Software Engineering Notes. Volume 26. Number 5. 10p.

[34] Brynjolfsson, E. 1993. The productivity paradox of information technology. ACM. Communications of the ACM. Volume 36. Number 12. 12p.

[35] Pareto, L. 2001. Types for Crash Prevention. Thesis, Chalmers University of Technology.

[36] Flanagan, C. and Abadi, M. 1999. Types for safe locking. Programming Languages and Systems. Springer. 18p.

المنارة للاستشارات
www.manaraa.com

[37] Vinoski, S. 2008. Multilanguage Programming. IEEE, Internet Computing. Volume 12. Number 3. 3p.

[38] Ostrand, T., Weyuker, E. and Bell, R. 2005. Predicting the location and number of faults in large software systems. IEEE Transactions on Software Engineering. Volume 31. Number 4. 16p.

[39] Gaffney, J. 1984. Estimating the Number of Faults in Code. IEEE Transactions on Software Engineering. Volume 10. Number 4.

[40] Lipow, M. 1982. Number of Faults per Line of Code. IEEE Transactions on Software Engineering. Volume 8. Number 4. 3p.

[41] Spiewak, D. 2008. The Plague of Polyglotism. [WWW]. [Accessed on 23.12.2012]. Available at: http://www.codecommit.com/blog/java/the-plague-of-polyglotism/

[42] Hunt, A. and Thomas, D. 2000. The pragmatic programmer: from journeyman to master. Addison-Wesley Professional. 349p.

[43] Duarte, G. 2008. Language Dabbling Considered Wasteful. [WWW]. [Accessed on 23.12.2012]. Available at: http://duartes.org/gustavo/blog/post/language-dabbling-considered-wasteful/

[44] Nilsson, N. 2008. Should you really learn another language. [WWW]. [Accessed on 23.12.2012]. Available at: http://www.infoq.com/news/2008/05/should-you-learn-languages/

[45] Braithwaite, R. 2007. The challenge of teaching yourself a programming language.[WWW]. [Accessed on 23.12.2012]. Available at: http://raganwald.com/2007/10/challenge-of-teaching-yourself.html

[46] Norvig, P. 2001. Teach Yourself Programming in Ten Years. [WWW]. [Accessed on 23.12.2012]. Available at: http://norvig.com/21-days.html

[47] Graham, P. 2004. Hackers & painters: big ideas from the computer age. O'Reilly Media, Incorporated.

[48] Neward, T. 2009. The Polyglot Programmer. Mixing And Matchin Languages. [WWW]. [Accessed on 06.08.2013]. Available at: http://msdn.microsoft.com/en-us/magazine/dd483224.aspx

[49] Steele, O. 2004. The IDE Divide. [Accessed on 17.07.2013]. Available at: http://osteele.com/posts/2004/11/ides

المنارة للاستشارات
www.manaraa.com

[50] Keznikl, J., Malohlava, M., Bures, T. and Hnetynka, P. 2011. Extensible Polyglot Programming Support in Existing Component Frameworks. IEEE 37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA). 10p.

[51] Martin Fowler. 2007. RailsConf. [WWW]. [Accessed on 07.04.2013]. Available at: http://www.martinfowler.com/bliki/RailsConf2007.html

[52] Martin Fowler. 2009. Ruby at ThoughtWorks. [WWW]. [Accessed on 07.04.2013]. Available at: http://martinfowler.com/articles/rubyAtThoughtWorks.html

[53] 451 Research. 2012. The Rise of Polyglot Programming. [WWW]. [Accessed on 06.08.2013]. Available at: https://451research.com/report-long?icid=2265

[54] Eyler, P. 2006. A New JRuby Interview and More. [WWW]. [Accessed on 07.04.2013]. Available at: http://www.linuxjournal.com/node/1000103

[55] Janzen, D. and Saiedian, H. 2005. Test-driven development concepts, taxonomy, and future direction. IEEE Computer. Volume 38. Issue 9. 8p.

[56] North, D. 2006. Introducing BDD. Better Software. [WWW]. [Accessed on 29.04.2013]. Available at: http://dannorth.net/introducing-bdd/

[57] Beck, K. 1999. Embracing Change with Extreme Programming. IEEE Computer. Volume 32. Issue 10. 8p.

[58] Thomas, D. and Hunt, A. 2002. Mock objects. IEEE Software. Volume 19. Issue 3. 3p.

[59] jMock. A library that supports test-driven development of Java code with mock objects. [WWW]. [Accessed on 29.04.2013]. Available at: http://jmock.org/

[60] Mocha. A mocking and stubbing library for Ruby [WWW]. [Accessed on 29.04.2013]. Available at: https://github.com/freerange/mocha

[61] Tucker, A. and Noonan, R. 2007. Programming languages: principles and paradigms. McGraw-Hill Higher Education. Second Edition. 411p.

[62] Thompson, S. 2011. Haskell: the craft of functional programming. Addison-Wesley. Third Edition. 585p.

[63] Hughes, J. 1989. Why Functional Programming Matters. The computer journal. Volume 32. Issue 2. 10p.

[64] Erlang. [WWW]. [Accessed on 04.04.2013]. Available at: http://www.erlang.org/

المنارة للاستشارات
www.manaraa.com

[65] Armstrong, J. 2007. Programming Erlang: software for a concurrent world. O'Reilly & Associates, Incorporated. 515p.

[66] Ghodsi, A. and Armstrong, J. 2007. Apache vs. Yaws. [WWW]. [Accessed on 29.04.2013]. Available at: http://www.sics.se/ joe/apachevsyaws.html

[67] Yaws. A HTTP High performance Web Server written in Erlang. [WWW]. [Accessed on 29.04.2013]. Available at: http://hyber.org/

[68] Apache httpd. The Apache HTTP Server Project. [WWW]. [Accessed on 29.04.2013]. Available at: http://httpd.apache.org/

[69] Onnen, E. 2007. Worst Measurement Ever. [WWW]. [Accessed on 29.04.2013]. Available at: http://mykakotopia.blogspot.fi/2007/10/worst-measurement-ever.html

[70] Hoff, T. 2008. New Facebook Chat Feature Scales To 70 Million Users Using Erlang. [WWW]. [Accessed on 29.04.2013]. Available at: http://highscalability.com/blog/2008/5/14/new-facebook-chat-feature-scales-to-70-million-users-using-e.html

[71] Letuchy, E. 2008. Facebook Chat. [WWW]. [Accessed on 29.04.2013]. Available at: http://www.facebook.com/note.php?note_id=14218138919

[72] Apache Thrift. Software framework for scalable cross-language services development. [WWW]. [Accessed on 29.04.2013]. Available at: http://thrift.apache.org/

[73] Slee, M., Agarwal, A. and Kwiatkowski, M. 2007. Thrift: Scalable Cross-Language Services Implementation. Facebook, 156 Univesity Ave, Palo Alto, CA. [WWW]. [Accessed on 29.04.2013]. Available at: http://thrift.apache.org/static/files/thrift-20070401.pdf

[74] Apache MPM worker. Multi-Processing Module implementing a hybrid multi-threaded multi-process web server. [WWW]. [Accessed on 30.04.2013]. Available at: http://httpd.apache.org/docs/2.0/mod/worker.html

[75] July 2013 Web Server Survey. [WWW]. [Accessed on 04.07.2013]. Available at: http://news.netcraft.com/archives/category/web-server-survey/

[76] nginx ("engine x"). An open source web server. [WWW]. [Accessed on 30.04.2013]. Available at: http://nginx.org/

[77] Jagielski, J. 2011. Apache httpd v2.4: Hello Cloud: Buy you a drink? [WWW]. [Accessed on 30.04.2013]. Available at: http://people.apache.org/ jim/presos/ACNA11/Apache_httpd_cloud.pdf

المنارة للاستشارات
www.manaraa.com

[78] Business Natural Language material. [WWW]. [Accessed on 30.04.2013]. Available at: http://blog.jayfields.com/2006/07/business-natural-language-material.html

[79] Martin Fowler. 2008. Domain Specific Language. [WWW]. [Accessed on 29.04.2013]. Available at: http://martinfowler.com/bliki/DomainSpecificLanguage.html

[80] Drools - The Business Logic integration Platform. [WWW]. [Accessed on 30.04.2013]. Available at: http://www.jboss.org/drools/

[81] Jess, the Rule Engine for the JavaTM Platform. [WWW]. [Accessed on 30.04.2013]. Available at: http://herzberg.ca.sandia.gov/

[82] InRule - The Premier .NET Business Rule Solution for the Microsoft Platform. [WWW]. [Accessed on 30.04.2013]. Available at: http://www.inrule.com/

[83] BizTalk Server. [WWW]. [Accessed on 30.04.2013]. Available at: http://www.microsoft.com/en-us/biztalk/default.aspx

[84] Blank, S. 2011. Startup Suicide - Rewriting the Code. [WWW]. [Accessed on 06.04.2013]. Available at: http://steveblank.com/2011/01/25/startup-suicide-%E2%80%93-rewriting-the-code/

[85] Feathers, M. 2004. Working effectively with legacy code. Prentice Hall. 456p.

[86] Evans, B. and Verburg, M. 2011. Polyglot Programming on the JVM. Java magazine, Oracle. 3 p.

[87] Evans, B. and Verburg, M. 2012. The Well-Grounded Java Developer: Vital techniques of Java 7 and polyglot programming. Manning. 496 p.

[88] Mandelbrot, B. 1983. The fractal geometry of nature. Henry Holt and Company. 468p.

[89] Gouyet, J-F. and Mandelbrot, B. 1996. Physics and fractal structures. Paris: Masson. 514p.

[90] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. 1995. Design Patterns: Elements of Reusable Object-Oriented Software. 416p.

[91] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M. 1996. Pattern-Oriented Software Architecture: A System of Patterns. Wiley. 465p.

[92] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M. 2007. Pattern-Oriented Software Architecture: On Patterns and Pattern Languages. Wiley. 465p.

المنارة للاستشارات
www.manaraa.com

[93] Perkins, L. 2009. What the Obama IT team teaches us about polyglot programming. [WWW]. [Accessed on 06.08.2013]. Available at: http://blog.appfog.com/what-the-obama-it-team-teaches-us-about-polyglot-programming/

[94] Raible, M. 2009. Comparing Kick-Ass Web Frameworks at The Rich Web Experience. [WWW]. [Accessed on 30.12.2012]. Available at: http://raibledesigns.com/rd/entry/comparing_kick_ass_web_frameworks

[95] Sebesta, R. 2009. Concepts of Programming Languages. Addison-Wesley. 765p.

[96] Syme, D., Petricek, T. and Lomov, D. 2011. The F Asynchronous Programming Model. Practical Aspects of Declarative Languages. Springer Berlin Heidelberg. 15p.

[97] Petricek, T., Syme, D. 2011. Joinads: a retargetable control-flow construct for reactive, paraller and concurrent programming. Practical Aspects of Declarative Languages. Springer Berlin Heidelberg. 15p.

[98] Loui, R. 2008. In praise of scripting: Real programming pragmatism. IEEE Computer. Volume 41. Issue 7. 5p.

[99] Morin, R. and Brown, V. 1999. Scripting Languages. A Cross-OS Perspective. MacTech. Volume 15. Issue 9. [WWW]. [Accessed on 01.05.2013]. Available at: http://www.mactech.com/articles/mactech/Vol.15/15.09/ScriptingLanguages/index.html

[100] O'Grady, S. 2012. The RedMonk Programming Language Rankings: September 2012. [WWW]. [Accessed on 14.01.2013]. Available at: http://redmonk.com/sogrady/2012/09/12/language-rankings-9-12/

[101] Dahlke, F. 2012. JVM language popularity. [WWW]. [Accessed on 14.01.2013]. Available at: http://ubercode.de/blog/jvm-language-popularity

[102] Schwartz, A. 2010. Developer "flow." Better for all? [WWW]. [Accessed on 14.01.2013]. Available at: http://gristmillanalytics.com/blog/?p=763

[103] Java Platform Standard Edition 7 Documentation. [WWW]. [Accessed on 13.01.2013]. Available at: http://docs.oracle.com/javase/7/docs/

[104] Gosling, J., Joy, B., Steele, G., Bracha, G. and Buckley, A. 2012. The Java Language Specification. Java SE 7 Edition. 640p.

[105] Lindholm, T., Yellin, F., Bracha, G. and Buckley, A. 2012. The Java Virtual Machine Specification. Java SE 7 Edition. 649p.

المنارة للاستشارات
www.manaraa.com

[106] JSR 223: Scripting for the Java Platform. Java Community Process. [WWW]. [Accessed on 13.01.2013]. Available at: http://jcp.org/en/jsr/detail?id=223

[107] Lyman, J. 2012. Open Source Lives in Polyglot Programming. [WWW]. [Accessed on 02.12.2012]. Available at: http://www.linuxinsider.com/story/76343.html

[108] OpenJDK. The place to collaborate on an open-source implementation of the Java Platform, Standard Edition, and related projects. [WWW]. [Accessed on 13.01.2013]. Available at: http://openjdk.java.net/

[109] The Da Vinci Machine Project, a multi-language renaissance for the Java Virtual Machine architecture. OpenJDK. [WWW]. [Accessed on 13.01.2013]. Available at: http://openjdk.java.net/projects/mlvm/

[110] JSR 292: Supporting Dynamically Typed Languages on the Java Platform. Java Community Process. [WWW]. [Accessed on 13.01.2013]. Available at: http://jcp.org/en/jsr/detail?id=292

[111] Buckley, A. 2011. Toward a Universal VM. Java magazine, Oracle. 3 p.

[112] JSR 335: Lambda Expressions for the Java Programming Language. Java Community Process. [WWW]. [Accessed on 13.01.2013]. Available at: http://jcp.org/en/jsr/detail?id=335

[113] Java. [WWW]. [Accessed on 13.01.2013]. Available at: http://java.com/

[114] Design Goals of the Java Programming Language. 1997. Sun Microsystems, Inc. [WWW]. [Accessed on 13.01.2013]. Available at: http://www.oracle.com/technetwork/java/intro-141325.html

[115] Java SE 7 Documentation. Type Inference for Generic Instance Creation. [WWW]. [Accessed on 05.04.2013]. Available at: http://docs.oracle.com/javase/7/docs/technotes/guides/language/type-inference-generic-instance-creation.html

[116] TIOBE. 2013. TIOBE Programming Community Index. [WWW]. [Accessed on 13.01.2013]. Available at: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

[117] LangPop. 2011. Programming Language Popularity. [WWW]. [Accessed on 13.01.2013]. Available at: http://www.langpop.com/

[118] Groovy. [WWW]. [Accessed on 04.03.2013]. Available at: http://groovy.codehaus.org/

المنارة للاستشارات
www.manaraa.com

[119] Scala. [WWW]. [Accessed on 13.01.2013]. Available at: http://www.scala-lang.org/

[120] Clojure. [WWW]. [Accessed on 13.01.2013]. Available at: http://clojure.org/

[121] Vert.x. [WWW]. [Accessed on 19.07.2013]. Available at: http://vertx.io/

[122] Wolf, E. 2012. Vert.x - an asynchronous, event-driven Java web framework. [WWW]. [Accessed on 03.01.2013]. Available at: http://www.h-online.com/developer/features/Vert-x-an-asynchronous-event-driven-Java-web-framework-1615383.html

[123] Netty. [WWW]. [Accessed on 03.04.2013]. Available at: http://netty.io/

[124] Hazelcast. [WWW]. [Accessed on 03.04.2013]. Available at: http://www.hazelcast.com/

[125] Cholakian, A. 2012. Vert.x: Why the JVM May Put Node.js on the Ropes. [WWW]. [Accessed on 03.01.2013]. Available at: http://blog.andrewvc.com/vertx-node-on-ropes

[126] Bintray. Serving Your Binaries. [WWW]. [Accessed on 05.08.2013]. Available at: https://bintray.com/

[127] The Vert.x Central Module Repository. [WWW]. [Accessed on 03.04.2013]. Available at: https://github.com/vert-x/vertx-mods

[128] Node.js. [WWW]. [Accessed on 04.04.2013]. Available at: http://nodejs.org/

[129] Maven. [WWW]. [Accessed on 15.07.2013]. Available at: http://maven.apache.org/

[130] jQuery. [WWW]. [Accessed on 15.07.2013]. Available at: http://jquery.com/

[131] Twitter Bootstrap. [WWW]. [Accessed on 15.07.2013]. http://twitter.github.io/bootstrap/

[132] Spring Framework. [WWW]. [Accessed on 13.01.2013]. Available at: http://www.springsource.org/spring-framework

[133] Hibernate. [WWW]. [Accessed on 13.01.2013]. Available at: http://www.hibernate.org/

[134] Grails. [WWW]. [Accessed on 13.01.2013]. Available at: http://grails.org/

[135] SockJS. [WWW]. [Accessed on 06.04.2013]. Available at: https://github.com/sockjs/

المنارة للاستشارات
www.manaraa.com

[136] AngularJS by Google. HTML enhanced for web apps! [WWW]. [Accessed on 06.04.2013]. Available at: http://angularjs.org/

[137] SiteMesh. [WWW]. [Accessed on 08.07.2013]. Available at: http://wiki.sitemesh.org/

[138] Java Persistence API. [WWW]. [Accessed on 10.07.2013]. Available at: http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html

[139] MongoDB. [WWW]. [Accessed on 06.04.2013]. http://www.mongodb.org/

[140] Gosling, J. and McGilton, H. 1995. The Java TMLanguage Environment. A White Paper. Sun Microsystems Computer Company.

المنارة للاستشارات
www.manaraa.com