Service-Oriented Design Methodology to reduce Software Development Complexity

Ruth Patricia Cortez Recinos
Graduate Department of Computer and Information Systems
University of Aizu

DISSERTATION SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS FOR DEGREE OF DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE AND ENGINEERING

2015
I hereby declare that this dissertation is entirely the result of my own work except where otherwise indicated. I have used only the resources given in the list of references.

Date Candidate's Signature

Aizu-Wakamatsu 27.02.2015

© Copyright by Ruth Patricia Cortez Recinos 2015
All rights reserved.
The dissertation titled

Service-Oriented Design Methodology to reduce Software Development Complexity

by
Ruth Patricia Cortez Recinos
is reviewed and approved.

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Main Referee

Professor
Alexander Vazhenin

Date
21.02.2015

Professor
Subhash Bhalla

Date
25.02.2015

Professor
Vitaly Klyuev

Date
26.02.2015

Senior Associate Professor
Incheon Paik

Date
27.02.2015

University of Aizu
March, 2015
Acknowledgements

This thesis is specially dedicated to my parents Virgilio A. Cortez, and Marta Ruth de Cortez, my grandmother Otilia Trejo, my aunt Patricia Lang, and my siblings whose unconditional love and support have inspired me to pursue my goals.

I would like to express my deep gratitude to my advisor Professor Alexander Vazhenin for his support and guidance during my research work, and time in Japan. I would also like also to express my gratitude to Professor Subhash Bhalla, Professor Vitaly Klyuev, and Professor Incheon Paik, for their time and suggestions reviewing my thesis. Special thanks to Professor John Brine for his advice and valuable insights about research and life.

The University of Aizu has provided an excellent multicultural environment, from which my study and personal life has been enriched. I express my deep appreciation to University Staff for their great support helping international students. Special thanks to ALO staff and Students Affairs Division.

At the University, I have met colleagues and friends with whom we shared ideas, as well as good moments in life: Dr. Aastha Maadan, Ms. Daria Vazhenina, Ms. Quanhua Piao, Dr. Shelly Sachdeva, Ms. Saphina Machome, Dr. Dmitry Vazhenin, Dr. Julian Villegas, Dr. Shinji Kikuchi, and many others along this years in Aizu. I also would like to thank my second family in Sendai, specially Dr. Adriana Ledezma, Ms. Lilin Chaw, Dr. Sulastri Yahya, and Ms. Olga Yuil. I also would like to thank to my laboratory group, specially to Mr. Kensaku Hayashi, and Mr. Hayato Tan for their friendship and also help with my Japanese struggles.

Finally, this thesis would not have been possible without the support of MEXT Scholarship Monbukagakusho.
Abstract

Re-usability and interoperability of applications and data are of vital importance with the increasing adoption of Cloud Computing, Mobile technologies, and Big data. An enabler to these technologies is Service-Oriented Architecture (SOA) introducing principles, strategies, and patterns to develop loosely coupled, standard-based software components where services are the main building blocks. However, implementing a SOA solution is currently a complex and costly endeavor.

Challenges. One of the main benefits of SOA is to reduce the complexity of enterprise applications once is fully adopted. The architecture is composed in several layers to conceptually organize all the elements involved. It has to deal with Basic services functionality such as capability, interface, quality of services, publication, discovery, and selection. As well as composite services functionality for aggregation, coordination, conformance and semantics. Finally, managing services through their life-cycle, service level agreement (SLA), auditing, monitoring, dynamic provision, and scalability. The efforts assembling the necessary layers and functionality can be a complex and time consuming task.

Integration approaches such as point to point, are not suitable for SOA due to the direct coupling between client and service, which is difficult to maintain when the application complexity increases such as in an enterprise environment or Cloud. The Enterprise Service Bus (ESB) is a middle layer design pattern to handle the connections between services as one to many, and comprising integration, intelligent routing and composition capabilities, reducing most of the maintenance and integration complexity. However, harnessing all this concepts into SOA enterprise architecture requires assembling together a number of products which are usually costly, and requires specialized IT staff. It means that special training should be provided to work with new tools, standards, and programming models. To better enable the benefits of Cloud technologies it is necessary that SOA is not only adopted by big size enterprises, but that it becomes the development model of next generation of applications of any type and size of company.

Solution approach. This research proposes a methodology and design pattern named as Virtual Model-View-Controller, and a programming approach to reduce software development complexity of SOA implementation.
The methodology is motivated from the traditional Model-View-Controller (MVC) design pattern that organizes the applications in three main components. However, the MVC pattern is based on Object Oriented concepts, and it does not address SOA key elements (services, compositions, and service management). Therefore, this work proposes an extension of the controller with the Enterprise Service Bus (ESB) pattern comprising the integration logic, and performing functions for data and protocol transformation, message exchange, and intelligent routing necessary for service composition. The link between view and model is removed and reorganized it within the enhanced controller as a virtual link. Dependency Injection (DI) pattern is introduced in the ESB for service virtualization and workflow compositions. It allows decoupling services from hard-coded dependencies on its collaborators, including expensive initialization logic; facilitating the test of individual services.

The decoupling achieved protects the privacy of the business logic (model) from remote reference access, and reduces the complexity of deploying and assembling software components. This means that the efforts of additional code for integration and wrapping the service to be deployed in a service environment are greatly reduced for the developer. Moreover, if there is a change in the model, it can be deployed without requiring changes in the view, instead are handled at the controller level.

Research methods and Key contributions. The first contribution is a novel methodology Virtual-MVC to reduce software development complexity of SOA applications, and its corresponding programming approach. The decoupling of the view from the model allows the programmers to concentrate on building new functionality and services without bothering on how the services will be exposed, consumed, and maintained. The integration logic is outsourced from the developer’s work, and the view and model can be built independently. This means a faster deployment time, higher productivity, and facilitates service maintenance with minimum impact to the view.

The second contribution is a development framework based on the Virtual-MVC pattern was implemented using Web services standards, J2EE, and WebSphere application server. Several case scenarios were developed for E-Learning and Visual programming, formalizing the development approach. The programming approach was taught and evaluated in a Software Engineering course at University of Aizu. The evaluation of the methodology shows positive results in complexity reduction and deployment time. Findings of this evaluation demonstrated that the pattern was understandable even for novice programmers (not specialized skills required). Analysis with related MVC programming approaches demonstrated the simplification of integration code, by calculating an index of integration code based on the components required to implement the same Web based application using
different programming approaches. Virtual-MVC obtained the lowest ratio.

Finally, the Virtual-MVC programming approach is compared with Service Component Architecture, a related SOA programming approach. The main outcome of the analysis is the decoupling between services itself, which in Virtual-MVC does not keep any internal reference in the business logic minimizing the need of modifying the business logic to participate in new compositions.

The competitive advantages of faster deployment and cost reduction for enterprises are one of the main benefits. However, research environments can benefit from SOA adoption where small to medium-sized software teams are assembled. The rotation of students is higher than in a company, which is a drawback to advance software implementation. By introducing Virtual-MVC the students can be assigned with a set of functionality instead of the whole application, and it can be constructed by aggregations.
Contents

List of Figures x
List of Tables xii

1 Introduction 1
  1.1 Overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
  1.2 Thesis contribution . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
  1.3 Chapters overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

I Part I: Related Work and Background 7

2 Service-Oriented-Architecture Implementation and Complexity Issues 8
  2.1 Service-Oriented Architecture . . . . . . . . . . . . . . . . . . . . . . . 8
  2.2 Integration of applications and Service-Oriented Architecture . . . . . . 11
    2.2.1 Big Web Services . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
    2.2.2 REST Web Services . . . . . . . . . . . . . . . . . . . . . . . . . 14
    2.2.3 Point-Point integration . . . . . . . . . . . . . . . . . . . . . . . 15
    2.2.4 Middleware and Enterprise Service Bus . . . . . . . . . . . . . . 16
    2.2.5 Service Virtualization . . . . . . . . . . . . . . . . . . . . . . . . 17
    2.2.6 Service Component Architecture . . . . . . . . . . . . . . . . . . 18
  2.3 Evaluating SOA Complexity . . . . . . . . . . . . . . . . . . . . . . . . 19
  2.4 Chapter Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

3 Model-View-Controller and related Design Patterns 23
  3.1 MVC and Related Design Patterns . . . . . . . . . . . . . . . . . . . . . . 23
    3.1.1 Presentation Model . . . . . . . . . . . . . . . . . . . . . . . . . . 23
    3.1.2 Model-View View Model . . . . . . . . . . . . . . . . . . . . . . . 24
    3.1.3 Model-View-Presenter . . . . . . . . . . . . . . . . . . . . . . . . 24
  3.2 MVC Frameworks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
  3.3 Chapter Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
## II Part II: Virtual Model-View-Controller Methodology: Concepts and Design Strategies

4 Virtual MVC Methodology, Design Pattern, and Architecture  
  4.1 Motivation ............................................. 30  
  4.2 Proposed Virtual Model-View-Controller Design Pattern .......... 33  
  4.3 Controller Architecture .................................... 35  
  4.4 Dependency Injection on Virtual-MVC  
  4.5 Virtual Model-View-Controller Architecture  
  4.6 Chapter Summary ......................................... 41

5 Virtual Model-View-Controller Framework and Programming Strategy  
  5.1 Virtual Model-View-Controller development framework .......... 43  
    5.1.1 Framework requirements ................................ 44  
    5.1.2 Application Layout ..................................... 44  
    5.1.3 Framework settings ..................................... 46  
  5.2 Programming Strategy ...................................... 46  
    5.2.1 Application Case Scenario  
    5.2.2 Designing the Model  
    5.2.3 Designing the Controller  
    5.2.4 Designing the View  
    5.2.5 Assembling the application  
  5.3 Chapter Summary ......................................... 56

## III Part III: Development and Evaluation

6 Virtual Model-View-Controller based Applications  
  6.1 Task Management Environment  
    6.1.1 Task Management Principles  
    6.1.2 Application overview and E-learning services  
      6.1.2.1 Formula Engine  
      6.1.2.2 Verification Engine  
      6.1.2.3 Formula Editor  
      6.1.2.4 Problem Editor  
      6.1.2.5 Task Editor  
      6.1.2.6 Task Management database  
    6.1.3 Usage scenario ....................................... 67  
  6.2 Wiki-Gloss Tool for Language Learning  
    6.2.1 Wikipedia Miner  
    6.2.2 Wikification process  
    6.2.3 Wiki-Miner Engine  
  6.3 WikiGloss Features and Design Issues ................................ 70
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.4 Usage Scenarios</td>
<td>70</td>
</tr>
<tr>
<td>6.5 Schema of Applications in the Virtual-MVC architecture</td>
<td>72</td>
</tr>
<tr>
<td>6.6 Other areas for implementing the Virtual-MVC design pattern</td>
<td>73</td>
</tr>
<tr>
<td>6.6.1 Visual Programming Services</td>
<td>73</td>
</tr>
<tr>
<td>6.6.1.1 Movie-Based Programming Environment (MMEPAD) architecture and reusability features</td>
<td>74</td>
</tr>
<tr>
<td>6.6.2 Distributed Tsunami Modelling Services</td>
<td>76</td>
</tr>
<tr>
<td>6.7 Chapter Summary</td>
<td>78</td>
</tr>
<tr>
<td>7 Analysis and Evaluation Studies</td>
<td>80</td>
</tr>
<tr>
<td>7.1 Evaluation Study</td>
<td>80</td>
</tr>
<tr>
<td>7.1.1 Analysis of Time perception</td>
<td>81</td>
</tr>
<tr>
<td>7.1.2 Analysis of development complexity perception</td>
<td>82</td>
</tr>
<tr>
<td>7.2 Degree of Service-Orientation</td>
<td>86</td>
</tr>
<tr>
<td>7.3 Qualitative comparison of MVC roles</td>
<td>88</td>
</tr>
<tr>
<td>7.4 Analysis of related MVC Design Patterns and programming approaches</td>
<td>90</td>
</tr>
<tr>
<td>7.5 MVC programming approaches</td>
<td>90</td>
</tr>
<tr>
<td>7.5.1 Comparison with Service Component Architecture</td>
<td>93</td>
</tr>
<tr>
<td>7.6 Chapter Summary</td>
<td>94</td>
</tr>
<tr>
<td>8 Summary and Conclusions</td>
<td>96</td>
</tr>
<tr>
<td>Appendix A</td>
<td>99</td>
</tr>
<tr>
<td>A.1 Abbreviations</td>
<td>99</td>
</tr>
<tr>
<td>A.2 Virtual-MVC Qualitative evaluation study. Survey to evaluate Devel-</td>
<td>102</td>
</tr>
<tr>
<td>oper’s perception of complexity</td>
<td></td>
</tr>
<tr>
<td>References</td>
<td>107</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Extended SOA conceptual Architecture [46]</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>N-tier SOA Enterprise Architecture [15]</td>
<td>10</td>
</tr>
<tr>
<td>2.3</td>
<td>Application integration styles</td>
<td>11</td>
</tr>
<tr>
<td>2.4</td>
<td>Point to point integration</td>
<td>16</td>
</tr>
<tr>
<td>2.5</td>
<td>SCA Assembly model. Example of a composite artifact [53]</td>
<td>19</td>
</tr>
<tr>
<td>3.1</td>
<td>Model View ViewModel Design pattern [34]</td>
<td>24</td>
</tr>
<tr>
<td>3.2</td>
<td>Model View Presenter Design pattern</td>
<td>25</td>
</tr>
<tr>
<td>4.1</td>
<td>Traditional MVC design pattern</td>
<td>31</td>
</tr>
<tr>
<td>4.2</td>
<td>Web based MVC architecture</td>
<td>32</td>
</tr>
<tr>
<td>4.3</td>
<td>Virtual MVC Design Pattern</td>
<td>33</td>
</tr>
<tr>
<td>4.4</td>
<td>Business logic layer conceptual schema</td>
<td>34</td>
</tr>
<tr>
<td>4.5</td>
<td>Virtual-MVC controller architecture</td>
<td>35</td>
</tr>
<tr>
<td>4.6</td>
<td>Component’s integration mapping</td>
<td>37</td>
</tr>
<tr>
<td>4.7</td>
<td>Dependency implementer of mapping scheme</td>
<td>38</td>
</tr>
<tr>
<td>4.8</td>
<td>Algorithm for DI implementation in controller’s Service Orchestration Engine</td>
<td>39</td>
</tr>
<tr>
<td>4.9</td>
<td>Dependency Injection stages for data transformation in Java</td>
<td>40</td>
</tr>
<tr>
<td>4.10</td>
<td>Virtual-MVC architecture</td>
<td>41</td>
</tr>
<tr>
<td>5.1</td>
<td>Virtual MVC application layout</td>
<td>45</td>
</tr>
<tr>
<td>5.2</td>
<td>Service endpoint design schema</td>
<td>48</td>
</tr>
<tr>
<td>5.3</td>
<td>Service endpoints deployment within Virtual-MVC framework</td>
<td>49</td>
</tr>
<tr>
<td>5.4</td>
<td>Metadata table with services registered for Problem Editor</td>
<td>51</td>
</tr>
<tr>
<td>5.5</td>
<td>Virtual-MVC framework, controller mapping definition</td>
<td>54</td>
</tr>
<tr>
<td>5.6</td>
<td>Virtual-MVC. Schema of software component’s interaction and data flow</td>
<td>55</td>
</tr>
<tr>
<td>6.1</td>
<td>Formula Engine</td>
<td>61</td>
</tr>
<tr>
<td>6.2</td>
<td>Formula Verification Strategy</td>
<td>62</td>
</tr>
<tr>
<td>6.3</td>
<td>Example of Problem Editor view</td>
<td>64</td>
</tr>
<tr>
<td>6.4</td>
<td>Example of Task Editor view</td>
<td>65</td>
</tr>
<tr>
<td>6.5</td>
<td>A Fragment of the Task Management Database Structure</td>
<td>66</td>
</tr>
<tr>
<td>6.6</td>
<td>Example of the WikiGloss functionality</td>
<td>71</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>6.7</td>
<td>Example of Article translation gloss</td>
<td>71</td>
</tr>
<tr>
<td>6.8</td>
<td>WikiGloss practice activities</td>
<td>72</td>
</tr>
<tr>
<td>6.9</td>
<td>Service Engines and applications in Virtual-MVC Architecture</td>
<td>73</td>
</tr>
<tr>
<td>6.10</td>
<td>Problem Editor contextual help using Wiki-miner services</td>
<td>74</td>
</tr>
<tr>
<td>6.11</td>
<td>MMEPAD Architecture</td>
<td>75</td>
</tr>
<tr>
<td>6.12</td>
<td>Virtual-MVC Architecture and Movie-based components</td>
<td>76</td>
</tr>
<tr>
<td>6.13</td>
<td>Movie-based Web based GUI in the view</td>
<td>77</td>
</tr>
<tr>
<td>7.1</td>
<td>Survey analysis, student’s self expertise rating</td>
<td>81</td>
</tr>
<tr>
<td>7.2</td>
<td>Student’s profiles and time evaluation</td>
<td>82</td>
</tr>
<tr>
<td>7.3</td>
<td>Development complexity evaluation</td>
<td>83</td>
</tr>
<tr>
<td>7.4</td>
<td>Developers perception of the Virtual-MVC methodology</td>
<td>85</td>
</tr>
<tr>
<td>7.5</td>
<td>Summary of developer’s evaluation of Virtual-MVC and programming strategy</td>
<td>87</td>
</tr>
</tbody>
</table>
List of Tables

2.1 CRUD activities vs. HTTP Commands ........................................ 14
3.1 Model-View-Controller based Design Patterns comparison ........... 28
5.1 Problem Editor atomic operations .............................................. 48
7.1 Extended Model-View-Controller based design patterns comparison . 89
7.2 Analysis of related Model-View-Controller programming approaches . 92
7.3 Programming effort analysis .................................................... 93
Chapter 1

Introduction

1.1 Overview

Service Oriented Architecture (SOA) approach introduced principles, strategies, and patterns for developing loosely coupled, standard-based software components [46]. Services are the basic building blocks to construct complex distributed applications. The increasing interest of enterprises to adopt SOA is due to its ability to accelerate the development of applications, respond better to a dynamic business environment, while decreasing the time of deployment and the total cost of ownership, as well as solving applications and Business to Business (B2B) integration issues [23]. However, the realization of a service-based flexible infrastructure and development environment is a complex, time consuming, and costly task. To achieve the benefits of SOA, it is necessary to establish a Governance process, enforcing SOA principles and standards along a wide enterprise environment. A diverse set of tools and strategies are required to construct SOA solutions such as Middleware, BPEL4WS, WS-* standards, Service registries, Design Patterns etc., where the interaction of these tools introduce certain complexity during initial stages.

Traditionally, companies have used two main approaches to move their enterprise infrastructures to SOA [41] [57]. The first one is referred as “Top-Down”SOA, which usually consists of an organization-wide initiative to re-architect and integrate disconnected systems, and design new applications according to SOA principles. However, it requires a high initial investment to cover big vendors and proprietary SOA tool stacks that might be comprised of multiple products such as application servers, enterprise services buses, orchestration engines, management and development tools, hence the initial roll out can span long time. To deal with the diverse tools required for implementation, it is necessary that the human resource involved such IT architects and programmers, acquire specialized skills. Hence, additional investment in training should be added. Previous knowledge, tools and processes are not fully reutilized. These difficulties can lead to the failure of many of the initiatives based on the Top-Down approach [41].

The second approach, and that has proven more success is called “Bottom-Up ”SOA. It emphasizes incremental adoption and usually involves a standalone ESB or integration Platform-as-a-service (PaaS) to enable the creation and orchestration of services without the need of an application server or other infrastructure components. Hence,
reducing implementation complexity, the upfront costs of roll out period, and the costs of special training to learn new technologies and programming environments. Standalone Enterprise Service Bus (ESB) do not impose vendor lock or architectural choice, providing the organizations with flexibility to integrate with a wider range of systems, applications, and Cloud services [41].

There are some parameters indicate the relative complexity of implementation which are: number of products required for implementation, effort required for to setup tasks and development, time needed for a given operation, and the specialization required by the developers. Additionally, a qualitative study of the relative implementation complexity between Oracle and IBM SOA products [23] proposed four main evaluation parameters:

- Number of products involved and the degree of integration.
- Number of steps involved to accomplish an operation.
- Time needed to accomplish an operation.
- Ranking of task complexity and specialization required.

Both solutions are actually robust in supporting SOA. However, the practical results demonstrated that IBM approach is more complex due to the amount of products involved in the implementation, and the degree of integration between them is a challenge. The deployment of each of these products is done separately, the time to setup and configure each product, and the specialization level to accomplish tasks is higher. Another example of SOA implementation, uses the open source MuleSoft ESB [41], which is also referred to as a solution to simplify SOA realization. To implement this solution the developers need to have only one additional product. Moreover, it is not necessary to provide developers a special training, or hiring consultants [42]. This can pose a considerable impact in reducing upfront costs and the time of implementation.

The implementation of SOA brings higher benefits for enterprise architectures dealing with complex systems interactions between different local and external domains, under changing environments, and where multiple teams require sharing business services [57]. We believe that to enable the benefits of Cloud and Big Data technologies, it is necessary that SOA is not only adopted by big size enterprises, but also becomes the development model of the next generation of applications irrespective of company size. A potential scenario is the university research environment, where there might be small or medium-sized software teams working on different projects. However, the rotation of students is higher than within a company environment, and the need to change platforms according to research requirements and students skills is common. A more efficient approach is necessary to support projects evolution, extension and reusability of software components. The environment should support interoperability of the components, facilitate integration tasks, and adopts standard technologies reducing the training time.

Adopting SOA for medium or small sized software initiatives such as in a university research environment can help to overcome the aforementioned constraints. According
with the approaches discussed, the main challenges of SOA adoption for small sized software projects are the complexity of implementation, upfront costs, roll out time, and adherence to SOA principles. Hence, solutions that do not require a specialized training, and a big set of proprietary stack of tools to simplify the work, are more likely to be the preferred choice and will enable a more widespread SOA adoption (MuleSoft [41]).

The adoption of Cloud computing model such as the provision of Software-as-a-Service (SaaS), discussed in [39], is based on the premise towards a more cost-effective solution where the customer should pay only for the consumed services. E-Learning applications are evolving from restricted monolithic designs to platforms embracing Service-Oriented concepts to be deployed in Cloud environments. Traditional Learning Management Systems (LMS) offer a set of basic or general functionalities, trying to cover a wide range of needs. However, there might be functionality that the system does not have. It is necessary then to obtain this functionality by adding components from external vendors, or developing them in house. Developing e-Learning applications based on loosely coupled services facilitates reusability, and integration with heterogeneous applications. The next generation of e-learning platforms should allow the dynamic discovery and assembling of e-learning services [13]. Therefore, the SaaS model is in line with the next generation of e-Learning platforms. The implementation aspect on e-Learning SOA frameworks is still an open area, and it requires more effort to be adopted and moved into the new generation of e-Learning tools.

The purpose of this research is to reduce the development complexity of SOA based applications, by proposing the Virtual-MVC design pattern. The presented pattern extends the concepts of the traditional MVC focusing on reusability, and removing the hurdles of additional integration and deployment code from the programmer. The integration task is abstracted in a configurable controller. The pattern programming approach is evaluated in a software engineering class, obtaining positive results reducing development complexity. The research in this work aims to contribute to the implementation of e-Learning SOA platforms within the framework of an e-Learning Computational Cloud.

1.2 Thesis contribution

A well constructed SOA can empower a business environment with a flexible infrastructure to provision business processes as independent and reusable sets of services. It should rely on service-oriented engineering methodologies that help enterprises to effectively design and deploy services. Older software development paradigms for object-oriented and component-based development cannot be applied without considering the appropriate adaptation to the SOA key elements including services, flows of information (services composition), and components realizing services [47]. These methodologies just partially addressed the key issues. One of the challenges is the application of sound design principles for engineering service applications, which should guarantee that services are self-contained, have clearly defined boundaries and service endpoints
for better services composability and loose coupling. Development environments and architectures that enforce SOA design principles can have a positive impact decreasing the complexity of implementation and further flexibility of a service based environment. The presented approach focuses on the architectural approach to decrease complexity of implementation, by proposing a so called Virtual-MVC Design Pattern based on the integration of MVC, ESB and Dependency Injection to simplify the development process of SOA applications. The Design pattern is realized into a development framework that does not requires extensive training for developers, and based on integrated SOA functionality in a form of an enhanced controller. The research work addresses the following issues:

- **Extending MVC design pattern for Service-Oriented Architecture.** The Model-View-Controller (MVC) is a well-known and widely used design pattern. It fosters reusability by breaking down an application’s functionality into three components, separating the presentation layer from the domain model. However, while keeping a reference to the model, the view is not completely decoupled compromising the model’s privacy.

  MVC is originally an object-oriented design pattern, and it doesn’t addresses SOA key elements such as services, compositions, and components related with their realization. Therefore, it cannot be directly applied to a service-oriented paradigm. The proposed pattern is named Virtual Model-View-Controller (Virtual-MVC). It retains the main concepts of reusability and loose coupling of MVC components, but extending the pattern for SOA. Virtual-MVC decouples the view from the model and redirects the Model-View link within an enhanced controller as a virtual layer for distributed and service-oriented applications. Such decoupling facilitates task’s distribution and developer’s collaboration.

- **Decoupling view from model by virtualizing its dependency through and enhanced controller.** The controller is re-designed as an Enterprise Service Bus (ESB) to foster sophisticated interconnectivity between services. It comprises the integration logic and performs functions related with data and protocol transformation, message exchange and intelligent routing. Service virtualization is realized by incorporating the Dependency Injection (DI) pattern, which is a better alternative for decoupling than Service Locator Pattern, where every client has a dependency on the registry class of the Service Locator [20].

  The controller forms a connectivity channel to process service’s engines that choreograph the flow of activities between loosely coupled services. By introducing DI, the managed objects are decoupled from any hard-coded dependencies on its collaborators including expensive initialization logic and allows to test objects easily in isolation. This protects the privacy of the business logic from remote reference access as well as reduces the complexity of deploying and assembling SOA software components.
Facilitating the design of loosely coupled Virtual-MVC based components by realizing a development framework and programming approach. To implement a Design pattern it is necessary to adopt a programming strategy, but in some cases it can lead to a complex scenario if the available platforms don’t have the appropriate features [3] [51].

In our approach, the pattern’s proof of concept is formalized through a development framework that follows Virtual-MVC. Hence, the programming approach is designed to minimize the hurdles of developers. The framework is based on standard technologies to avoid long learning efforts. The framework and design pattern have been introduced in Software Engineering course at the University of Aizu, to analyze the developer’s experience implementing SOA applications. By following Virtual-MVC, the applications are based on SOA principles from the initial steps of design, enforced by the pattern.

Implementing Virtual-MVC framework for an E-Learning Computational Cloud environment. A set of applications have been developed using the Virtual-MVC programming approach. Those applications are exposed as e-learning services using an instance of the Virtual-MVC framework. The instance of the environment is called as E-learning computational cloud. Two applications for E-Learning arena are discussed, demonstrating the reusability of services components. The first one is a glossing tool for language learning activities called Wiki-Gloss tool, and a Task Management environment that contains a group of e-Learning components, that can be used as stand-alone applications, or the atomic services can be integrated into a Learning Management System (LMS) such as Moodle. Although the focus of the applications are on E-Learning arena, The Virtual-MVC framework is not restricted to this area. Another example of Visual Programming area, deals with the integration and extension of stand-alone applications. The stand-alone version of the Visual Programming software is extended by consuming services from the Virtual-MVC platform.

1.3 Chapters overview

The present dissertation is organized in the following chapters.

Chapter 2. An overview of SOA complexity issues and typical architectures is presented, discussing how complexity impacts the development process, and initial implementation. SOA based approaches to decrease complexity are studied, as well as related metrics to measure complexity.

Chapter 3. Traditional Model-View-Controller pattern, concepts, and related adoption are discussed in this chapter. The relevant variations of the Model-View-Controller design pattern are studied, and identified the benefits and limitations related with current development trends in SOA.
1. INTRODUCTION

- **Chapter 4.** The proposed Virtual Model-View-Controller and the foundation concepts of the approach are discussed in this chapter. To reduce SOA implementation complexity, we adapted the Model-View-Controller to SOA. The decoupling strategy, and the construction of a development framework following the pattern is explained.

- **Chapter 5.** The focus of this chapter is on the development framework and the programming strategy following Virtual Model-View-Controller pattern. A case scenario is discussed to demonstrate the steps to realize services, and the assembly SOA based applications.

- **Chapter 6.** This chapter explains a set of E-learning based applications realized using Virtual Model-View-Controller approach. The demonstration explains how components are organized within the architecture, and the advantages on reusability of E-Learning services that can be exposed on an E-learning Cloud environment. Additional areas of application such as Visual Programming, explain the integration with stand-alone applications.

- **Chapter 7.** The evaluation studies to validate the Virtual Model-View-Controller design pattern and results are discussed in this chapter. The evaluations carried out comprise qualitative and experimental based analysis in a course based programming scenario. As well as a comparative analysis measuring development efforts in related programming approaches.

- **Chapter 8.** A summary of the research work, results, and main contributions are presented in the Conclusions chapter.
Part I

Part I: Related Work and Background
Chapter 2

Service-Oriented-Architecture Implementation and Complexity Issues

Overview. The initial implementation of a Service Oriented Architecture (SOA) can be a daunting task. Several layers of functionality are involved, therefore huge efforts, resources and time are required. This chapter presents an overview of software complexity issues from the architectural point of view, and how those affect the development process. Initial approaches of integration such as point-to-point restricted flexibility of the applications while introduced complexity overloading the integration code within the service. Approaches such as the Enterprise Service Bus, and service virtualization helps to reduce the integration logic encoded in the service. Mature SOA implementations are build with a combination of technologies, Service Component Architecture aims to reduce the complexity related of managing different tools and paradigms by providing an standardized framework for assembling disparate services into higher composites. Finally, the chapter discusses metrics to understand the complexity involved in SOA solutions.

2.1 Service-Oriented Architecture

A well constructed SOA can empower a business environment with a flexible infrastructure to provision business processes as independent and reusable sets of services. An infrastructure supporting SOA can be composed in several layers depending on the business needs. Two main architectural approaches are reviewed, to help understanding of the SOA functionality that is required to support the enterprise solutions, and the organization of such functionality in logical layers.

Extended Service-Oriented Architecture (xSOA) is an stratified service-based architecture proposed by [46]. The conceptual architecture is organized in three main layers that build upon each other as a pyramid conceptual structure (see Figure 2.1). The bottom layer is comprised by the Basic services that define the basic SOA, handling concepts such as service capability, interface, behaviour, QoS, publication, discovery, selection and binding. The second layer builds on top of the basic concepts, and it’s named as Composite services. It encompasses roles and functionality for the aggregation of multiple services into compositions. Therefore, coordination, conformance,
monitoring, and semantics are part of this layer. The third and top most layer is Managed services, that incorporate the concepts to manage SOA based applications through their life cycle. The most prominent functions are: Service Level Agreement (SLA) management, Auditing, monitoring and troubleshooting services; Dynamic service provisioning, Service life cycle/state management, Scalability/extensibility, Service operations management, and Service market management.

The second example is a SOA architecture proposed by IBM [15], it is a more concrete approach compared with xSOA. However, it is congruent with the functionality and concepts discussed in the previous example. The architecture is composed by nine layers (Figure 2.2). The top and first layer Consumers, consist of users interfaces, external service consumers, and Business to Business (B2B) integration. In this layer is important the selection and adoption of appropriate standards, roles identification, and customization depending on the client devices. The next two layers Business process and Services, are related with the workflow design and implementation for composition, choreography, business state machine, and orchestration, as well as the services that are selected to participate. The fourth layer is Service Components, comprised by the enterprise components responsible of realizing functionality, maintaining the Quality of Services (QoS), and ensuring conformance to SLA. This layer typically contains technologies such as application servers. The fifth layer Application Services, consist of the legacy systems to be integrated. The sixth layer Data Repositories and Information
2. SERVICE-ORIENTED-ARCHITECTURE IMPLEMENTATION AND COMPLEXITY ISSUES

Services is comprised by elements such as databases, data warehouses, data providers, information services associated with data management, master data management, data analytic services and technologies.

The last three layers are considered as cross-cutting aspects of all the previous functionality in the architecture. The seventh is the Integration layer, enabling the integration of other services introducing a reliable set of capabilities, such as intelligent routing, and transformation mechanisms (data/protocol), where the implementation is usually through the ESB. The eight layer is Quality of Service (QoS). It encompass the mechanisms and services for security, management and monitoring. The last and ninth layer is Governance, representing the people, process and procedures required to maintain trust and control of the information services. The efforts related for implementing such architecture are not to be taken lightly, it usually requires a set of tools, and platforms that support all the requirements of the architecture. It is necessary to develop the skills and coordinate among developers, stakeholders, and company employees involved. The initial implementation of a SOA enterprise architecture can be a complex task. As shown in the previous examples several layers of functionality are involved, and necessitate to assemble a set of different tools to support the architecture. Huge efforts, resources and time are usually required to gain the benefits of interoperability, reusability, and high degree of integration promised by SOA. The decision of selecting an appropriate platform, should take in consideration its ability to simplify the fundamental tasks of development, deployment, and management [23].
2.2 Integration of applications and Service-Oriented Architecture

A service is a unit of solution logic comprised with a set of functions and capabilities [18], that are exposed through a network accessible endpoint. Service consumer and provider use messages to exchange invocation request and response information in the form of self-containing documents that make few assumptions about the technological capabilities of the receiver [45].

In SOA, the purpose of a service is to represent a reusable unit of business complete work. The exposed service functionality has three main properties: it is self-contained, platform independent, and dynamically located, invoked, and combined [46]. The implementation and execution of the application providing the desired functionality is encapsulated behind the service interface which is invoked, and thus it is irrelevant whether services are local or remote, the protocol for invocation, or the infrastructure components required to establish the connection.

The most common architectural approaches to integrate applications suggested by [48], rely on shared database, file transfer, remote procedure calls, or exchange asynchronous messages such as enterprise service bus. This work focuses on the SOA related choices, which are remote procedure call (RPC), and Enterprise Service Bus (ESB). Figure 2.3 is adapted from [48]. The integration style influences the underlying platforms, as well as the technology options, which can be summarized as follows.

- Build the client module to match the characteristics of every server module that it will invoke. This approach requires to develop an interface for each connection, thus resulting in a point-to-point topology. Point-to-point integration is not scalable and very complex as the number of integration points increases, it becomes harder to maintain due to the tight coupling.

![Figure 2.3: Application integration styles](image)
2. SERVICE-ORIENTED-ARCHITECTURE IMPLEMENTATION AND COMPLEXITY ISSUES

- Insert a layer of communication and integration logic between the client and server modules. It must support interoperability and coexist with deployed infrastructure and applications. Such layer is usually implemented in SOA in the form of an ESB.

There are two main technologies to implement Web services, as shown in Figure 2.3. Web services based on SOAP and XML message communication, WSDL interfaces, and based on a set of Web services standards (WS-*) have been the baseline for SOA implementation. A second main technology option that is gaining relevance to reduce the perceived complexity of WS-* is Representational State Transfer (REST) web services. In the later subsections both technologies will be revised in more detail, as well as the main integration approaches.

2.2.1 Big Web Services

The term Big Web Services refers to Web services that use XML messages following the Simple Object Access Protocol (SOAP) standard, an XML Language defining a message architecture and message format. Such systems, often contain a machine readable description of the operations offered by the service, written in the Web Services Descriptions Language (WSDL) [45]. Hence, the core Web Services definition and interface standards are: XML, WSDL, XML Schema Definition Language (XSD), and SOAP. Additional to these core standards, concepts such as reliability, security, management, and communication protocols necessary to achieve greater levels of interoperability are as well introduced in an extended standards stack known as Web Service specifications (WS-*) [5]. The WS-* technology stack covers also many others QoS features required to ensure the interoperability of advanced middleware systems, but given the modularity and composability required for the approach, it has led to a fairly large set of WS-* specifications [48].

In the context of a Web Service, XML is the coding scheme as well as the structural representation of a message instance. It is platform agnostic allowing service consumers and providers to communicate, regardless of the technology on which they are developed. WSDL describes the service interface, including the name and location of the service, the operation exposed by the service, and the expected inputs and outputs defined by the interface. The XML Schema is used to define and constrain the content of the XML documents. The message is derived from the service interface definition of the XML schema, which is embedded within the WSDL interface. SOAP protocol and message framework is used to define the message encapsulated in XML tags. It defines a top level element called envelop containing a header and body. The SOAP header is an extensible container for message layer infrastructure that contains descriptive addressing and delivery information. It can be used for routing proposes and QoS configuration (such as transactions, security, reliability, etc.). The body contains the encoded message and data (payload).

Big Web Services are a very strong technological option to gain the interoperability and flexibility needed by enterprise applications. However, it as well has some limita-
tion. We discuss as summary of strength and limitations that have been identified by different authors [48, 5].

**Advantages**

- Using SOAP, the same message in the same format can be transported across a variety of middleware systems, which may rely on HTTP or other transports.

- Protocol transparency and independence can be achieved following WS-*.

- The service interface described using WSDL helps to abstract from the underlying communication protocol and serialization details, as well as the service implementation platform.

- SOAP and the WSDL interface definition language are widely adopted as gateway technologies to deliver interoperability between heterogeneous middleware systems.

- SOAP engines and WSDL tools, can help to hide the perceived complexity from the application programmer.

**Limitations**

- SOAP is deemed to be complex rather than a simple approach, and a considerable learning curve can be necessary for new programmers.

- Interoperability problems can occur when, for instance, native data types and language constructs of the service implementation are present in its interface. However, it can be mitigated by design and coding guidelines such as contract-first development.

- The ability to reuse schemas and schema content by reference and inheritance are powerful. However, the cross-references between schemas if are not carefully designed can lead into a highly complex hierarchy of schemas.

- Similarly, another complexity of schema reuse is namespace inheritance. Complex inheritance and class structures can present a problem for some object-oriented applications.

- In early implementations, interoperability problems arise due to the expressiveness of the WS-* stack of standards. WS-I helped to clarify misinterpretations. Difficulties on the translations between XML and the corresponding memory data structures are still the main source of performance inefficiencies. For example, JAVA several attempts to produce a stable Web service marshalling layer (Apache SOAP, JAX-RPC, JAX-WS, and JAX-B)

- XML Schema is a very rich language, making difficult to identify the right construct to express a data model in a way that is fully supported by all SOAP/WSDL implementations.
2. SERVICE-ORIENTED-ARCHITECTURE IMPLEMENTATION AND COMPLEXITY ISSUES

Table 2.1: CRUD activities vs. HTTP Commands

<table>
<thead>
<tr>
<th>CRUD Activities</th>
<th>HTTP Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>POST</td>
</tr>
<tr>
<td>Read</td>
<td>GET</td>
</tr>
<tr>
<td>Update</td>
<td>PUT</td>
</tr>
<tr>
<td>Delete</td>
<td>DELETE</td>
</tr>
</tbody>
</table>

2.2.2 REST Web Services

Representational State Transfer (REST) style services mimic the basic operations of the World Wide Web and relies mainly on HTTP protocol. While Big services focus on transactions, REST services focus on resources. It relies heavily upon the notion of an identified resource, the resource type, and the core HTTP command set. Such HTTP commands are similar to the Create, Read, Update, and Delete activities known as CRUD [5]. According to [48] REST architectural style is based on four principles:

- **Resource identification through URI.** A RESTful Web service exposes a set of resources that identify the targets of the interaction with its clients. Resources are identified by URIs that provide a global addressing space for resource and service discovery.

- **Uniform interface.** Resources are manipulated using a fixed set of four operations: PUT, GET, POST, DELETE. Table 2.1 shows the correlation with CRUD, and the operation usage. The resources share a uniform interface to transfer the state between the client and server.

- **Self-descriptive messages.** Resources are decoupled from their representation, hence their content can be accessed in a variety of formats such as HTML, XML, plain text, PDF, JPEG, etc. Metadata about the resource is available, used for example to control caching, detect transmission errors, negotiate the appropriate representation format, and perform authentication or access control.

- **Stateful interactions through hyperlinks.** Every interaction with a resource is stateless. For example, the request messages are self-contained. The resources contain the application’s state and functionality, which is represented by an unique URL. Several techniques exist to exchange state, such as URI rewriting, cookies, and hidden form fields. The state can be embedded in response messages to point or validate future states of the interaction.

RESTful Web services are considered an important architectural option, due to the simplicity of deployment compared with Big web services. The main advantages and
limitations are discussed as follows.

**Advantages**

- Low barrier of adoption, and inexpensive tooling. RESTful web services use existing well-known standards (HTTP, XML, URI, MIME), and have a lightweight infrastructure \[45\], because they don’t use SOAP or any other application layer protocol except for HTTP \[6\].

- The effort required to build a client is very small as developers can begin testing such services from an ordinary Web browser, without having to develop custom client-side software. Deploying RESTful web services is very similar to building a dynamic Website.

- Web resources can be discovered without an approach based on centralized repository due to the usage of URI and hyperlinks.

- Stateless services can be scaled to serve a large number of clients thanks to the support for catching, clustering and load balancing built into REST.

- REST is particularly useful for limited-profile devices, such as mobile phones, for which the overhead of headers and additional layers of SOAP elements on the XML payload must be restricted.

**Limitations**

- Proxies and Firewalls may not always allow HTTP connections that use operations outside POST and GET (for idempotent requests). As well as XHTML only allows both methods. These limitations lead to a series of workarounds, such as using `X-HTTP-Method-Override` special header in HTTP, or using a hidden form field `_method` when coding with Ruby on Rails.

- Difficulties following strictly GET vs POST rule. For idempotent requests having large amounts of input data (usually more than 4KB) it is not possible to encode it in the resource URI, as it will be rejected as malformed URI, or expose the service to buffer overflow attacks.

- It may also be challenging to encode complex data structures into an URI, since there is no commonly accepted marshalling mechanism. However, POST method does not suffer from such limitations.

### 2.2.3 Point-Point integration

Web services are important means to integrate disparate applications. They wrap the low level computational logic by using a high level service interface. The service interface ascribes the behaviour of the underlining service logic. Therefore, in order to invoke a service, the software component or client should know the high level details of the service interface. There could be more than one Web service, and there could
be more than one client. Hence, every client has to understand and remember the individual web service interfaces. Such point-to-point integration does not have high level of flexibility (Figure 2.4) because it requires for clients to accumulate knowledge about an individual service. This leads that clients and services are not completely decoupled. Client has to formulate the workflow in order to execute the complicated problems that require more than on web services to be executed. Furthermore, the integration logic and implementation logic of a service are tightly coupled in a single web service. That is why non-functional logic such as monitoring, security, session control, message routing, service composition is overloaded in the service.

### 2.2.4 Middleware and Enterprise Service Bus

The Enterprise Service Bus (ESB) is an open standards based message bus designed to enable the implementation, deployment, and management of SOA based solutions with the focus on assembling, deploying, and managing distributed SOA [46]. The ESB is defined as an standards based integration platform that combines messaging, web services, data transformations, and intelligent routing to reliably connect and coordinate the interaction of significant numbers of diverse applications across extended enterprises with transactional integrity [9]. Conceptually, the ESB has evolved from the store-and-forward mechanism found in middleware products, e.g., Message Oriented Middleware, and combines conventional EAI technologies with Web services, XSLT, orchestration, and choreography technologies, e.g., BPEL, WS-CDL, and ebXML. Physically, an ESB provides an implementation backbone for an SOA.

Endpoints in the ESB, provide abstraction of physical destination and connection
information transcending plumbing level integration capabilities of traditional, tightly coupled, and distributed software components. Thus, it is possible to set up at runtime its location properties. Endpoints allow services to communicate using logical connection names, which an ESB will map to actual physical network destinations at runtime.

Thus, it is possible to set up at runtime its location properties. Endpoints allow services to communicate using logical connection names, which an ESB will map to actual physical network destinations at runtime. This destination independence offers the services connected to the ESB, the ability to be upgraded, moved, or replaced without having to modify code and disrupt existing ESB applications. The advantage of the connection abstractions is that it facilitates to duplicate processes along different physical locations to handle fail over if a service is not available. The endpoints can be configured to use several levels of QoS, which guarantee communication despite network failures and outages.

The event driven SOA provides a means of abstracting away from the details of the underlying service connectivity and protocol. An event source typically sends messages through some middleware integration mechanism like the ESB, and then the middleware publishes the messages to the services that have subscribed to the events. The event itself encapsulates an activity, constituting a complete description of a specific action. In a brokered SOA, the only dependency between the provider and client (service) is the service contract (described in WSDL). To achieve more lightweight arrangement, an event-driven SOA requires that two participants in an event (server and client) be fully decoupled, thus the two participants in an event do not need to have any knowledge about each other before the transaction.

Ideally, a sophisticated graphical business process management tool can be used to configure, deploy, and manage services and endpoints [46]. This allows flexible manipulation and reconfiguration of services without requiring re-writing or modifying the services themselves. The most important capabilities of the ESB to support SOA implementation are service routing and substitution. Additional capabilities are: leverage of existing assets (legacy application integration), service communication capabilities (protocol transformation), Dynamic connectivity capabilities (connectivity code should be the same regardless of the service implementation, no need of separate API or proxy), content based routing capabilities, Endpoint discovery with multiple quality of service capabilities, Integration capabilities (not only service-style interactions but legacy systems, COTS components, etc), Transformation capabilities, Reliable messaging capabilities, Security capabilities, Long running process and transaction capabilities, Management and monitoring capabilities, and Scalability capabilities. The architectural integration style of the ESB is technology agnostic, meaning that it can reuse functionality in existing applications into development of new ones.

### 2.2.5 Service Virtualization

The introduction of tools and techniques associated with n-tier environments such as SOA has helped to speed up the development and lower IT cost for enterprises. However, this evolutions have introduced complexity in the development of applications for such environments. It is common that developers today must add additional layers of code to core application logic, just to deploy their applications within a service environment [10], where the authors suggest that about 40% of the efforts are spent
managing complexity.

Establishing service polices that can be enforced across different technologies, instead of coding them into the services, is a strategy for deploying, monitoring, and managing multiple interdependent services. Replacing code with rules-based policies that can be configured at runtime independently of the core service, adding flexibility and improving reuse [10]. *Service virtualization* is a strategy that helps to address such design-time and run-time challenges and helps the developer and administrators focus back on building the service logic. The concept of abstraction is well known in Object Oriented Programming, and it is used to promote code reusability, and manage software complexity by hiding the underlying implementation details, and providing an interface to access such functionality. Similarly, *Service virtualization* is an abstraction layer that separates the service invocation from the request by a consumer [10]. It provides two main benefits. The first one is code reduction focusing on implementing a class containing the business logic, and deploy it as *virtualized service* in a container or framework that makes deployment and other attributes configurable. The second benefit is greater reuse and flexibility. Service bindings, security, policy, and other attributes can be reconfigured and changed as needed while reusing the service, instead of having to change the service code. Ideally this functionality should be supported and facilitated by the infrastructure.

### 2.2.6 Service Component Architecture

Mature SOA implementations usually are built with combinations of technologies such as Java connector Architecture (JCA), Business Process Execution Language (BPEL), and Web services. These technologies have standardized some fundamental aspects of application development. However, as the number of technologies increases, as well it does the number of separate tools. Similarly, the specialized knowledge required to build, deploy, and manage such tools and technologies. The complexity introduced by these separate tools, different metadata, and deployment paradigms is a challenge. The aim of Service Component Architecture (SCA) is to reduce IT complexity through a standardized framework for assembling disparate enterprise SOA components into a higher-level composite [53].

SCA provides a programming model for building applications and solutions based in SOA. The main approach is to model a business function providing a series of services, which can be assembled together to solve a particular business need [8]. The application can be composed from new services, and existing systems and applications, therefore supporting re-usability.

The SCA Assembly model consists of a set of artifacts defining the configuration of an SCA domain in terms of composites. A *Composite* contains assemblies of service components, the connection, and related artifacts describing how they are linked together. The SCA artifacts are represented in Figure 2.5. According with [53], the artifacts can be summarized as follows:

- **Composite**: Contains assemblies of service components.
• **Service.** Is considered the entry point into the composite. It contains the business function exposed to other components.

• **Component.** It is the unit of construction for SCA, and it provides the logic to be used within the composite.

• **Reference.** Are dependencies to other internal or external services in the composite.

• **Wire.** Connect services, components and references.

• **Properties.** Data values used for customization of the component behaviour in a particular deployment.

The component configures the implementation by providing values for the properties and by wiring the references to services provided by other components. SCA allows a variety of implementation technologies, including traditional programming languages such as Java, C++, and BPEL, but also scripting languages such as PHP and JavaScript and declarative languages such as XQuery and SQL.

### 2.3 Evaluating SOA Complexity

Software complexity refers to the extent of the difficulty to comprehend, modify and test a system [4]. Software complexity impacts development and maintenance efforts, which
are translated in higher costs for a company. Software complexity for the programmer interacting with a piece of software, is related with the difficulty performing tasks such as coding, debugging, testing, or modifying the software [27]. Software complexity measures are usually based on program code, and consider size of the program, control structure, and the nature of module interfaces.

The most widely known measures are Halstead measures and McCabe cyclomatic number. Halstead defined software complexity based on functions of the number of operators and operands in the program [24]. The metrics to calculate the volume, difficulty, and efforts to implement a program. McCabe considers the program as a directed graph where the edges represent line of control flow, and the nodes are straight line segments of code [32]. The cyclomatic number represents the number of linearly independent execution paths through the program. Both complexity measures consider a program or procedure as a single body of code, many other metrics have been developed over the time for procedural programming and object oriented.

Evaluating the complexity of Service-oriented application is important to estimate efforts, reduce costs, and keep a health status of SOA solutions. Structural complexity and autonomy, are two important quality attributes in SOA that have a great impact on other service attributes. Structural complexity impacts re-usability, performance, and maintainability. While autonomy impacts discoverability, re-usability, and statelessness. At design level, structural complexity of a service could be defined based on its interface. The simpler the interface, the less complex the service is. Hence, the structural complexity of a service could be defined based on its operations and messages [52].

An evaluation based on architectural considerations is suggested by [19]. A set of metrics is proposed The metrics are aggregated in three main SOA indexes.

- **SOA complexity index (SCI)**. Measures the inherent complexity of the SOA solution including its security, management and SOA governance measures, all of which offer significant benefits but also increase the complexity of the overall SOA.

- **Service Complexity Index (SVCI)** Measures complexity, but focus at the individual complexities of each of the services.

- **Flexibility and Agility Index (FAI)** Tracks the flexibility and agility of SOA solutions.

The proposed metrics evaluate the flexibility and complexity of a SOA implementation. The definitions indicate that improving flexibility, usually leads to an increase of complexity, due to additional layers for service virtualization, and service composition. It requires additional management and monitoring steps to ensure services and their associated components are functioning properly. This complexity is related to the implementation approaches and available technologies. However, when complexity is high and flexibility is low it is an indicator of a poor design.

Related implementation studies, proposes several parameters that can be considered to measure the relative complexity of implementation such as the number of products
required, the efforts required to set up tasks and development, the time needed for a given operation, and the developer’s expertise required. Additionally, a qualitative study of the relative implementation complexity between Oracle and IBM SOA products [23] proposed four main evaluation parameters: 1) the number of products involved and the degree of integration, 2) the number of steps involved to accomplish an operation, 3) the time needed to accomplish an operation, and 4) ranking of task complexity and skill level required. Both solutions are actually robust in supporting SOA. However, the practical results demonstrate that IBM SOA solution is more complex because the amount of products required for a SOA implementation is more than Oracle and the degree of integration between them is less. The deployment of each of these products is done separately, the time to set up and configure each product is higher, and the skill level required to accomplish tasks using the products needs to be more specialized, and sometimes this means hiring consultants.

Another example is the open source MuleSoft ESB, which is also referred to as a solution to simplify SOA realization. To implement this solution, the developers need to have only one additional product in comparison with the four products required by the other vendors. Moreover, it is not necessary to provide a special training of developers which permits them to apply their knowledge and skill much faster [41] [42]. This can pose a considerable impact in reducing upfront costs and the time of implementation.

2.4 Chapter Summary

The architectures discussed cover the different layers and functionality involved. The implementation of SOA platforms require the use of diverse tools, technologies and diverse strategies to enable a flexible architecture. However, the development steps, and interaction among them add a toll on the complexity of the architecture. There are two main styles to implement Web services, and their preference depend on the architectural needs of the particular business. Big Web services are suggested for professional enterprise application integration scenarios with a longer lifespan, and advanced QoS requirements. While RESTful services are recommended for ad-hoc integration over the Web, such is the case for Mashups. RESTful services are gaining importance for SOA implementation for their low cost of development, and their best use of well-know standards. Hence, an important consideration to reduce complexity and efforts, is that developers are more productive because they are adding to something they are already familiar with, rather than having to start from scratch with a new technology.

The introduction of the Enterprise Service Bus as a middleware layer is one the approaches to reduce the complexity of integration, and has become one of the backbones of SOA. While the preferred Web services style connecting the enterprise is Big Web services, using the ESB is possible to combine both SOAP based and RESTful web services.

Service virtualization is an abstraction layer that separates the service invocation from the consumer’s request. The main benefits of service virtualization are code reduction focusing on the business logic, and greater reuse and flexibility, because service
attributes related with integration can be reconfigured instead of changing the service code. Ideally this functionality should be facilitated by the infrastructure. Therefore, development platforms that enforce SOA design principles can have a positive impact decreasing the implementation complexity. The decision of selecting an appropriate SOA platform, should take in consideration its ability to simplify the fundamental tasks of development, deployment, and management.

Evaluating the complexity of Service-oriented applications is important to estimate development efforts, reduce costs, and keep a healthy SOA. Research on SOA metrics is still in progress, however there are many metrics proposed. The focus is identifying the factors that influence SOA complexity. Structural complexity impacts re-usability, performance and maintainability. It can be measure at design level based on the messages and operations described in the interface. Metrics related with SOA solution in a higher level, including elements such as security, management, and governance. An increase of flexibility usually leads an increases in complexity. However, when the complexity is high and flexibility is low, it is an indicator of poor design.
Chapter 3

Model-View-Controller and related Design Patterns

Overview. Design patterns are a strategy to re-utilize the knowledge of software artifacts. A pattern encodes the strategic knowledge to resolve a particular problem, and provides an standard language to share this knowledge among developers. In this chapter we discuss the most relevant MVC related patterns, analyze the roles of the components and their limitations to be adopted for current development in SOA.

3.1 MVC and Related Design Patterns

A Design Pattern encodes the strategic knowledge to resolve a particular problem. It is the basis of a common language and standard to communicate between developers beyond the formal structures entangled within the source code [61]. The essence of the MVC pattern is to demarcate the presentation logic (view), from the processing logic (model) by introducing an interface integration layer (controller) to handle user interactions, as well as increasing reusability of the software components [29]. The view and controller need to keep a reference to its model, if there is any change it requires an update of its correspondent view and controller. Although it increases the component’s re-usability, it can be complex to maintain, requires extra steps for testing individual components, and need to keep the reference from the view to the model compromising its privacy. Although MVC pattern has proven enhancing reusability, it introduced debugging complexity for simple interactions, and difficulties handling the view logic necessary for manipulate the presentation of data at the view end [21]. To resolve such difficulties, three main variations of the pattern were proposed to decoupled view and model, which are widely used to develop web applications. The patterns are described as follow.

3.1.1 Presentation Model

The Presentation Model (PM) pattern [21] follows the view and model separation and uses observer synchronization. It aims to resolve the difficulties handling the view logic in the controller by organizing it within a set of widgets. The controller is not
clearly defined, but its functions handling interactions with domain and presentation logic are organized in a model class called presentation-model, which acts as a controller interacting with the domain layer. The view stores its state in the presentation-model or synchronizes frequently for state updates. The presentation-model could be considered as an abstraction of the view and it is not tied to a specific domain object while each view is tied to a unique presentation-model, and introduces complexity in developing the GUI.

3.1.2 Model-View View Model

The Model-View View Model (MVVM) proposed by Microsoft [34], is based on the Presentation Model Pattern. It aims to separate completely the processing logic and presentation logic, facilitating testing, maintenance and extension of applications. The pattern is structured under the same three components as the original MVC (see Figure 3.1). However, the view is decoupled from the model via the controller facilitating the GUI development, and maximizing the re-usability of the components. The view encapsulates the state and view logic, and the view-model handles controller functions. These functions include queries, observers and coordinates updates to the model. As well as converting, validating, and aggregating data as necessary to display the view. Data binding plays an important role in the realization of MVVM applications. The GUI needs to implement the appropriate change notification interface to keep up to date with the data changes. Data binding can also be a drawback if it is used excessively as it can affect the application’s performance. Silverlight ¹ and Windows Presentation Foundation (WPF) ² are Microsoft technologies that support MVVM.

3.1.3 Model-View-Presenter

The Model-View-Presenter (MVP) pattern [21], establishes a clear separation between the model and view, where the presenter handles view logic and model interaction (see Figure 3.2; it does not require implementing the observer pattern. The Presenter

¹http://www.microsoft.com/silverlight/
Figure 3.2: Model View Presenter Design pattern

does not couple with a specific view but instead it couples with an abstract view interface [38]. There are two versions of MVP [40] [21]. The first one is the Passive view approach where the presenter is expected to handle all the interactions and the view logic; the view is decoupled from the model. The second version is the supervising controller approach, where the view interacts directly with the model to perform simple data binding. The presenter updates the model and handles view logic in case of high complexity of the GUI that cannot be specified declaratively. The advantages of MVP and MVVM are higher level of abstraction, facilitating GUI development and component’s testing. Examples of platforms that support MVP applications are ASP.Net MVP [33] and Google Web Toolkit 1.

Table 3.1, presents a summary of the patterns, and including a more standardized terminology to clarify points of comparison between each pattern. We used Fowler’s approach [21] to classify the view according to the two main functionalities. The first one, when the view is used to present the data output, is referred as view state. The second, when additional logic is required for specialized visualization of the content (for example, color coding the data, or using special figures to represent the output), is called view logic; this logic is differs from the processing logic. The view and the model coupling is considered as a limitation, when it keeps a direct reference to the model such as calling functions directly.

3.2 MVC Frameworks

Two popular platforms built following the MVC design pattern are Struts and Spring-Source [2, 55]. These frameworks are developed in Java 2 Platform Enter-prise Edition (J2EE) and use Extensible Markup Language (XML) configuration files which considers XML elements as descriptors in order to deploy objects. The Struts platform follows MVC design pattern for web applications development. The controller is based

1http://www.gwtproject.org/
3. MODEL-VIEW-CONTROLLER AND RELATED DESIGN PATTERNS

on the Service to Worker pattern; providing flexibility for integration of different view and models. Despite the advantages of integration support, other researches indicate that there are limitations for testing applications, due to certain complexity introduced in the view forcing inheritance and dependence of controllers on the dispatcher Servlet [14].

On the other hand, Spring is a combination of multiple design patterns including MVC, Aspect Oriented Programming (AOP), Java Database Connectivity (JDBC) Integration Framework and EJB Integration Framework. Simultaneously, it has a wide range of the view components including JavaServer Pages (JSP), ExtensibleStylesheet Language Transformation (XSLT), Java Server Faces (JSF), Velocity, Freemarker etc. for content generation. Spring framework supports DI pattern with Inversion of Control (IoC) to manage the JavaBean objects using runtime callbacks. Spring framework enables the developers to mix and match multiple frameworks such as Apache Struts and libraries while developing and deploying the applications [55]. The service object that is managed by the Spring IoC container is called as a bean. A bean is an object which is instantiated, assembled and managed by a Spring IoC container. The Spring container stores XML based configuration metadata that reflects the bean and its dependencies.

A similar research direction is discussed in [14]. It presents a model architecture decoupling the model from the view. To achieve the decoupling, the controller is designed as the applications entry point. It is mainly modelled with two patterns: The Intercepting Filter and the Service to Worker. A pre-processor in the controller is implemented by the Intercepting Filter pattern to manage the entry requests from clients in the presentation layer. The controller analyses each request, identifies the appropriate operation to perform invoking the associated business logic and controls the flow to the following view, as well as performing data marshaling. The Service to Worker design pattern encompasses a set of patterns including the Intercepting Filter allowing the separation of the actions and MVC components. A request dispatcher component is used to select the operation to be performed. It provides flexibility introducing different models or views by altering the components behaviour. The model is also realized with a set of patterns that facilitate the integration of distributed business logic and legacy applications. It applies Business delegate, Session Facade and EJB command patterns. To allow integration with legacy applications, the DAO pattern provides a mechanism to access the data sources. The model’s output is first produced in XML format to facilitate the subsequent transformation to XSLT. The implementation is realized through the combination of tree platforms, Struts as the controller, Cocoon for the view and StrutsEJB for the model. In our approach we are providing a single platform that supports the separation and re-utilization of the components, which simplifies the development task.

Play is an open source framework ¹ for web application development. It incorporates the MVC pattern right into the structure of the application, hence reducing the complexity to integrate the components. The platform supports service development,

¹https://www.playframework.com/
and uses RESTful services by default. Play implements a route file that maps HTTP request to controllers, which extract the request information and produce an HTTP result representation, optionally using a view template. Finally it serializes the result representation and returns it as a response to the client. It also facilitates to handle distributed data [16].

3.3 Chapter Summary

The distinguish feature of the MVC design pattern is the reusability of application components. It was proven to be an effective strategy even it introduces additional complexity for simple component interactions making some difficulties during testing. Although the model do not need to keep a reference to the corresponding view and controller, the view still needs to keep a reference to the model. That is why changes in the functions require also making changes in the view. In order to overcome these limitations variations of the pattern were proposed. MVP Passive view and MVVM achieved view and model decoupling through the controller. However, at the time of implementation, the developer has to deal with the construction of the controllers for events handling as well as interaction to the view and model, which increase a certain degree of complexity. Although the re-usability of the processing logic in the model is higher, reusability of the controller is still a limitation.

The patterns were designed based on object oriented concepts, but might not be suitable for large distributed applications that requires tasks such as integration, intelligent routing and processing/business logic composition such in SOA. The concept of well-defined roles for the view, controller and model components can be further extended to support service-orientation concepts.
### 3. MODEL-VIEW-CONTROLLER AND RELATED DESIGN PATTERNS

Table 3.1: Model-View-Controller based Design Patterns comparison

<table>
<thead>
<tr>
<th>Design Pattern</th>
<th>View</th>
<th>Controller</th>
<th>Model</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVC</td>
<td>View state</td>
<td>Input and user interaction</td>
<td>Processing Logic</td>
<td>Loose coupled.</td>
</tr>
<tr>
<td></td>
<td>View logic</td>
<td>Data</td>
<td>Debugging Logic</td>
<td>Difficulties handling view logic.</td>
</tr>
<tr>
<td>PM</td>
<td>View state</td>
<td>Intermediate layer</td>
<td>Processing Logic</td>
<td>Loose coupled.</td>
</tr>
<tr>
<td></td>
<td>View logic</td>
<td>View logic</td>
<td>Data</td>
<td>Development complexity (GUI).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Part of model</td>
<td></td>
<td>View depends on synchronization mechanisms.</td>
</tr>
<tr>
<td>MVVM</td>
<td>View state</td>
<td>Intermediate layer</td>
<td>Processing Logic</td>
<td>Performance affected by excessive data binding</td>
</tr>
<tr>
<td></td>
<td>View logic</td>
<td>View logic</td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td>MVP</td>
<td>View state</td>
<td>Complex view logic</td>
<td>Processing Logic</td>
<td>Loose coupled</td>
</tr>
<tr>
<td>Supervising controller</td>
<td>Simple view logic</td>
<td></td>
<td>Data</td>
<td>View requires additional testing</td>
</tr>
<tr>
<td>MVP</td>
<td>View state</td>
<td>Intermediate layer</td>
<td>Processing Logic</td>
<td>View requires</td>
</tr>
<tr>
<td>Passive view</td>
<td>View state</td>
<td>View logic</td>
<td>Data</td>
<td>additional testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Part of model</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Part II

Part II: Virtual Model-View-Controller Methodology: Concepts and Design Strategies
Chapter 4
Virtual MVC Methodology, Design Pattern, and Architecture

Overview. This chapter discusses a holistic approach through a methodology to help reducing the complexity of assembling together the various layers and functionality required by a service-oriented platform. Our design is motivated from three main patterns: Model-View-Controller, Enterprise Service Bus, and Dependency Injection. The later two are associated with SOA. However, MVC was originally an Object Oriented design pattern that gained further popularity for web based application’s development. The essence of the MVC is to separate the presentation logic (view), from the processing logic (model). It introduces an interface integration layer (controller) to handle user interactions as well as increasing reusability of the software components [29]. This chapter introduces the details of the architecture and adaptations for Service-Oriented applications.

4.1 Motivation

To gain the interoperability and flexibility promised by SOA, the initial implementation can introduce certain complexity assembling together a set of different tools, and requiring specialized knowledge. Additional integration code might be necessary to deploy the services. Depending on the implementation technology and the integration approach, the learning curve associated to the programmers might add to the complexity of the implementation.

Several limitations have been identified in Chapter 2. Big Web services are the most suitable technology for enterprise integration, however the considerable learning time for new programmers to grasp SOAP structure, and the set of WS-* standards takes a toll on the perceived complexity and initial implementation time. REST Web services, are an alternative to reduce such complexity, and simplify the development as it is based on HTTP, XML, URI and MIME which are well known standards. Low effort is required to build the client, as it can be tested right from the browser, and deploying the service seems closer to building a dynamic web site. However, with REST there are still some difficulties when it comes to manage complex data types, enclosing large data in URIs, or firewalls restricting the use of GET and POST operations which
leads to other work arounds. The less additional tools, or special constructs that are required to implement a Web service based application, the lower the adoption barrier becomes.

To decrease the programming efforts implementing Service-Oriented applications, Virtual-MVC methodology brings together two different concepts. First, applications are based on Big Web services due to its stronger support for transactions, reliability, and support of QoS properties that helps guarantee a certain quality level in the communication, which is provided by the WS-* standards stack. The second one is the MVC reusability and loose coupling concept that was originally proposed for Object-Oriented applications, it is extended in the Virtual-MVC pattern to organize the service client and provider communication.

In the original MVC design pattern [35] shown in Figure 4.1, the view updates itself from the model, via the Observer pattern. The original pattern, works like a closed loop wherein the view talks to the controller, this as well connects to the model, which in turns talks to the view and controller to update its state [22, 35]. In this loop, there is always an association between the three layers. An important contribution of the MVC pattern is the reusability of software components due to the separation of functionality organized within the model, view and controller.

A typical MVC architecture for Web applications is presented in Figure 4.2. The view comprises the user interface and servlets implementation in a Web server for dynamic content. The communication to the controller are usually handled through API calls. Depending on the implementation, the communication can rely on data
4. VIRTUAL MVC METHODOLOGY, DESIGN PATTERN, AND ARCHITECTURE

Figure 4.2: Web based MVC architecture

binding such as the case of the Spring framework [54]. The controller contains the API logic and Data Access Object to encapsulate the Data-tier. The model consists of the Database and stored procedures. MVP Passive view and MVVM achieved view and model decoupling through the controller. However, at the time of implementation, the developer has to deal with the construction of the controllers for event handling and interaction to the view and model. The reusability of the processing logic in the model is higher, but the reusability of the controller is still a limitation.

In the presented methodology, we attempt to reduce complexity of Service-Oriented application’s development by organizing service clients and providers based on the MVC pattern. The concept of well-defined roles for the model, view, and controller components, is further extended to support service-orientation. The MVC pattern has been widely used in programming platforms to decouple the development of components. However, the view is not completely decoupled from the model, the controller increase the programming effort although the gains of flexibility are worth the work. The traditional pattern does not incorporate service oriented concepts. In this section we introduce the Virtual-MVC pattern, which central point, is the modification of the Controller into an Enterprise-Service-Bus to manage message, protocol, and data transformation between them. The Dependency Injection pattern is introduced in the Controller for service virtualization, decoupling the View from the Model. Service virtualization is one of the main characteristics of the modified pattern, hence its name as Virtual MVC.
4.2 Proposed Virtual Model-View-Controller Design Pattern

The previous patterns were designed based on object oriented concepts, but might not be suitable for large distributed applications that require tasks such as integration, intelligent routing and processing/business logic composition. In the traditional MVC, the controller acts as an interface integrator and transforms the request/response parameters of the end user view to the model. Therefore, the direct link between the model and the view of the original MVC design pattern was redefined by removing and reorganizing the controller as an integration layer. With this modification, a complete decoupling of the view from the model can be achieved. The redefined version of the MVC design pattern is shown in Figure 4.3, in which the controller acts as a single point of contact for the view layer and the model layer, thus implies higher privacy of the business logic in model from the view, and higher reusability of application components.

The Virtual-MVC extends MVC concepts for SOA adopting services as the primary means of domain logic representation. The traditional role of the controller being input and user interaction is extended to act as a communication channel between the view and the model. The Enterprise Service Bus (ESB) design pattern is introduced in the controller to perform non-functional and integration logic, for reliability, scalability and communication disparity [17]. In the ESB approach, integration broker capabilities are distributed across a loosely coupled integration network. The integration components are independently deployed and managed [9]. In our approach, the core design pattern
4. VIRTUAL MVC METHODOLOGY, DESIGN PATTERN, AND ARCHITECTURE

![Conceptual Business Layer Schema](image)

**Figure 4.4: Business logic layer conceptual schema**

in the ESB to manage the integration of components and service virtualization is the DI pattern [50]. Therefore, the main role of the controller is to integrate a service interface and ensure that certain QoS properties related to the integration such as security, performance, and availability, work by negotiating the Service Level Agreements (SLA) with the service consumer and the service provider.

The model is normalized into the business logic layer and the database, as shown in Figure 4.3. The business logic layer is organized as well into the Service Model as the Business Object Pool, and the services schema. The Data Model, storing the endpoint services definitions under a service schema including information such as user profile, tasks rules, session control, verification templates, etc. To process a given request, the model makes use of the Data Model, and the business logic organized under a Service Inventory. The Service Inventory is a design pattern standardizing a collection of services that belong to a set of domains [17]. The conceptual business layer schema is depicted in Figure 4.4. The Service Inventory contains the metadata description of the associated service endpoint deployed in the Service Model. To organize the services in sub-domains, we introduce the concept of *Application Engines*. The services under an application engine, are organized by functionality corresponding to an application, and registered with a Service ID defined by Application Engine. In the example the Application Engines have registered three services, containing as well a reference to its interface, the service endpoint reference to the physical service in the Business Object Pool, and the service method.

The supporting engines are plugged as add-ons and connected via special software bus. As these engines are loosely coupled with the main controller, any third party utility can be interfaced and integrated as an add-on to the main controller. The services are exposed by the controller using the corresponding service ID, hence it abstracts the physical implementation from the view. The Data Access Object (DAO) pattern is
used to transfer the state of the data from the database layer to the business access layer. This design is mainly motivated from the Apache’s Struts and SpringSource’s Spring frameworks [2, 55].

The view handles the user requests, interaction, and view logic. The association between the view and model is decoupled introducing the DI pattern, by injecting controller dependency as an abstraction layer where the privacy of the business logic is only exposed to the controller and remains protected from remote reference access.

### 4.3 Controller Architecture

Figure 4.5 describes how the applications and services are collaborated through the Enterprise Service Bus (ESB) by using the modified MVC design pattern. The model layer consists of available services (business logic) and data. The controller layer is the ESB for the purpose of interface integration between disparate and heterogeneous applications (i.e. view layer) and services (i.e. model layer). It can communicate with the ESB in two modes: **Business-To-Client** (B2C) or **Business-To-Business** (B2B). The integration task mainly involves Data Transformation, Protocol Transformation, Message Routing (Mediation), and Service Composition.

The Service Orchestration Engine is the ESB component allowing the View Layer to communicate with the ESB via the web services interface based on the XML Simple Object Access Protocol (SOAP) including the Hypertext Transfer Protocol (HTTP) or HTTPS protocols. The request from the View Layer follows an itinerary based messaging. The itinerary message represents a set of discrete message routing operations.
4. VIRTUAL MVC METHODOLOGY, DESIGN PATTERN, AND ARCHITECTURE

The ESB separates the service definition from the mechanism for locating and invoking the services. Depending on the request parameters given in the message itinerary, the ESB invokes the appropriate service components including Protocol and Message (Data) Transformation, Ad-hoc Service Stack.

The Ad-hoc Service Stack is a pool of auxiliary services. These auxiliary services are responsible for Security, Monitoring, State Maintenance (or Session Control), Auditing and Logging. However, the auxiliary services are non mandatory. Its usage is decided as per the SLA document, which is mutually agreed by both the service consumer (view layer) and the service provider (model layer). Every application and its corresponding request are mapped for certain auxiliary services and it is documented in the SLA. The Service Virtualization mediates a request from the view layer in order to invoke and execute the desired service, which is implemented in the model layer. All the service interfaces are registered in the Service Repository and Registry. The Event and Transaction Controller manages the web services transactions across the distributed systems including Local Database, and Message Queues for two-phase atomic commit and concurrency control. The Data Model for Service Configuration is used for the service related data processing such as service definition, transactional data etc. The ESB acts as a container for instantiating the dependent and the client objects as well as keeping the references of those objects in the service pool and then ESB injects the dependent objects into the client objects. The view layer knows only the ESB and it needs to fill up an itinerary request form, which is defined by the ESB. This form contains all the information about the required functionalities by the view layer.

The view layer issues a high level request and provides the input data payload. Accordingly, the ESB contains the service mapping information placed in the Service Registry and Repository. The services are deployed and maintained in the model layer. By using this service mapping, the ESB composes endpoint services that are necessary for request execution. The Service Virtualization mediates the respective services by using Service Mediation Pattern. This approach does not need the Universal Description Discovery and Integration (UDDI) registries and thereby, the view layer does not need to know the endpoint services. Therefore, it increases the service interoperability and reusability.

4.4 Dependency Injection on Virtual-MVC

Dependency Injection design pattern was initially proposed by Martin Fowler in 1996 and later on enhanced it in 2004 [20]. DI pattern is influenced by Johnson and Foote’s proposal on IoC [60]. It enables the client application to consume a service without knowing in advance the specifics or origins of the consumed service. Therefore, the end user application is not required to hold the knowledge about the interface, behaviour and implementation of a service. The DI pattern hides the service logic implementation that is not apparent from the interface [60, 43]. The client object neither looks up for its dependencies, nor knows the location or class of the dependencies. The client in the DI pattern has a passive role that requires an external mechanism to pass the dependent
objects. This mechanism is called as Injection. The main benefit of the DI pattern is that it removes the dependency from concrete implementation of the business logic [20]. In the proposed approach, the DI pattern is implemented in the Service Orchestration Engine.

The integration components such as Protocol Transformer, Data or Message Transformer, and Service Virtualizer, are registered in the local data model or database of the service orchestration engine. The integration components are considered as client classes in the ESB. Every integration component declares its dependencies in the form of an interface, which is implemented by some implementer component. This implementer component is also registered in the orchestration engine. Therefore, the integration components are managed by the orchestration engine acting as an ESB container.

The local database schema for storing the mappings of the integration components is shown in Figure 4.6. It shows the tuples of various integration components such as Data Transformer (DATA_TNSFR), Protocol Transformer (PROTOCOL_TNSFR) and Service Virtualizer (SERVICE_VRTL). One component can have many interfaces or dependencies. Hence, the columns Component ID and Dependency Injection Interface form a composite key for this mapping to uniquely identify particular component. There are three types of dependency injections such as setter, constructor and interface based that are denoted by the Dependency Injection Type. The Injection Function Name denotes which function will be injected for the dependent object. The Main Function Name specifies the bootstrap function of the class mentioned in the column Component Class.

The schema for storing the mappings of the dependency implementer components or dependency classes is described in Figure 4.7. This scheme refers to the integration components (Figure 4.6) by the foreign keys of Component ID and Dependency Interface. The column DI Implementer ID is mapped to the foreign keys. The column DI Implementer Class denotes the name of the dependency implementation class. Figure 4.7 shows the tuples of various kinds of DI implementers, for example, in case of Data Transformation there are XML Data Transformer (XML_DATA_TNSFR) Comma–Separated Values or CSV Transformer (CSV_DATA_TNSFR) and JavaScript Object Notation or JSON Transformer (JSON_DATA_TNSFR).

An algorithm for implementing the DI pattern in the service orchestration engine is explained in Figure 4.8. The orchestration engine receives an itinerary based request

<table>
<thead>
<tr>
<th>Component ID</th>
<th>Component Class</th>
<th>Dependency Injection Interface</th>
<th>Dependency Injection Type</th>
<th>Injection Function Name</th>
<th>Main Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA_TNSFR</td>
<td>DataTransformer</td>
<td>DataTnsfrInterface</td>
<td>Setter</td>
<td>setDataTnsfr</td>
<td>beginDataTnsfr</td>
</tr>
<tr>
<td>SERVICE_VRTL</td>
<td>ServiceVirtualizer</td>
<td>ServiceVrtlInterface</td>
<td>Setter</td>
<td>setServiceVrtl</td>
<td>beginServiceVrtl</td>
</tr>
</tbody>
</table>

Figure 4.6: Component’s integration mapping
4. VIRTUAL MVC METHODOLOGY, DESIGN PATTERN, AND ARCHITECTURE

<table>
<thead>
<tr>
<th>Component ID</th>
<th>Dependency Interface</th>
<th>DI Implementer ID</th>
<th>DI Implementer Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA_TNSFR</td>
<td>DataTnsfrInterface</td>
<td>XML_DATA_TNSFR</td>
<td>XMLDataTnsfr</td>
</tr>
<tr>
<td>DATA_TNSFR</td>
<td>DataTnsfrInterface</td>
<td>CSV_DATA_TNSFR</td>
<td>CSVDataTnsfr</td>
</tr>
<tr>
<td>DATA_TNSFR</td>
<td>DataTnsfrInterface</td>
<td>JSON_DATA_TNSFR</td>
<td>JSONDataTnsfr</td>
</tr>
<tr>
<td>PROTOCOL_TNSFR</td>
<td>ProtocolTnsfrInterface</td>
<td>HTTP_PROTOCOL_TNSFR</td>
<td>HTTPProtocolTnsfr</td>
</tr>
<tr>
<td>PROTOCOL_TNSFR</td>
<td>ProtocolTnsfrInterface</td>
<td>FTP_PROTOCOL_TNSFR</td>
<td>FTPProtocolTnsfr</td>
</tr>
</tbody>
</table>

Figure 4.7: Dependency implementer of mapping scheme

message from the view layer. This message is processed by the orchestration engine in order to analyze what are the various integration components, and the respective dependency implementer classes required in order to execute the request. The algorithm assumes that the request message in \( \text{req}_\text{msg} \) is analyzed. The details about integration components in \( \text{comp}_\text{tobe}\_\text{executed} \) and DI interfaces in \( \text{DL}_\text{interfaces}\_\text{required} \) are known. The orchestration engine loads the components mapping in \( \text{inte}\_\text{comp}_\text{map}_\text{schema} \), as given in Figure 4.6, into the memory at the runtime. Next, the orchestration engine loads the DI implementation mapping in \( \text{dep}\_\text{imple}\_\text{map}_\text{schema} \), as given in Figure 4.7. Then it traverses the list of all the integration components in \( \text{comp}_\text{tobe}\_\text{executed} \) within a loop, and instantiates a component in every iteration to form a client object. During the same iteration, it instantiates the corresponding dependency interface implementer class and injects that dependency object into the client object. At the end of the every iteration, the orchestration engine invokes the main function of the integration component in order to execute the required functionality.

To explain the integration components of data transformer; a Java code snippet is shown in Figure 4.9. The class \textit{Controller} acts as an Assembler, which is a service orchestration engine in the proposed ESB. The class \textit{DataTransformer} is registered as an integration component in the Controller. There are various types of data formats as shown in Figure 4.7, such as XML, CSV, and JSON. The class \textit{DataTransformer} converts or transforms one type of data format into another. Therefore, the class \textit{DataTransformer} depends on the function of data transformation. Thereby, this function can be considered as a dependency for DataTransformer.

The dependency function is declared in the \textit{DataTnsfrInterface} by two methods. Firstly read the input data format by \textit{readInputData()} and secondly convert the input data format to the output data format by \textit{convertToOutputDataFormat()}. The output data format is same as that of model layer. Every type of data transformation can be realized by a separate dependency implementer class that implements the \textit{DataTnsfrInterface} dependency interface.

Figure 4.9 shows how XML data transformation and the class \textit{XMLDataTnsfr} implement the \textit{DataTnsfrInterface}. The interface implementer class \textit{XMLDataTnsfr} is registered in the Controller. The Controller receives a request message (\textit{String reqMessage}) from the view layer, and the request message is processed by a function
Variables:

comp_to_be_executed []; 
DL_interfaces_required []; 
inte_comp_map_schema = nil; 
dep_imple_map_schema = nil;

Input:

req_msg; 
DB_Conn = Database connection of local database of Controller;

Action:

Accept itinerary based request message from the view layer; 
req_msg := ReadMessage();
comp_to_be_executed := FindCompToBeExecuted(req_msg);
DL_interfaces_required := FindInterfacesToBeRequired(req_msg);
inte_comp_map_schema := LoadInteCompMapSchema(DB_Conn);
dep_imple_map_schema := LoadDepImpleMapSchema(DB_Conn);
for all ci ∈ comp_to_be_executed do 
begin 
Get the details of ci from inte_comp_map_schema 
Instantiate ci; 
for all di ∈ DL_interfaces_required do 
begin 
Get the details of di from dep_imple_map_schema 
Instantiate di; 
Inject di into the dependency injection function of ci; 
end 
Invoke the main function of ci; 
Do some post-processing if any; 
end

Figure 4.8: Algorithm for DI implementation in controller’s Service Orchestration Engine

ordinationEngine. At the beginning, the function orchestrationEngine instantiates the DataTransformer integration component in order to form a client object, then it instantiates XMLDataTnsfr in order to form a dependency object. The dependency object is injected into the client object by using setter type of injection. After this, the client object is ready to process the data transformation task. The orchestrationEngine invokes the main function of the DataTransformer to begin the XML data transformation. The DataTransformer does not need to perform expensive initialization logic and therefore, the dependent objects can be tested separately in isolation. As a result of this autonomy, there is a potential scope for a modular programming paradigm such as Aspect Oriented Programming (AOP).
4. VIRTUAL MVC METHODOLOGY, DESIGN PATTERN, AND ARCHITECTURE

4.5 Virtual Model-View-Controller Architecture

The architecture of Virtual-MVC framework and components interaction is presented in Figure 4.10. The view is realized as a servlet to manage the interaction with the controller. At the view layer, specialized visualization APIs for GUI can be incorporated. The view is deployed in a Web server. The ESB Controller is realized as an EJB container and deployed within an application server. The logic to manage data mapping, protocol and data transformation is contained in the transformation services layer. The DI is implemented in the service orchestration engine, for service virtualization. Local persistence is used to register the controller’s configuration properties, such as remote location of the model, database access, metadata, and controller XSLT rules and request/response messages definition.

The model is implemented as an EJB container, and service endpoints are also deployed within the model. Service endpoints can be used to wrap the logic of API’s and legacy applications. The DAO layer is used to connect with local and remote database servers. The access is configured as part of the model properties. The database is considered as part of the model as well. WebSphere application server is used to deploy the platform components (model-view-controller), which are as a framework to the developers. The architecture allows that each of the components deployed in different servers in distributed environments. The developer’s work is concerned to design...
the presentation logic in the view using technologies such as JavaScript, Ajax, etc. or through visualization APIs. At the controller the development corresponds to the request/response message definition. At the model, the development efforts are concentrated on the service endpoints. Due to the decoupling, and clear separation of components, it facilitates that multiple developers can be involved in creating applications. The programming approach is further explained with an example using a case based scenario. In the current realization the base programming language in the model is Java. However any other language that supports Web services standards could have been used for the implementation.

### 4.6 Chapter Summary

This section presented the proposed Virtual MVC design pattern. The MVC design pattern extends the controller to form the ESB. The ESB performs high level functionalities so that the integration logic becomes independent. The ESB enhanced with DI pattern reinforces the EAI by facilitating service loose coupling, global service contract, service implementation autonomy, service reusability, service modularity, statelessness and dynamic discoverability of the services. The DI pattern benefits the software development by bringing forth an abstraction layer between high level modules and low level modules. As a result of this abstraction layer, the integration components can be developed, deployed, tested and replaced independently. Therefore, the low level integration tasks can be seamlessly orchestrated. Security functionality is embedded into the ESB, it wraps the sensitive Data Model providing data privacy, which is a crucial hindrance for adapting Cloud Computing driven applications.
4. VIRTUAL MVC METHODOLOGY, DESIGN PATTERN, AND ARCHITECTURE

Service virtualization has abstracted peer services from transparent service lookup. The DI pattern enables the ESB to grow or shrink as required by the network and workload being supported. The integration components become more extensible because it can be used with different kinds of implementations of the dependent code without undergoing any modifications in it. Due to such loose-coupling attribute, the DI can be used as Plug-Points of an application framework.
Chapter 5

Virtual Model-View-Controller Framework and Programming Strategy

Overview. The Virtual-MVC design pattern concept and architecture presented in the previous section is the baseline for a development framework supporting SOA applications. This chapter focuses on the development framework, the components of Service-Oriented applications, and the programming strategy. The approach is explained step by step using a case example, describing the coding efforts required for each of the model, view and controller components, and how the final application is assembled.

5.1 Virtual Model-View-Controller development framework

The Virtual-MVC architecture has been realized by means of a Java development framework because of its portability. It is also possible to choose another programming language that supports Web Services technologies. The framework runs under WebSphere Application Server breaking all the components into Model, View, and Controller, facilitating the programmers to implement Virtual-MVC applications. The advantage of the framework is that it enforces the Service-Oriented concepts from the initial design, as all the applications must be constructed from service compositions. The controller handles those compositions, and the integration code is abstracted from the developer. The detail of the framework is explained along this section. The programmer focuses on developing the components independently, as minimal references are required between them. The model is developed in Java language, and the database access is setup using external properties. The controller components are a set of XSLT files that uses the data structure in the model for data transformation. Because the View is independent from the Model it only takes care of creating the customized GUI.

The usage of an Application Programming Interface (API) facilitates the communication between applications, and functions calling from within a software program. Similarly Web services are means of communication between applications over a network, typically the Internet, and using well defined standards such as WSDL, SOAP, REST, and XML-RPC. Although a Web service can be considered as an API wrapped over HTTP, it might not expose all the functionality of an API. The Virtual-MVC
programming framework supports the usage of APIs in the model. However, the processing logic in the model is exposed itself as a services through the controller. As the view can be independently constructed, it can as well make use of visualization APIs to provide special features for the GUI.

5.1.1 Framework requirements

Virtual-MVC framework does not require an installation process in the developer’s machine. It can be imported as a project into a local copy in Eclipse, or other similar IDE. The size in disk is 3.8 MB, and 1GB of memory is recommended. It requires Java 6 or later. The usage of a Concurrent Version system (CVS) such as the one included in Eclipse CVS, or Git is recommended for a collaborative development environment.

At the server machine, the framework runs under Websphere Application Server to facilitate the scalability of the Service-Oriented applications. The requirements for implementation are those of Websphere Application Server.

The developer can program the code at the local machine, and transfer it to the main framework directory in the server. The services need to be registered into the metadata table to make them available for other programmers and applications.

5.1.2 Application Layout

The framework organizes the application components following the Virtual-MVC design pattern. The application layout is presented in Figure 5.1, and it is imported in the Eclipse IDE environment as a project for this example. Following we present the explanation of the directories that are used by the developer to add the customized model, view, and controller components.

- **source** The source directory contains the business layer. The model classes implementing the service endpoint are developed within the `sda.model.customerservice`.

- **config** The directory contains the property files used to configure each of the components.

- **web** The view layer corresponding to a web based application is organized in the web directory. The corresponding HTML files, JavaScripts, CSS, etc. and multimedia files used to develop the GUI are placed in this directory.

- **xslt** The controller components are organized in two directories under `xslt`. The first is `controller` containing the request parameters for view to controller transformation. The second directory `view` Contains the output to be render back to the view, in the controller to view transformation.

---

1 https://www.eclipse.org/downloads/
2 http://git-scm.com/
3 http://www-01.ibm.com/software/websphere/?lnk=mprSO-webs-use
4 http://www-01.ibm.com/support/docview.wss?uid=swg27023941
Figure 5.1: Virtual MVC application layout
5. VIRTUAL MODEL-VIEW-CONTROLLER FRAMEWORK AND PROGRAMMING STRATEGY

- **Referenced Libraries** The developer can add additional APIs to the project creating this customized library.

### 5.1.3 Framework settings

The Model-View-Controller components in the Virtual-MVC framework can be setup externally for portability of the code between servers and instances of the framework supporting distributed applications. The directory that contains the configuration files is `config` (See Figure 5.1).

The `view.properties` file, requires four properties to be setup. (1) The view uses a predetermined directory to transfer file resources uploaded during the input process. (2) The second property contains the path to the XSLT file that should be rendered as response from the controller. (3) The destination folder for file uploads should be specified. (4) The last property is the URL address of the controller.

```properties
FILE_UPLOAD_SRC_PATH= C:\userPath\fileuploadsource
VIEW_XSLT_PATH=C:\userPath\eLC2_Dev\xslt\view
FILE_UPLOAD_DEST_FOLDER=/uploadedfiles
CONTROLLER_EJB_URL=iiop://localhost:port#
```

In the `controller.properties` file, the properties related to the controller component are as follows. (1) The path to the XSLT directory. (2) The URL connection to the model. (3) the user and password (4) to access the metadata information. (5) The database driver. (6) The database location.

```properties
CONTROLLER_XSLT_PATH=C:\userPath\eLC2_Dev\xslt\controller
MODEL_EJB_URL=iiop://localhost:port#
DBCON_USER_ID=
DBCON_USER_PSWD=
DBCON_DRIVER=com.mysql.jdbc.Driver
DBCON_URL=jdbc:mysql://127.0.0.1:port#/databaseName
```

The `model.properties` file, contains the properties used to access the corresponding database in the model. It is self explanatory in the code below.

```properties
DBCON_USER_ID=
DBCON_USER_PSWD=
DBCON_DRIVER=com.mysql.jdbc.Driver
DBCON_URL=jdbc:mysql://127.0.0.1:port#/databaseName
```

### 5.2 Programming Strategy

Within the Virtual-MVC framework, a programming strategy to development Service-Oriented (SO) applications is proposed. The strategy facilitates the understanding of
the efforts required by the developer to design the model, view and controllers components. A service development and deployment scenario is discussed in this chapter. The scenario consists on the design of an e-learning web application to manage the creation of problem’s collection. A problem can include diagrams, formulas, question and answer fields. The user can add a new problem to the collection, edit the problem’s data, and remove problems from the collection. In the example scenario, it is assumed that the functionality does not exist yet in the model. Otherwise, the development will be concentrated in the view. As it was explained previously, each component can be designed independently.

5.2.1 Application Case Scenario

Virtual-MVC design pattern main objective is to increase components reusability and facilitates the development of SO applications. Therefore, to develop an application, each component is designed separately, and finally composed in a workflow. To explain process, we demonstrate the development of an e-learning web application to manage a standard collection of problems for an educational course. In this example the end users are teachers. The application allows managing a problem collection by adding, editing, and deleting problems in the collection. It is assumed that the functionality does not exist yet in the model.

5.2.2 Designing the Model

The programming at the model, is mainly focused on designing the service endpoints (business logic). The application’s functionality is divided into atomic operations. To facilitate reusability, each service should be designed as agnostic as possible, and avoid direct calls between other endpoints, and representing one specific and autonomous function. Development of each service is simplified to design of four elements, which are represented as Java classes (See Figure 5.2). The elements are: 1) An interface class. 2) The service endpoint containing the main business logic which is the main function the service is providing. 3) A request and response value objects containing the input and output data. 4) An optional class containing a data structure. Using the problem editor example, the services in the Virtual-MVC model are designed according to the following steps.

- The main application functions are designed as atomic operations. For each operation, the corresponding input and output data is as well identified. The atomic operations for the Problem editor application are summarized in Table 5.1.

- The inputs are implemented in the class RequestValueObject, and the outputs into the ResponseValueObject class for each of the service respectively.

- In the example, the problem is designed as a data structure, which is used in the input or output depending on the problem. To add a new problem the data
5. VIRTUAL MODEL-VIEW-CONTROLLER FRAMEWORK AND PROGRAMMING STRATEGY

![Service endpoint design schema](image)

**Figure 5.2: Service endpoint design schema**

<table>
<thead>
<tr>
<th>Operations</th>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add new problem</td>
<td>+ Problem (structure)</td>
<td>+ Confirmation message</td>
</tr>
<tr>
<td>Show problem catalog</td>
<td>+ Catalog Id</td>
<td>+ Catalog (structure)</td>
</tr>
<tr>
<td>Retrieve problem</td>
<td>+ Problem Id</td>
<td>+ Problem (structure)</td>
</tr>
<tr>
<td>Edit problem</td>
<td>+ Problem (structure)</td>
<td>+ Problem (structure)</td>
</tr>
<tr>
<td>Delete problem</td>
<td>+ Problem Id</td>
<td>+ Confirmation message</td>
</tr>
</tbody>
</table>

| Problem structure is used in the Request value object, but for retrieving a problem, it is used in the Response value object. |

- A service interface is designed extending from ModelMainService class from the framework. The business logic is implemented in a Java class referred as a service endpoint, and contains the service method.

- Finally, the service is registered in a metadata table in the model. The metadata registered is the service interface, endpoint service, and the method. A service Id is assigned, according to its corresponding the Engine (service domain).

The specific endpoint name and operation is protected from external calls at the view. The service classes are deployed at the model, under `model/customservice/ directory` (see Figure 5.3).

The Request and Response value object classes hold the data type’s definition for which the transformation is handled by the controller. The service implements an interface that extends from the Model Main Service, and abstracts the details of the SLA that are managed at the controller by the Orchestration Engine. Additional
objects can be defined in the model to hold customized data structures. The service interface, endpoint service, and service method are registered at the Data model, and a service Id must be assigned, according to corresponding Engine (service domain). Following it is presented an example of the model elements, and a snippet of the corresponding code.

The Service Interface contains the signature of the Service endpoint’s method. For example, to add a new problem a service called AddProblem is created, a snippet if the corresponding interface is shown below. The request value object RequestValueObjectAddProblem is passed as the input parameter in the prm_in variable.

```java
package sda.model.customservice;
import sda.model.ModelMainService;
import sda.model.ModelServiceConnector;

public interface InterfaceAddProblem extends ModelMainService{
    public ResponseValueObjectAddProblem addProblem(ModelServiceConnector prm_connector, RequestValueObjectAddProblem prm_in);
}
```

The Service Endpoint class contain the business logic of the service. It implements the InterfaceAddProblem and the corresponding method addProblem. The input data is provided by the request value object RequestValueObjectAddProblem, and the output of this class is send to the controller through the response value object ResponseValueObjectAddProblem. The developer does not need to deal with the Web service interface definition (WSDL), and concentrate on designing the core business logic simplifying the development. The database connection it setup in the model.properties configuration file. The class handling the Database connection is ModelServiceConnector. The programmer assigns the connection using the prm_connector, and then uses a normal prepared statement. As it is shown in the example, the Virtual-MVC framework doesn’t interfere with special constructions beside the ones explained previously. Therefore, there is no need of special knowledge besides Java language to start working with the framework.

```java
package sda.model.customservice;
```
5. VIRTUAL MODEL-VIEW-CONTROLLER FRAMEWORK AND PROGRAMMING STRATEGY

import sda.model.ModelServiceConnector;
public class AddProblem implements InterfaceAddProblem{
    public ResponseValueObjectAddProblem addProblem
        (ModelServiceConnector prm_connector, RequestValueObjectAddProblem prm_in){
        int lca_count;
        ResponseValueObjectAddProblem lca_out = new ResponseValueObjectAddProblem ();
        // Corresponding processing code
        // Saving into the database
        lca_preps = prm_connector.fetchDBConnection("STD").prepareStatement
            ("INSERT INTO problem_catalog(id, description, diagram, formula, variable, constant) VALUES(?, ?, ?, ?, ?, ?)");
        lca_preps.setString(1, prm_in.id);
                lca_preps.setString(2, prm_in.description);
                lca_preps.setString(3, prm_in.diagram);
                lca_preps.setString(1, prm_in.formula);
                lca_preps.setString(1, prm_in.variable);
                lca_preps.setString(1, prm_in.constant);
        lca_count = lca_preps.executeUpdate();
        // more code
        lca_out.message = "The problem has been added to the collection";
        prm_connector.isTxnSuccessful = true;
        lca_out.isTxnSuccessful = true;
    }catch(Exception e){
        prm_connector.isTxnSuccessful = false; lca_out.isTxnSuccessful = false;
        // Add here any output parameters to handle errors
        e.printStackTrace();
    }finally{try{ //add your code }catch(Exception e){ e.printStackTrace(); } }
    return lca_out;
}

The Request Value Object class contains the set of the request parameters that are required by the service endpoint. The parameters declared in this class should be consistent with the request message defined in the controller.

package sda.model.customservice;
import sda.model.ModelMainValueObject;
public class RequestValueObjectAddProblem extends ModelMainValueObject{
    public Problem [] problemData;
}

The Response Value Object class contains the response parameters that are returned
by the service endpoint to the controller, where it will be unmarshalled and transformed to the view response.

```java
package sda.model.customservice;
import sda.model.ModelMainValueObject;
public class ResponseValueObjectCalcAddition extends ModelMainValueObject{
    public int answer;
}
```

The `Problem` class holds the structure of the problem. It extends from `ModelMainValueObject`, and it is used to declare the data types, and then used in the Request or Response Value Objects for the Controller transformation.

```java
package sda.model.customservice;
import sda.model.ModelMainValueObject;
public class Problem extends ModelMainValueObject {
    public int id;
    public String description;
    public String diagram;
    public String formula;
    public String variable;
    public String constant;
}
```

Finally, the service elements are registered in the metadata table. The services registered for this example are shown in Figure 5.4

<table>
<thead>
<tr>
<th>Service ID</th>
<th>Service Interface</th>
<th>Service Endpoint</th>
<th>Service Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>PED-001</td>
<td>sda.model.customservice.InterfaceAddProblem</td>
<td>sda.model.customservice.AddProblem</td>
<td>addProblem</td>
</tr>
<tr>
<td>PED-002</td>
<td>sda.model.customservice.InterfaceShowCatalog</td>
<td>sda.model.customservice.ShowCatalog</td>
<td>showCatalog</td>
</tr>
<tr>
<td>PED-003</td>
<td>sda.model.customservice.InterfaceRetrieveProblem</td>
<td>sda.model.customservice.RetrieveProblem</td>
<td>retrieveProblem</td>
</tr>
<tr>
<td>PED-004</td>
<td>sda.model.customservice.InterfaceEditProblem</td>
<td>sda.model.customservice.EditProblem</td>
<td>editProblem</td>
</tr>
<tr>
<td>PED-005</td>
<td>sda.model.customservice.InterfaceDeleteProblem</td>
<td>sda.model.customservice.DeleteProblem</td>
<td>deleteProblem</td>
</tr>
</tbody>
</table>

Figure 5.4: Metadata table with services registered for Problem Editor

5.2.3 Designing the Controller

At the controller, the first step of the ESB communication is the protocol and message transformation. The Protocol Transformation engine provides transport bindings for
invoking services over HTTP, SOAP, File Transfer Protocol (FTP), Java Message Service (JMS) etc. The Message Transformation engine acts as an interpreter to map the request parameters with their corresponding Service Model's Data Transfer Objects (DTOs). XSLT is adopted for message transformation, and the ESB at the controller uses the XML standard for exchanging messages. Instead of Namespace based service lookup, message routing is used to invoke services dynamically. The workflow is carried out as an abstract message flow provisioning the message and events chaining.

The controller uses Itinerary based message routing based on a predefined sequence of activities. The Orchestration Engine routes the message based on the Itinerary to the desired service. By using Event Controller and Service Mediation engines, service Orchestration Engine binds WSDL service's interfaces.

There are two components created at the controller to transform XML request/response messages, using Extensible Stylesheet Language (XSLT). The request component defines the input parameters, and it is mapped to the corresponding RequestValueObject in the model. The response component contains the output parameters to be rendered to the view. The name of both XSLT files is by convention the name of the Service Id registered in the Services Metadata Definition, and located into different directories separating the the input and output transformation files. The following examples show the structure of the controller’s components.

The first component is the request XSLT file based on a template that specifies the Service Id and the <RequestValueObject> element with the corresponding name for that service in this case is <RequestValueObjectAddProblem>. The child nodes are the corresponding variables defined in the RequestValueObject Java class, and use the same name. The value of the children nodes corresponds to the parameters designed at the view. The following snippet, shows an example of the controller’s request for adding a problem.

```xml
<xsl:template match="/">
  <ControllerRequest>
    <ServiceID>PED-001</ServiceID>
    <ServiceRequest>
      <RequestValueObjectAddProblem>
        <problemData>
          <description>
            <xsl:value-of select="//elc2/viewrequest/viewDescription"/>
          </description>
          <diagram>
            <xsl:value-of select="//elc2/viewrequest/viewDiagram"/>
          </diagram>
          <formula>
            <xsl:value-of select="//elc2/viewrequest/viewFormula"/>
          </formula>
        </problemData>
      </RequestValueObjectAddProblem>
    </ServiceRequest>
  </ControllerRequest>
</xsl:template>
```
The second component is the **controller response**, that renders the data from the model into the view. XSLT is used to render the view. In the following snippet example, the given path is updated with the corresponding *response value object* and the output parameter in this case *message*.

```html
<table align="center" border="0" cellspacing="2" cellpadding="2" width="70%">
  <tr>
    <td width="30%">Answer of Addition : </td>
    <td width="70%">
      <xsl:value-of select="/elc2_dev/modelresponse/responsevalueobjectaddproblem/message"/>
    </td>
  </tr>
</table>
```

Figure 5.5 shows an example of the request/response messages deployed with the framework. The request message is saved under the *controller* directory, which represents the view to controller messages. The *view* directory contains the response messages that will be rendered by the view.

### 5.2.4 Designing the View

The view hosts the presentation as a Web portal. A Web container is clustered to host static or dynamic HTML pages, CSS, JavaScript, AJAX, images and multimedia objects, etc., the view issues a high level request to the controller and provides the input data payload, participating only on input and output exchange messages. The view can be developed independently as it is outsourced from the processing task. The view only knows the *service Id* in the controller, and has no knowledge of the service in the model, or function to process the data. Hence its decoupled from the implementation details in the model. The following example shows a snippet of a simple view developed in HTML. The form action contains the call for the view (Assuming that is under the view server). The parameters are passed in the input elements of the form such as *viewDescription, viewVariables, viewConstants, viewFormula*, and *viewDiagram*. The
service ID is passed using a hidden text element argServiceId. There is no direct reference to the model encoded. Hence any change of the implementation code is decoupled from the view.

```html
<form action="view" method="post" name="inputProblem" enctype="multipart/mixed">
  <table width="95%" align="center">
    <tr> <td> <h2>V</h2> </td> </tr>
  </table>
  <table align="center" border="0" cellspacing="2" cellpadding="2" width="90%">
    <tr>
      <td width="30">Problem description</td>
      <td width="70"> <textarea rows="7" cols="40" name="viewDescription"></textarea></td>
    </tr>
    <tr>
      <td width="30">Variables</td>
      <td width="70"> <textarea rows="3" cols="40" name="viewVariables"></textarea></td>
    </tr>
    <tr>
      <td width="30">Constants</td>
      <td width="70"> <textarea rows="3" cols="40" name="viewConstants"></textarea></td>
    </tr>
    <tr>
      <td width="30">Formula</td>
      <td width="70"> <input type="text" size="32" name="viewFormula" /></td>
    </tr>
    <tr>
      <td width="30">Image File: </td>
      <td width="70"> <input type="file" name="viewDiagram" /></td>
    </tr>
  </table>
</form>
```
5.2.5 Assembling the application

The component’s interaction for AddProblem service, is summarized in Figure 5.6. The view contains the GUI, and issues a high level request using the service ID and a set of parameters that constitute the input payload data to the controller (1). The service ID corresponds to a service endpoint registered in the Service Metadata Definition table (2). Using the XSLT schema, the controller routes the requests to the appropriate elements, such as value object, service endpoint, interface, endpoint data, and the corresponding function (3). The mapped data is in string format and needs to be inflated into the Data Transfer Object (DTO) of the service model. The controller deserializes a series of bytes into DTOs. After the completion of the service request in the model, the controller unmarshalls the Response Value Objects corresponding to that service endpoint (4), rendering a response message in XML which is delivered to the view via the controller (5), and produces the presentation in HTML, with the look and feel from its corresponding HTML, CSS, JavaScript, etc (6).

The rest of the services follow the same development approach as AddProblem. The Problem Editor web GUI calls the services in the model, and assembles the view as an
end user application. Similar with the traditional MVC, the services can be used in different applications. Moreover, the services can be consumed by external applications such as Moodle by connecting directly with the controller.

5.3 Chapter Summary

The example presented shows the application development process using the Virtual-MVC framework. The development of the view and model can be carried out independently. The framework does not impose special annotations or constructs besides the basic service elements described in section 5.2.2. Therefore, the main service logic can be implemented in Java and there are no special skills to use the framework, besides basic Java and knowledge of Web development technologies.

The controller approach in Virtual-MVC minimizes the developer’s efforts for service’s integration, and composition. The controllers components are used to define the request and response parameters for message transformation. Basic knowledge on XML, XPath, and XSLT is required. The controller uses Itinerary based message routing which is an effective approach for highly distributed applications [9].

The view can be constructed independently and only participates in providing input and output requests. The technology used for the view development is flexible and depends on the developer’s choice.

The components independence provided by the framework, makes the Virtual-MVC a suitable programming approach for collaborative environments. The framework and programming approach enforces the design of the applications as service components. Supporting atomicity, loose coupling, composability, and reusability of the business logic.
Part III

Part III: Development and Evaluation
Chapter 6

Virtual Model-View-Controller based Applications

*Overview.* The adoption of Cloud computing model such as the provision of Software-as-a-Service (SaaS), discussed in [39], is based on the premise towards a more cost-effective solution where the customer should pay only for the consumed services. Therefore, the SaaS model is in line with the next generation of e-Learning platforms. The implementation aspect on e-Learning SOA frameworks is still an open area, and it requires more effort to be adopted and moved into the new generation of e-Learning tools. The research discussed in this section aims to contribute to the implementation of e-Learning SOA platforms within the framework of an e-Learning Computational Cloud. E-Learning applications are evolving from restricted monolithic designs to platforms embracing Service-Oriented concepts, to be deployed in Cloud environments. Traditional Learning Management Systems (LMS) offer a set of basic or general functionalities, trying to cover a wide range of needs. However, there might be functionality that the system does not have. It is necessary then to obtain this functionality by adding components from external vendors, or developing them in house. Developing e-Learning applications based on loosely coupled services facilitates reusability, and integration with heterogeneous applications. The next generation of e-learning platforms should allow the dynamic discovery and assembling of e-learning services.

A set of applications have been developed using the Virtual-MVC programming approach. To demonstrate the reusability of services components, two applications for E-Learning arena are discussed in this section. The first one is a glossing tool for language learning activities called *Wiki-Gloss tool,* and a *Task Management environment* that contains a group of e-Learning components, that can be integrated into a Learning Management System (LMS) such as Moodle, or they can be used as stand-alone applications. Although the focus of the applications are on E-Learning arena, The Virtual-MVC framework is not restricted to this area.

### 6.1 Task Management Environment

The Task Management Environment focuses on enhancing the collaboration between teacher and students for subjects that require formula expressions, where typically it is
difficult and time consuming the task of designing and personalizing of exercises and test materials. The instructor requires to spend a significant portion of time and effort to customize the problems for each particular student. An important aspect of supporting the verification of formulas expressions is the capacity of producing automatic feedback for each particular version. According with an evaluation using Web–based assessments [56], automatic feedback helps to increase the confidence and comprehension perception of students, by giving the opportunity to practice an exercise multiple times and get the evaluation at the moment. The importance of real-time feedback is also highlighted as a principle of good practice design for instructional methods in education [7]. Automatic task generation is an important topic when it comes to the development of collaborative tools that support the guidance of students in the learning process. Research on this topic highlights the few or lack of support on automatic Task generation [31]. To facilitate the task of creation and evaluation of formula expressions, a component-based Task Management approach is proposed. that is important to the development of Web–based collaborative applications, and is an attempt for resources interoperability according to [1].

The Task Management environment is redesigned in order to adapt it into the Service Oriented Architecture (SOA) concepts, and design the logic of the application as e-learning services. To provide higher reusability of the components logic, we are integrating the components to an e-Learning Computational Cloud ($eLC^2$). The $eLC^2$ was developed based on the Virtual MVC design pattern. This modification supports the development of e-learning services under distributed environments as well as the reusability in different compositions. Therefore, the Task Management components can be reutilized in diverse applications reducing development efforts.

6.1.1 Task Management Principles

Accordingly, we define the Task Management as the process to organize a set of activities or Tasks that need to be executed to achieve a Learning Objective. An example of a Learning Objective in the Task Management environment can be a subject course. The simplest Task unit is referred in this paper as an Atomic Task ($AT$), defined as follows:

$$D = \{d_i\}_{0}^{i-1}$$  
$$P = \{ed_j\}_{0}^{j-1}$$  
$$AT = \{d_i \epsilon D, ed_j \epsilon P : \{ed_j, d_i\}\}$$  

(6.1)

For a given $AT$, a Problem $P$ has a set of element definitions represented by $ed_j$, there exist a set of input data $D$ for every $ed_j$, an instance of input data is represented by $d_i$.

Let the set of $AT$ be a Task Collection $TC$, where $l$ is the number of Tasks in the
6. VIRTUAL MODEL-VIEW-CONTROLLER BASED APPLICATIONS

collection. The TC is defined as follows:

\[ TC = \{ AT_l \}^{l-1}_0 \]  \hspace{1cm} (6.2)

In this paper we are concentrating on three main types of TC: Practice, Quiz, and Exam. The first type is the Practice (PR), which is the simplest type of TC, and it is defined as:

\[ PR = TC \]  \hspace{1cm} (6.3)

The second type of TC is the Quiz (Q), where \( a_l \) is the answer, and \( s_l \) the score associated with the AT, the definition is the following:

\[ Q = Evaluate(AT_l, a_l, s_l) \]  \hspace{1cm} (6.4)

The last type of TC is the Exam (E), it incorporates an additional parameter, being \( t \) the time assigned for completing the TC. The Exam is defined as:

\[ E = Evaluate(AT_l, a_l, s_l, t) \]  \hspace{1cm} (6.5)

We are proposing the Task Management principles to enhance the understanding of the organization of elements under the Task Management environment. Figure 6.3 presents an example of an input Problem, where the Problem’s element defined \( j \) of the AT corresponds to the name and description of the following parameters: constant, variable, question, and formula. In the parameter Problem description \(^1\) the element type that can be automatically generated, is represented by adding the name of the element within the square brackets ([ ]) characters. The Problem input data \( d \) is defined by the element value, this value can be specified during the Problem creation, or can be defined later by the user setting a range of values from which different versions of the Problem can be distributed to a large group of students, saving time to the instructor in the Problem creation.

6.1.2 Application overview and E-learning services

The initial Task Management approach [44] was implemented under client-server architecture. Accordingly, an Exercise server application manages a database of Problem collection, and a set of functions to facilitate the Problems preparation, distribution and automatic verification. Based on the Task Management environment, we are extending the Task definition to facilitate the verification strategies proposed in [58].

6.1.2.1 Formula Engine

The Formula Engine handles mathematical operations with formulas (see Figure 6.1). It consist of a Formula Parser that transforms a formula from text-based linear representation to an internal object tree structure and vice versa. The internal structure

\(^1\)The Problem statement presented in this example can be found at http://www.physicsclassroom.com/Class/vectors/u3l2e.cfm

60
is stored within the Formula Internal Container and it manages the exchange of the formula data with the other components. The Formula Calculator is used to obtain the numerical answer corresponding to the formula. The Formula Symbolic Transformer parses the formula using standard symbolic algebraic conversions like factoring out overall coefficients, reduction of similar terms, etc., to obtain the formula standard ordered form. To improve the perception of the formula in a graphical way, the Engine provides a Visual Formula Editor component to process visual objects associated with symbolic expressions within the formula, for example variables or constants. The Visual Formula Editor verifies that the correct construction of the visual formula, and that each visual object being part of the formula is able to produce linear text according with the syntax rules specified.

6.1.2.2 Verification Engine

The Verification Engine encapsulates the logic of the Task verification process. When the AT is created, the teacher can define the type of answer associated with the Problem. The answer types supported for an AT are: numerical and formula answer. For the numerical answer, the teacher can input directly the answer value or assign it randomly from a set of data. It is also possible to indicate the Engine to perform the automatic answer calculation based on the formula. The Engine uses a Precision rate parameter to validate the correctness of the answer. It compares the answer introduced by the student, and the system’s answer by calculating the absolute value of the difference. Then, it evaluates if this difference is within the range of the precision rate.

The generation of multiple versions of the Problem is handled as well by the Verifi-
cation Engine. During the AT creation, the Engine receives the variables element type as parameters. For each of the elements it also receives a value range, which is going to be used for a random generation of the variable value, to create multiple versions of the Problem depending on the number of students that are targeted for the AT. The Engine will calculate the answer based on the given values and save it in the database, as well as the automatic generated values to keep track of each version.

The Verification Engine applies three verification strategies to evaluate not only numerical answers but also formula expressions characteristic of CFES that are typical for mathematics, physics, statistics, chemistry, etc. The first one is the numerical answer verification, where the Engine performs a comparison between the student’s answer and the system’s answer. The second strategy is the multiple answer verification, which is applied when the AT is generated as multiple version. An example is an AT that was generated from a single Problem definition, but the data of a variable value is different for each student. The last strategy is the formula answer verification (see Figure 6.2), it is used when the answer of the AT corresponds to an equation system. In order to perform the formula answer validation, the Verification Engine sends a request to the Formula Engine with the student and teacher formula answer in text-based linear representation to be parsed and compared by the Formula Symbolic Transformer [58].

### 6.1.2.3 Formula Editor

This component is a Web–based interface designed to provide the graphical edition of a formula. The teacher can create a formula by uploading symbols to be associated with each of the formula arguments and represent it visually. Therefore, this component helps to provide additional means of representation for the formula, helping to stimulate the perception of the student[7]. The Formula Editor uses the logic of the Formula Engine passing the parameters to the Visual Formula Editor component to be parsed.
and create the linear representation to be saved in the database; saving as well the associated graphical representation.

6.1.2.4 Problem Editor

The Problem Editor component is a Web-based user interface that provides tools to create, modify and delete Problem’s collections associated to a subject. It helps to manage the definition of the Problem, specifying the type of elements and an initial value used as sample data to create ATs. The example in Figure 6.3 shows the Problem addition feature, the user can define each element type, assign a name, a description and an initial value; the value can be later replaced through the Task Management Editor.

6.1.2.5 Task Editor

The Task editor is a Web-based user interface to manage the AT edition and Task Collections, which are an implementation of the TC definition. The user can define a CT by selecting a set of Problems. For each individual Problem, the user can specify the data to create an AT. In the Task Editor the user can define the verification strategy that will be applied to every Problem. This interface uses the logic of the Verification Engine. Figure 6.4 shows an example of AT creation in which the user has selected to generate a multiple version of the Problem. The Task Management editor has been improved providing the user the option to define a range that will be used to produce the different variants of the Problem.

6.1.2.6 Task Management database

In section 2, we introduced the Task Management principles that are the basis for the element’s organization. Using the AT and CT principles, we are extending the database presented in our previous research [44]. Originally, the data and Problem description were defined within the Task table. To increase the flexibility to handle multiple answer generation and verification, in the extended database (see Figure 6.5) we have reorganized the information with a clear separation between the Problem definition and data, as well as the Task information concerning with the AT. Therefore, we provide more flexibility and facilitate the creation of multiple versions of the same Problem to customize it for each student, reutilizing the Problem definitions. An explanation of the main tables is presented as follows:

The table MDL_taskCollection, contains the general information of the Ctask that will be applied to each AT, such as the group of students assigned to complete the Task, assignment times and alerts. It also contains an external reference to the AT’s details, stored at MDL_atomicTask table.

The MDL_atomicTask table stores the individual Task version for each student, and the answer introduced by the professor or calculated automatically in the case they are generated as a variant of the Problem. It uses a reference to a common Problem
Figure 6.3: Example of Problem Editor view
description, but it is linked to an specific value of the data type for that Problem, which is stored in \textit{MDL.dataValue}. For example, the same Problem definition can be used to generate different parameters values from the range assigned to each variable. The difference between the answers can be calculated at runtime since the student and teacher’s answer for that specific Task are saved in the database.

\textit{MDL.problem}. It stores the main description of the Problem, this information is to be used directly as the Problem statement by the \textit{AT}.

\textit{MDL.dataDefinition}. Each Problem can have a different set of data definitions from the main element’s types associated to the Problem, for example the specific name of a variable. It also stores the interval to be used during the multiple answer value generation of each element definition, as well as the graphical element that can be used as special symbol parsed by the graphic formula editor.

\textit{MDL.masterProblem}. It keeps the reference of the main version of a Problem. The teachers can customize a Problem and create a new version, but the reference to the original will be stored. This is specially useful when the same Problem is translated in different languages, facilitate the collaboration between professors, and the re-utilization of the Problem as a resource.

\textit{MDL.dataValue}. This table stores the Problem data value. This provides more flexibility to create multiple versions of the Problem to be assigned in an \textit{AT}.

Figure 6.4: Example of Task Editor view
Figure 6.5: A Fragment of the Task Management Database Structure
independent of the Problem definition itself. For example, a variable can have different values for each of the Problem variants of an AT.

MDL\textit{dataTypeCatalog} table stores the general data types of the Problem such as: constants, variables, formula, questions, and the precision rate.

MDL\textit{resourceCollection}, stores other resources that can be associated with an AT. It can store multimedia and document files. This resources are tied to the subject to give flexibility to be re-used for different Tasks.

6.1.3 Usage scenario

The \textit{Task Management environment} can be extended to support a more robust e-Learning platform, or it can be integrated as a service to a Learning Management System (LMS) such as Moodle \footnote{https://moodle.org/}.

6.2 Wiki-Gloss Tool for Language Learning

A gloss in language learning often refers to provide a brief explanation of selected terms within a given text. These definitions can be obtained from additional tools like electronic or computer-mediated dictionaries. Glossing is related to promoting a positive effect in reading comprehension, and vocabulary learning, enhancing the likelihood of acquiring words incidentally as a by-product of reading \cite{28,30}. However, the task of choosing appropriate readings and complementary materials such as glosses can be a time consuming task for the teacher. A system that provides automatic glossing is beneficial for both teachers and students. Similar research is being undertaken by Keio University \cite{26}, where an automatic glossing system for Japanese language is being developed not only for concepts, but functional language expressions, obtaining positive results on its adoption. The difference with our approach is that the development is based on a specific application, since the main scope is the Japanese language, and the content uses a specialized English-Japanese dictionary, although it provides options to link to other web resources including Wikipedia. The advantage of our approach is that an automatic glossing tool developed as a service can be used, not only for language learning activities, but for different environments, and tasks that require the comprehension of specialized concepts such as contextual help components. Wikipedia has been chosen as a source of content because of its huge amount of knowledge and vast range of topics. The content is continuously updated due to its collaborative dynamic nature. It contains articles related to almost any kind of information, and references to their corresponding translations in numerous languages, as well as the relations among articles. These characteristics are very attractive to provide content flexibility for diverse learning scenarios.

This section focuses on the realization of an automatic glossing e-Learning tool for language learning activities. Initially the language used has been English, but it can be extended to other languages that are available in Wikipedia, which is the source of the
glosses. The approach we present adapts Wikipedia Miner techniques [36] in order to extract the required content from Wikipedia and feed the corresponding glosses. The WikiGloss application, has additional tools to help reinforce vocabulary such as review activities. It also provides topic translation linking with the corresponding article in different languages. The application was implemented in the framework of a distributed platform called e-Learning Computational Cloud (eLC2) [49]. The advantage of the eLC2 is its Service-Oriented design that supports re-usability and interoperability among different applications. Therefore, the services used to build WikiGloss can be easily incorporated into other applications that have similar glossing requirements.

The usage of support materials to provide additional guidance to students facilitates the comprehension of learning tasks. Wikipedia is one of the richest sources of human knowledge, encompassing a vast range of topics of all kinds of information, and content, which is in constant change due to its collaborative dynamic nature. The Wikipedia Miner tool provides the functionality to parse a given document identifying main topics and link them to corresponding articles or short definitions from the Wikipedia content. In this paper, we discuss the realization of a reusable Wikipedia Miner service for the e-Learning Computational Cloud (eLC2) Platform based on the Virtual MVC pattern. In this framework, Wikipedia Miner services were prototyped as an Application Engine that wraps the logic of the Wikipedia Miner API in order to re-use it for different types of applications. Particularly, we are focusing on two applications in order to demonstrate the usability of the proposed approach. The first application is the WikiGloss tool, which is based on a glossing approach to help learners of English-as-second-language with an extensive reading task. The second is an Intelligent Hints service for the Task Management Environment which provides explanatory links from relevant Wikipedia articles related to topics of the e-Learning task. This allows re-use of the same problems in different task type modes such as lectures, exercises, and quizzes.

### 6.2.1 Wikipedia Miner

Wikipedia is one of the richest sources of human knowledge, encompassing a vast range of topics and content that is in constant change due to its collaborative dynamic nature. Articles related to almost any kind of information can be found and contain references to their corresponding translations into different languages, as well as the relations with other articles. Moreover, linking the content of Wikipedia to use it for contextual help in documents where content is not restricted to any particular topic can enhance an e-Learning tool that supports reading comprehension, as well as tools that support learning activities. Articles for subjects like mathematics and physics can be easily found in Wikipedia. For these reasons, we are exploring the usage of Wikipedia Miner, an open source software tool that among its main functionalities facilitates the extraction and manipulation of Wikipedia content and structure in an object oriented fashion. It provides algorithms to produce semantic comparison between terms and concepts, and cross-reference documents with relevant topics [36].
6.2.2 Wikification process

The Wikipedia Miner API provides code that can parse a given document identifying the main topics that the document describes. These topics can be made into links to their corresponding articles or a short definition from the Wikipedia content, choosing the most appropriate meaning based on the surrounding text and the relation with other topics. In this document, we will refer to such process as Wikification. The Wikification process is summarized in the following steps. The first one is Candidate selection, which refers to the process of gathering terms or phrases that are considered to have high probability to be selected for a link. The second is applying disambiguation algorithms to decide which of the candidate topic senses should be selected. The final step is detection that decides which topics will be finally converted into links for the document. For example, lets consider a computer science article about data structures; it can be wikified in such a way that the main topics of the article are identified and the appropriate links to the Wikipedia articles are included in the document as glosses. The Candidate selection will identify the terms or phrases that correspond to an article in Wikipedia. Once identified, the disambiguation algorithm will be applied to choose the articles that have higher probability to be related to the surrounding context based on a relatedness measure. For example, if the document includes the word tree, the existent articles in Wikipedia that can be associated with this definition might be related to botany, data structures, graph theory, etc., since the article is referring to data structures Wikipedia Miner will select the sense closer to the other topics in the document; which in this case is the tree data structure. Finally the terms that represent the main topics of the document will be identified as links to Wikipedia articles. The details of the algorithm are covered in [36] [37].

6.2.3 Wiki-Miner Engine

In the model, we design the services corresponding to the Wiki-Miner Engine. Although the capabilities of the API are larger, the scope is defined by the following functionality: Topic detection. Parses the given text for candidate selection, link disambiguation, and returns the terms to be linked. Gloss data extraction. Gets as input an array of terms and returns an array of a gloss structure containing for each term, the article title, the first paragraph, and an image. Augment document. Gets the terms to be linked and the text, adding markups that will be processed at the view to insert the glosses information. Extract term senses. Extracts from Wikipedia a list of articles that correspond to that term, for example ambiguous definitions. Extract translation. Gets the title of the term corresponding with the language selected. Extract related topics. For a given term, it returns a list of related topics among the list of articles linked to that term. The endpoint services are stored in the model under the service inventory as an Engine layer. The advantage of the proposed development approach, is that the services can be configured for different language versions of Wikipedia.
6.3 WikiGloss Features and Design Issues

WikiGloss is a tool that supports reading comprehension activities by enhancing the text with glosses for vocabulary reinforcing activities. The features of the application are the following: text wikification, topic translation, and practice activities. In the text wikification step (Figure 6.6), the user can create a new document by introducing text or a web url; the corresponding text is parsed to identify which terms have corresponding articles in Wikipedia and which of them are worthy of being linked according to the context given by the surrounding text. The relevant articles are then presented as hypertext links, and a brief definition of the topic is displayed on the mouse-over event. The second feature is the topic translation (Figure 6.7) of the words detected as main topics into one of the languages available in Wikipedia and processed using Wikipedia Miner. The language can be specified in the general settings of the application. The last function is a set of practice activities (Figure 6.8). The application provides a set of activities to test vocabulary comprehension and reinforce retention. The activities are based on the topics found in the text. It prioritizes the vocabulary in accordance with the topics that have been reviewed during the reading task. Initially there are four practice activities described as follows.

**Guess the meaning.** The student is presented with a word and a set of definitions to choose the correct meaning in accordance with the recent reading.

**Words in Context.** This activity tests word comprehension in accordance with the context from which is used. The student is presented with a sentence with a blank space. The student has to select the correct word from a set of options that correspond to the context of the sentence. The options are chosen from the closest senses of the corresponding topic.

**Translate the word.** The student is presented with a word matching activity. There will be a set of words and an equal set of translations, and the student has to match the correct word with the translation.

**Choose synonyms.** In this activity the student should select the synonyms of a word, showing understanding of the topic and relations with other concepts.

6.4 Usage Scenarios

We envision two main usage scenarios for the WikiGloss application supporting language learning. The first one is non-guided usage. Students can use the tool for their language learning or reading activities outside of a classroom. The selection of the topic is not restricted, which is one of the advantages of using Wikipedia as a source of content. However, there are two main drawbacks to this scenario. The disambiguation performed by Wikipedia Miner algorithms might contain some inaccuracies, which can affect some of the review activities, for example, displaying synonyms that are not so closely related to the topic. Although the tool is designed to support reading and comprehension language activities, for a more complete support of extensive reading the text should be selected according to the level of the student. For example, in some
Figure 6.6: Example of the WikiGloss functionality

Figure 6.7: Example of Article translation gloss
personal use cases students may not be able to identify the appropriate level and choose difficult readings. Another issue is that there might be some words not considered by the algorithm as main topics, although including them could be useful for the student. Therefore, in such cases, it will be better to manually add them as links.

The second, is the guided usage scenario. It overcomes to some extent the issues described above. WikiGloss can be used for class activities and connected to an external e-Learning application such as Moodle. The teacher can select the document to be wikified, while having the option to revise the accuracy of the parsed document, and to decide the links that will be shown to the students, as well as including additional links. The activities can be reviewed first by the teacher to eliminate inaccuracies in the exercises. Hence, the WikiGloss application can help to facilitate the creation of glosses and assign activities based on the classroom material. Under this scenario, the extensive reading task is better supported. The teacher can use his or her expertise to decide which material is more suitable based on the student language level.

6.5 Schema of Applications in the Virtual-MVC architecture

The advantage of the implementation of Task Management and Wiki-Gloss under Virtual-MVC is that new applications can be assembled from services composition. The organization of the architecture in the case of these application is presented in Figure 6.9. The engines are organized within the service inventory. From Task management environment, The Problem Editor component application can make use part
6.6 Other areas for implementing the Virtual-MVC design pattern

Virtual-MVC framework can as well support other areas of application, besides E-Learning applications. Following Virtual-MVC design pattern, the services design is standardized independently of the server or different instance of the framework. Therefore, the services in any instance of the model can be reused by registering in the corresponding inventory. In this subsection we present two more scenarios where different instances of the framework are used for two different areas.

6.6.1 Visual Programming Services

The idea behind Movie-based programming is to create and support high-level language constructs using special symbols and images with semantic support to model dynamical features of the algorithms [12]. This visual representation of algorithms is
6. VIRTUAL MODEL-VIEW-CONTROLLER BASED APPLICATIONS

Figure 6.10: Problem Editor contextual help using Wiki-miner services

in demonstrating their features in the form of a movie, in which is possible to specify a behaviour in time to dynamic elements such as film, frames, control lines, structures and structure elements as well as computational formulas. It facilitates not only the programming task but also the debugging process.

6.6.1.1 Movie-Based Programming Environment (MMEPAD) architecture and reusability features

The main components of the stand-alone architecture of MMEPAD are shown in Fig 6.11. An MP-film is associated with an internal semantic representation or metadata, this metadata is stored in XML files called MP-templates. A code generator component processes the MP-templates generating the executable code and a movie animation based on a set of source code templates. The templates store snippets of code in different programming languages semantically associated with the visual objects [59]. With the current features, MMEPAD facilitates the development of complex algorithms and parallelization of tasks. The executable code produced in C language can be incorporated into external applications such as OpenMP platform, therefore reducing the development effort in such complex algorithms. However, MMEPAD is a standalone application and does not support collaboration and sharing of resources on a centralized repository, which could greatly enhance the reusability of MB-programs for collaborative work in such complex tasks. Therefore, it is necessary to create a repository organizing MP-films in the form of a library (knowledge base), where the developers can share MP-templates and have access to a common library and reuse the algorithms by importing them into their own MP-films, and in future expose them as a service which can be integrated on external applications.
The MMEPAD functionalities are extended under the Virtual-MVC pattern to support reuse of MP-templates containing the MB-programs and the executable code under a distributed environment [11], as explained in Fig 6.12. The stand-alone application is extended with the functions to import (download) and export (upload) the executable code and MP-templates to the library in the model. The users can also browse the library and add new categories. Once the MP-templates and executable code are stored in the model, they can be viewed independently of MMEPAD environment under a Web-based application. The independent version, supports library browsing functionalities (See Figure 6.13). When the user selects an algorithm from the library, the application shows the description, enhanced with automatic glossary, the display of the corresponding algorithm movie, and an execution monitor screen to show data values color coded. The user has the option to download the algorithm or create a new entry in the library. At the model the MP-films are organized within a library structure in the database, as well as the corresponding executable code. The business logic to process the MP-film library, and the execution functions are organized under the MB-Engine as endpoint services, which are registered in a service metadata table.

The WikiMiner Engine services previously described are reused in the Visual Programming application. The services are integrated to provide contextual help for algorithms descriptions. The glossing services are used to display a dynamic glossary using Wikipedia. The glossing services enhance the descriptions of mathematical terminology, formulas, etc. The development effort to reuse the Wiki-Miner services is at the view layer by selecting the necessary services. At the controller layer the request and response messages in XSLT for message exchange are designed. A a component embedded in MMEPAD to connect with the Controller via HTTP is developed.

![MMEPAD Architecture](image)

Figure 6.11: MMEPAD Architecture
The executable code of the MP-film uploaded in the model can be reused for processing the corresponding calculations and exposed as individual applications that can be accessed outside of MMEPAD as well. The variety of local and remote computational resources can be also implemented as a set of Engines. From the user’s point of view, it is a subset of endpoint services that encompass functionalities of an API and realize processing that is specific to an application. This makes possible to provide the flexibility of available computational resources, supporting the execution of MP-programs on various programming platforms. Additionally, the integration with Wiki-Miner Engine helps the users to perform a semi-automatic generation of glossaries for algorithm descriptions.

### 6.6.2 Distributed Tsunami Modelling Services

The Virtual-MVC methodology is also being adopted for the implementation of Distributed Tsunami calculation services in a different track of research [25]. The implementation of Tsunami calculation services is out of the scope of this thesis work, and it is carried out by a different research group. However, a brief outline on how the methodology is adopted is provided as follows.

The Tsunami Modelling System components are developed as Application Engines in the model. Three main application engines reflect all stages of modelling process.
Movie-based Programming Environment

Figure 6.13: Movie-based Web based GUI in the view
(tsunami wave propagation, inundation and impaction). It supports the typical working scenario including the following steps:

- Specifying information on the modelling scenario including bottom topography or bathymetry data, initial and boundary conditions, information about earthquake data, modelling parameters such as time-steps and length of model runs, etc.
- Defining a computational resource (engine) taking into account the scenario parameters.
- Launching a modelling engine, and monitoring modelling process with on-line storing of intermediate and final results.
- Viewing/analyzing/visualizing results in human-centred representation.

Every step is supported by a corresponding component in the MODEL part. Agent-based Manager of Modelling Engines analyzes the request, selects a corresponding calculator (computer) of Tsunami Modelling Calculation, and monitors a modelling process. It is realized by means of an agent-semantic system, designed for resource management and adopting to the tsunami modelling research. The Impaction Engine is to model impaction taking into account the tsunami wave parameters and physical characteristics of objects (artificial or natural) such as houses, bridges, etc. Tsunami Visualizing Engine (TVE) is a converter of digital results of modelling to the human-centred data representation. Tsunami data loader downloads the bathymetry data, tsunami scenario and modelling results from the Internet and store in the database.

The special component TVE allows to transform the digital results of modelling to the human-centred data representation. In the view, a special editor can be provided to access the calculation data, allowing the user to change the bathymetry data by including and manipulating artificial objects of variable placement, shape and sizes. The advantage of using Virtual-MVC pattern is that the functionality of the application can be exposed as services supporting components in a distributed environment, scalability of resources and reusability.

6.7 Chapter Summary

The Virtual-MVC framework aims to contribute to moving forward the implementation of third generation of E-Learning platforms. As discussed in the chapter, the Virtual-MVC framework supports a SOA environment where e-learning applications can be assembled from available services in the model. To demonstrate the advantage of using Virtual-MVC, we presented the Task Management environment and the WikiGloss tool for reading comprehension tasks in Language Learning activities.

The applications are designed since the first stage as a set of components, and logically organized in the form of Engines. Additional APIs for specialized processing can be incorporated in the model and exposed as reusable services, such as the case of the Wikipedia Miner API. The service inventory is realized by registering the
services metadata in the model. The applications are assembled by developing the corresponding graphical interfaces in the View. However, the services do not keep internal dependencies for composition, instead are assembled using Service Virtualization implemented in the controller facilitating the reusability in different compositions, and a variety of applications. Such example is the use of part of the services in Wiki-Miner Engine in the Problem Editor application, as well as in the Visual programming environment.

The Virtual-MVC Design pattern support SOA, which in important approach to implement Cloud based environments. The applications presented are suitable for Cloud environments, presenting the possibilities of enabling E-Learning Cloud environments. However, it is not limited to E-Learning applications, as the pattern is independent of a particular application area. We presented additional examples in the development of other areas of application, such as Distributed Tsunami calculation services, and Visual Programming.
Chapter 7
Analysis and Evaluation Studies

Overview. The focus of this chapter is on the evaluation and analysis of the Virtual MVC design pattern. The virtual VMC components roles were analyzed based on the MVC and related patterns. A study was performed in a Software Engineering course at the University of Aizu, where services were designed following the Virtual-MVC programming approach and framework, obtaining positive results on development complexity reduction. The programming approach and framework are also evaluated based on the reduction of integration code as complexity parameter. Finally a qualitative comparison corresponding with the services design based on Service Component Architecture is explained in detail.

7.1 Evaluation Study

Along the Virtual-MVC methodology design, a number of twenty students have participated on incremental evaluations. Half of the group were master students, and the other half undergraduate students. Such evaluations have been used to improve the Virtual-MVC methodology and programming approach. The students have participated in different times and groups. In this section we explain the final qualitative study that has been undertaken after the methodology and programming approach has been established. The purpose of the presented study is to analyze the developer’s perception of complexity using the Virtual-MVC methodology. We also analyzed whether this perception is affected by developer’s skill level. We considered as well perception of time required for learning the pattern, and start working with the platform. The subjects were six graduate students from the University of Aizu. The methodology was taught as part of a Software Engineering course.

Procedure. The students were taught the pattern and the programming steps during the class. The overall time was 3 hours of explanation, 3 hours of practice, and 1.5 hours of consultation. The time was divided into 5 classes of 90 minutes each. Finally, the classes were distributed over a 2 week time frame. They were given a class assignment where they have to develop an application. The level of difficulty was similar for all the students, considering the time limit during the course period. They were requested to create and compose 3 services using the Virtual-MVC methodology. At the end of the course, they were provided with a survey to evaluate the development experience. The
individual programming expertise of the students was variable, as it can be expected in a normal course environment. The students were asked to rate their expertise on Web application development, and Web services (Figure 7.1). In Figure 7.1.a, the evaluation shows that 4 of the students considered having strong expertise with Web applications (67%). However, not all of them have the same expertise working with Web services. When they were asked to rate themselves on their expertise developing Web services, only one of them consider having strong expertise, and 50% of the students considered as having fair expertise.

Since our interest in on the perceived complexity on Web services, the responses are analyzed based on their own evaluation of expertise developing Web services. We classified the students in three groups, based their Web services expertise: Poor or novice (2 students), Fair (3 students), Strong (1) (see Figure 7.1.b). The programming languages they have expertise on were: Java, JSP, PHP, Python and JavaScript, and knowledge on XML standards. Only the student with fair expertise was familiar with XSLT, the rest learned by themselves during the course.

The analysis is divided in three parts. The first part is concerning to the perception of time spend learning the methodology and framework, as well as developing the project. The second part deals with the difficulty perception taking in consideration the number of components, understanding relations and data flow, the development process, debugging and maintenance. The third part aims to understand the perception of the new methodology compared with their knowledge and experience developing Web services. The corresponding question number is represented with the letter Q following by the number in the survey.

### 7.1.1 Analysis of Time perception

The evaluation about the time perception is based in two questions.

- **Q3.** Mention the programming languages that you normally use for Web development (Open answer).

- **Q5.** What is the most appropriate time for you in order to learn the pattern and development steps? (Answers: 1 week or fewer, 2 weeks, 3 weeks, 4 weeks, 5 weeks or more).
7. ANALYSIS AND EVALUATION STUDIES

Figure 7.2: Student’s profiles and time evaluation

- **Q10.** Do you consider the time required to develop services is appropriate? (Answers: Strongly disagree, Disagree, Neutral, Agree, Strongly Agree).

Figure 7.2 shows the student’s answers for this analysis. As it was explained previously, the class was distributed in a period of 2 weeks with a total of 7.5 hours divided along the time of the classes. In the group of fair expertise, one student suggested to increase the time learning the methodology along a 5 weeks time or more. However, this answer can be due to his different programming expertise. In the other two groups, students also suggested to increase the learning time, including the student with strong expertise. Taking in consideration that the increase of time is suggested in all of the groups, in future sessions it will be extended. In a different course setting such as workshop or seminar, the teaching could have been schedule in a day. Therefore, the increase of time if its considered in a total amount of hours might not exceed 15 hours, which is still considerable small for learning a new methodology and start developing applications.

The evaluation of development time was quite positive in all of the groups. The student response helps to clear the concerns such as if their perceive the new methodology as time consuming due to the amount of components and relations. The answers that are more relevant in such as case, are from the fair and strong expertise groups, because novice developers they might not have a strong background to compare. However, the answer was still positive.

7.1.2 Analysis of development complexity perception

In section 5, we explained the components that were required to develop an application using Virtual-MVC. The objective is to know based on the student’s experience using the Virtual-MVC methodology, what is their perception of complexity developing applications. The students were asked about the difficulty of steps, difficulty of developing components, the development task, debugging the application, tasks related to application maintenance. The last evaluation is if they were able to implement the project using the Virtual-MVC framework. The questions for this evaluation are described as follows.
Q7. Do you consider that is difficult to develop Service-Oriented applications using Virtual-MVC? (Answers: Strongly disagree, Disagree, Neutral, Agree, Strongly Agree).

Q9. Does the number of steps required to implement a service makes the implementation complex? (Answers: Strongly disagree, Disagree, Neutral, Agree, Strongly Agree).

Q11. Can you specify the most difficult component needed to develop the application? Put a number from 1 (easier) to 3 (most difficult) to evaluate the level of difficulty.

Q12. Would you consider easy to understand the relations between model, view and controller components, in order to implement a Web service? (Answers: Strongly disagree, Disagree, Neutral, Agree, Strongly Agree).

Q13. From 1 (easier) to 5 (most difficult), rate the difficulty of debugging your application.

Q14. Once you have developed the services, if you need to modify the function in the model, and update the correspondent views and controllers, do you think this will be a complex task compared with the same tasks in traditional web applications (JSP, ASP.NET, PHP, etc)? (Answers: Strongly disagree, Disagree, Neutral, Agree, Strongly Agree).

Q15. Once you have developed the services, if you need to add a new service in the sequence, do you think this will be a complex task compared with the same task in traditional web applications (JSP, ASP.NET, PHP, etc)? (Answers: Strongly disagree, Disagree, Neutral, Agree, Strongly Agree).

The result of this evaluation is presented in Figure 7.3. According with the responses, most of the students in all groups rated as not being difficult to develop applications (83%). However, one student in the group of fair expertise responded as being difficult to develop applications. One factor that could influence this answer is that the expertise of this student (as mentioned in previous analysis) is Phyton programming language, and not strong expertise with Java. The novice students, as well as the expert student both perceived as not difficult to develop applications.
The number of steps to develop an application is a point of concern to evaluate complexity. The component where most effort is done is the model, since each service is designed by four classes to implement the endpoint (Interface, RequestValueObject, ResponseValueObject, ServiceEndpoint). The result shows that in general most of the students considered that the number of steps required to develop the applications didn’t add to the complexity. Similarly, the perception of the simplification of relationships between components, has a positive result in the three groups (Q12). As the answer was positive and consistent in all of the groups, we can preliminary conclude that as long the relationships of the components are easy to understand, few additional steps do not add to complexity perception. The students were asked to rate the difficulty of each component development. They considered that Model was the most difficult component to develop, and the view was the easiest. The answer was somewhat expected as the model holds the business logic, most of the coding task developing services is devoted to the model component.

One of the shortcomings of the framework is the debugging process (Q13), currently the framework provides a error log files using WebSphere server. The group of novice students and fair expertise rated the debugging process similarly as being difficult. The student with strong expertise consider easier the debugging. We expected this result because the current programming environment has no special tools for debugging the model. Eclipse was used for development; however it supports only the debugging of syntax errors, while the functionality had to be tested at runtime, and the errors were generated in log files. Further development of the programming environment is required to overcome this issue.

The difficulty of maintenance was evaluated by asking the students of their perception updating or adding more services to the application (Q14), and redesigning the composition (Q15). For the novice students the perceptions are different. One of the students considered not difficult the tasks. This student has knowledge of JSP which is based on Java. Since the Virtual-MVC framework is based on Java, the perception could have also been related with his expertise. In the second case, the student is use to work with PHP. The answers rate the maintenance as difficult. For the students with fair knowledge, the answers were more neutral when asked the difficulty of modifying the services, but for application maintenance, two of them perceived the task difficult, and one of them which has the strongest background in Java perceived the task as very easy. The student with strong expertise was neutral with the task of service update, and rated as easy the modification task.

The final evaluation was the project implementation. All students in novice and expert group implemented the project. However, in the group of fair expertise the student with different programming background could not complete the task. The result leads to think that such result is more due to the programming background of the student.

The final part of the evaluation is concerned with the student’s comparison with the Virtual-MVC platform, and their Web services development experience. Therefore, the answers only were asked to the group of fair and strong expertise. Novice students were
not asked to answer the related questions since they have no reference for comparison. The questions asked to the students are as follows.

- **Q16.** From your experience working with Web services, do you consider that Virtual-MVC facilitates the development? (Answers: Strongly disagree, Disagree, Neutral, Agree, Strongly Agree).

- **Q17.** Compared with your usual approach developing and integrating Web services, mention what do you think are advantages of using Virtual-MVC platform?

- **Q18.** Compared with your usual approach developing and integrating Web services, mention what do you think are disadvantages of using Virtual-MVC platform?

Figure 7.4 shows the student’s answers. In the group of fair expertise, two of the students considered that the platform facilitates Web service development. One of them considered that there was not difference, as it rated neutral. The advantages that were considered by the students are related with the simplification of the relationships between the components. Which is one of the objectives of loose coupling of view and model in our approach. Such loose coupling allows that view and model can be developed independently. Therefore, the developer’s can concentrate on the corresponding design. The result of this question is also related with previous analysis of Q12. The disadvantages mentioned by the group were related with debugging, and mapping view and controller fields. Improvements in the controller are the automatic template generation based on the model RequestValueObject. When the template is partially generated the developer needs to care of adding his/her input parameters from the view facilitating the task. The comment related to the network environment is due to the development setting. The students were working directly with the server and not need to install special software in their computers to simplify the task. The evaluation from the student with strong expertise is neutral. The advantage of simplification of component relations is also highlighted, and suggested is good for beginners. The disadvantage pointed out is related with the service composition style which is based
on message itinerary. Currently we don’t manage visually the workflow, but is a point to explore for future research. The summary of all groups results is presented in Figure 7.5.

7.2 Degree of Service-Orientation

Finally, we summarize how Virtual-MVC supports SOA design principles [17] enabling applications with native reusability and interoperability, as well as facilitating the integration task.

- **Standardize Service Contract.** The service contract is managed by the controller, using the Repository and Registry. The functional expressions, and data model are standardized according to the registered endpoints. The programmer does not need to specify the service contract at development time.

- **Service Loose Coupling.** Services in the Engines are designed to be as independent as possible, and do not contain direct calls or references to other services in the data model. This is due to service virtualization managed by the controller.

- **Service Reusability.** In the Virtual-MVC, the services registered in the inventory have no knowledge of other services components, only through the controller. As they are independent pieces of code, they can be reused in different applications as long as they are registered in the service inventory.

- **Service Composability.** Reusable and loose-coupled services are effective participants in compositions for Virtual-MVC and external applications. The compositions are handled by the controller, and follows an itinerary based approach.

- **Service Discoverability.** The controller contains the mechanisms to interpret the meta-data description of the services in the model; therefore they can be effectively discovered. The current Virtual-MVC framework implementation does not support automatic service discovery, which is considered for future extension.

- **Service Abstraction.** Services in the model follows abstraction by exposing its capabilities only to the controller. The view has no knowledge of the model outside the Service ID. It only has access through the controller.

- **Service Autonomy.** Each service endpoint controls their runtime execution environment.

- **Service Statelessness.** The controller manages the information state, minimizing this task from the services in the model.
Figure 7.5: Summary of developer’s evaluation of Virtual-MVC and programming strategy
7.3 Qualitative comparison of MVC roles

We compare the Virtual-MVC components based on the functionalities we classified for the related MVC patterns in section 3 (See table 7.1). In the Virtual-MVC the view is hosted as a Web container and it handles both view state and logic. In principle, the view can be developed independently and include richer programming resources such as jQuery, AJAX, JavaScript, etc. The controller keeps the concept of an intermediate layer, however it extends from the traditional functionality of handling user interactions to Integration layer for Service-Orientation comprising functionality for data transformation, service virtualization, intelligent routing, orchestration, security, etc., abstracting these complex tasks from the developer. The model is itself designed for SOA, the processing logic in the model is organized and standardized in the Service Inventory, facilitating services re-usability for new compositions. Loose coupling is one of the main principles and a desirable characteristic of the services in SOA, as is expected if the components are decoupled and then independent testing becomes necessary for the business logic.
<table>
<thead>
<tr>
<th>Design Pattern</th>
<th>View</th>
<th>Controller</th>
<th>Model</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model-View-Controller</td>
<td>View state</td>
<td>Input and user interaction Data</td>
<td>Processing Logic</td>
<td>Loose coupled.</td>
</tr>
<tr>
<td>(MVC)</td>
<td>View logic</td>
<td>Data</td>
<td>Debugging complexity.</td>
<td>Difficulties handling view logic.</td>
</tr>
<tr>
<td>Presentation-Model (PM)</td>
<td>View state</td>
<td>Intermediate layer View logic Part of model</td>
<td>Processing Logic</td>
<td>Loose coupled.</td>
</tr>
<tr>
<td></td>
<td>View logic</td>
<td>Data</td>
<td>Data</td>
<td>Development complexity (GUI). View depends on synchronization mechanisms.</td>
</tr>
<tr>
<td>Model-View-ViewModel (MVVM)</td>
<td>View state</td>
<td>Intermediate layer</td>
<td>Processing Logic</td>
<td>Performance affected by excessive data binding</td>
</tr>
<tr>
<td></td>
<td>View logic</td>
<td>View logic</td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td>Model-View-Presenter (MVP)</td>
<td>View state</td>
<td>Complex view logic</td>
<td>Processing Logic</td>
<td>Loose coupled View requires additional testing</td>
</tr>
<tr>
<td>Supervising controller</td>
<td>Simple view logic</td>
<td>Data</td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td>Model-View-Presenter (MVP)</td>
<td>View state</td>
<td>Intermediate layer View logic Part of view</td>
<td>Processing Logic</td>
<td>View requires additional testing</td>
</tr>
<tr>
<td>Passive view</td>
<td></td>
<td>Data</td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td>Virtual-MVC</td>
<td>View state</td>
<td>Intermediate layer Integration Orchestration Data</td>
<td>Processing Logic (Service Inventory), Data Model</td>
<td>Due to the loose coupling, independent testing is necessary at the model</td>
</tr>
</tbody>
</table>
7. Analysis and Evaluation Studies

7.4 Analysis of related MVC Design Patterns and programming approaches

In this section we discuss the most relevant MVC related patterns, analyze the roles of the components, and their limitations. Table 7.1 presents a summary of the patterns. We used Fowler’s approach [21] to classify the view according to the two main functionalities. The view state, where the view is used to present the data output. The view logic, in which additional logic is required for specialized visualization of the content. For example, color coding of the data, or using special figures to represent the output.

We compare Virtual-MVC and related MVC patterns components based on the previous classification of functionalities (See table 7.1). In the Virtual-MVC the view is hosted as a Web container and it handles both view state and logic. In principle, the view can be developed independently and include richer programming resources such as jQuery, AJAX, JavaScript, etc. The controller keeps the concept of the intermediate layer, but it extends from the traditional functionality of handling user interactions to an integration layer for Service-Orientation. It comprises functionality for data transformation, service virtualization, intelligent routing, orchestration, security, etc., abstracting these complex tasks from the developer. The model is itself designed for SOA, the processing logic in the model is organized and standardized in the Service Inventory, facilitating services re-usability for new compositions. Loose coupling is one of the main principles. As it is expected if the components are decoupled; independent testing becomes necessary for the business logic.

7.5 MVC programming approaches

The previous classification shows the conceptual comparison of the MVC design patterns. To quantify a sample of the coding effort following the programming approach, we analyze two main frameworks: Google Web Toolkit (GWT)\(^1\) for MVP implementation and Spring\(^2\) for MVVM pattern. The Problem Editor application described in Section 6.1.2, is used for the comparison analysis. The operations implemented are to show a list of problems (catalog), add a new problem, edit and delete an existing problem (See Table 5.1). The main implementation language for all of the frameworks is Java. For each of the platforms, the application is analyzed by components, taking in consideration the number of elements corresponding to Java classes, XML, XSLT, JSP or HTML files that are necessary to implement the basic functionality. Part of the programmers effort associated with complexity of development, is the amount of code corresponding to component’s integration. In MVC patterns the integration code is in the controller. Based on this assumption, we define an index of Integration code (\(Ic\)), calculated as the ratio of controller elements divided by the total elements of the application. The formula is as follows:

---

\(^1\)http://www.gwtproject.org/

\(^2\)http://spring.io/
First, the applications are analyzed based on the amount of elements which includes Java classes and other files with XML content, HTML, JSP, etc., (Table 7.2). The second analysis (Table 7.3) is based on the lines of code (LOC) of each element (excluding blank and comments lines). At the model, the development of Virtual-MVC consists in 5 service endpoints, with its corresponding interface, Request and Response value object java classes. The class `problem` contains the data structure of the problem. The model in Spring consists of 6 elements implemented as Plain Old Java Objects (POJOs). Two problem classes are designed, the first contains the problem structure, and the second follows the input form in the view to get the problem data. There is only one service endpoint that takes care of the problem operations (add, edit, delete, retrieve). Similarly in GWT, there is only one service endpoint. Virtual-MVC has the highest number of elements in the model (Java classes), this is due to the service structure, and the atomic nature of the functions. However, Virtual-MVC and Spring MVC have similar LOC number. This is because both implement the business logic in the model. On the contrary, GWT using MVP pattern implements part of the logic in the Controller using the Presenter, therefore the code in the model is lower for similar functionality.

The controller performs the integration functions. In Virtual-MVC the programming consist on the XSLT request and response elements, mapping the data to the corresponding service. Using the Virtual-MVC framework, the programmer do not deal with writing an algorithm, and the associated lines of code are data description using XML and XSLT. 81.3% of the code written in the controller corresponds to the response elements, that contains the layout to be displayed at the view (HTML code). Therefore, no algorithm is being implemented by the programmer. Moreover, one of the advantages of using XSLT in the controller is that it support implementation language independency. On the contrary, the other platforms specific implementation is on Java.

Spring uses Java classes to implement the controllers managing the problem editor service, which accounts to 80.1% of the LOC. The configuration and mapping of the elements is stored in the XML elements. In GWT, the controller elements are implemented as Java classes. The application controller contains the history management and view transition logic to coordinate the application. The presenters drive the view and handle events from the widgets. An event is created for each operation. The corresponding event and handler are designed. In GWT the amount elements necessary for the integration in the controller is the highest. However, the relation between the elements is more clear than the integration code using Spring.

\[ Ic = \frac{controller}{model + view + controller} \times 100\% \]

\[ \text{.........................(1)} \]
Table 7.2: Analysis of related Model-View-Controller programming approaches

<table>
<thead>
<tr>
<th>Component</th>
<th>Virtual-MVC</th>
<th>Spring (MVC)</th>
<th>GWT (MVP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Java classes</td>
<td>Java classes</td>
<td>Java classes</td>
</tr>
<tr>
<td></td>
<td>Interface (5)</td>
<td>Interface (1)</td>
<td>Interface(1)</td>
</tr>
<tr>
<td></td>
<td>Services (5)</td>
<td>Services (1)</td>
<td>Services (1)</td>
</tr>
<tr>
<td></td>
<td>RequestValue-Object (5)</td>
<td>Service Test (2)</td>
<td>RPC service (1)</td>
</tr>
<tr>
<td></td>
<td>ResponseValue-Object (5)</td>
<td>Problem (2)</td>
<td>Problem (1)</td>
</tr>
<tr>
<td>Elements</td>
<td>21</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Controller</td>
<td>XSLT</td>
<td>Java classes</td>
<td>Java classes</td>
</tr>
<tr>
<td></td>
<td>Request (5)</td>
<td>Controllers (2)</td>
<td>Application-Controller (1)</td>
</tr>
<tr>
<td></td>
<td>Response (5)</td>
<td>Controller test (1)</td>
<td>Presenters (3)</td>
</tr>
<tr>
<td></td>
<td>XML</td>
<td>SpringApp-servlet (1)</td>
<td>Events (4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Message Properties (1)</td>
<td>Events handlers (4)</td>
</tr>
<tr>
<td>Elements</td>
<td>10</td>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>View</td>
<td>HTML/XSLT</td>
<td>HTML/JSP</td>
<td>Java classes/HTML</td>
</tr>
<tr>
<td></td>
<td>View (4)</td>
<td>View (3)</td>
<td>View (3)</td>
</tr>
<tr>
<td>Elements</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Integration code</td>
<td>29%</td>
<td>36%</td>
<td>63%</td>
</tr>
</tbody>
</table>

Although the number of elements and LOC metric in Virtual-MVC is the highest, the coding is not complex due to the following: the relations between the components are conceptually clear. This can be reflected in the coding style. The service logic and objects are at the model layer, all java code implementation is at the model. The controller handles the transformation, which is basically the service mapping (input/output data). There is no algorithm implemented at the controller. The view handles only the presentation which in the case of the example is in HTML. The development of the components is based on templates, and the controller can be semi-automated to facilitate the data mapping.

The efforts and complexity explained are consistent with the programmer’s evaluation (Fig.7.5.d and e). This can also be reflected in the calculations of Integration code index based on LOC. Although is not the lowest index (49.2%) compared with Spring and GWT, the efforts developing the algorithms are lesser, which is reflected in the percentage of java code implemented (25.8%) to construct the application. Moreover, in Virtual-MVC the index of Integration code based on elements is the lowest (29%) compared with the necessary elements in Spring and GWT.
Table 7.3: Programming effort analysis

<table>
<thead>
<tr>
<th>Component</th>
<th>Virtual-MVC</th>
<th>Spring (MVC)</th>
<th>GWT (MVP)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lines of Code</td>
<td>373</td>
<td>375</td>
<td>146</td>
</tr>
<tr>
<td>Java (%)</td>
<td>100.0%</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td><strong>Controller</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lines of Code</td>
<td>712</td>
<td>306</td>
<td>382</td>
</tr>
<tr>
<td>Java (%)</td>
<td>0.0%</td>
<td>80.1%</td>
<td>100.0%</td>
</tr>
<tr>
<td>XML Standards, HTML (%)</td>
<td>100.0%</td>
<td>19.9%</td>
<td>0.0%</td>
</tr>
<tr>
<td><strong>View</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lines of Code</td>
<td>363</td>
<td>106</td>
<td>218</td>
</tr>
<tr>
<td>Java (%)</td>
<td>0.0%</td>
<td>0.0%</td>
<td>94.0%</td>
</tr>
<tr>
<td>JSP (%)</td>
<td>0.0%</td>
<td>100.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td>XML Standards, HTML, others (%)</td>
<td>100.0%</td>
<td>0.0%</td>
<td>6.0%</td>
</tr>
<tr>
<td><strong>Total Lines of Code</strong></td>
<td>1448</td>
<td>787</td>
<td>746</td>
</tr>
<tr>
<td><strong>Java Code (%)</strong></td>
<td>25.8%</td>
<td>78.8%</td>
<td>98.3%</td>
</tr>
<tr>
<td><strong>Integration code</strong></td>
<td>49.2%</td>
<td>38.9%</td>
<td>51.2%</td>
</tr>
</tbody>
</table>

7.5.1 Comparison with Service Component Architecture

The reduction of IT complexity and the simplification of development, deployment and management of enterprise applications, is vital for SOA implementation. Service Component Architecture (SCA) is a standard, and programming model that has been proposed and is supported by big SOA players such as IBM, Oracle, SAP. It has been standardized by OASIS group [53] [8]. The objective of SCA, is to reduce complexity through a standardize framework to assemble disparate SOA components into a higher level composite [8]. The basic artifact of SCA is called component. The structure of the component consists of the business function (business logic) that is called implementation. The implementation depends and can be configured by settable properties to define the data values, and declares its dependencies with other components in the references. Compositions are defined in a composite unit, containing the definition and properties of components that provide an atomic portion of the business functionality (see section 2.1. In the Virtual-MVC model, the atomic unit is the endpoint service, which is similar to the SCA component concept. However, in Virtual-MVC the data values or input/output parameters are not declared in the business logic as is the case of the parameters in SCA. They are mapped through the Request and Response value objects.

The second difference in the model, is that in Virtual-MVC it does not contain external dependencies within the business logic, as is the case with SCA. The service is decoupled of any hard coded dependencies by introducing DI in the controller. All
references to other services in a composition are handled by the controller. The service
does not keep a dependency among the other services in the domain. Hence, it minimize
the need to modify the business logic for new compositions. The controller manages the
composition workflow following an itinerary based message routing. SCA encourages
the use of introspection to obtain information about component type, properties, and
references. If introspection is not available, the metadata is handled using a component
type side file in XML [8]. Virtual-MVC has a similar approach on separating the
metadata of component properties. However, this metadata is handled outside of the
service itself, and registered in the Service Repository and Registry metadata table
which is managed by the enhanced controller. Therefore, Virtual-MVC focus on service
reusability by enforcing more agnostic logic.

Regarding the coupling with the view, both approaches decouple the model from
the view. SCA uses the composite to abstract the services properties and implement-
tation details from the view. Virtual-MVC abstracts the model reference through the
controller, the view knows only the service ID passed to the controller. Therefore, any
change in the service endpoint at the model does not affect the view. In SCA, mod-
ifications are required in the composite unit, and in Virtual-MVC, the modifications
are managed in the controller XSLT files. Both of the approaches target complexity
reduction. SCA approach main advantage is to focus on standardize services compo-
nents and the simplification of services compositions, where the programming approach
allows to manage different granularity of composites. Virtual-MVC main focus is re-
ducing complexity, while focusing on service decoupling and reusability of the business
logic, handling the compositions at a runtime.

SCA is adopted in several important platforms such as Tuscany, TIBCO Active-
Matrix, and SOA Suite 11g (Oracle). These tools provide a good set of visual tools
to facilitate the programming tasks. In the case of Virtual-MVC visual tools need
further development. However, as the platform is deployed in WebSphere server, the
programmer can import to Eclipse the project libraries and design the service based
on a template. It is not necessary to recompile the project but register the new service
in the metadata table, and save the code in the server location. Therefore, there is no
much hurdle in development event visual tools are not in place yet.

7.6 Chapter Summary

The study was undertaken by assigning a programming project to students in a Software
Engineering course. The findings of the evaluation demonstrated that the application of
the Virtual-MVC pattern was understandable even for novice programmers. Students
were able to implement the project with their current skills in Java and HTML pro-
gramming. The cumulative time studying the pattern took less than a day. Although
their application is factored in MVC components, it was not difficult to understand the
relations and the data flow among them, which is sometimes difficult with the MVC
pattern and related versions. Finally, their complexity reduction evaluation was posi-
tive, achieving the pattern goal. By following the pattern the programming framework
supports a structured approach, and SOA principles.

The Virtual MVC roles were analyzed comparing with related design patterns. The results summarized in Table 7.1, describe the major changes introduced to MVC extending the controller to perform functionality of an ESB, and the model which additionally to data, holds the processing logic in the form of service inventory. As the view and model components are decoupled, is necessary the independent testing is performed in the model. The second analysis based on MVC pattern, compares Virtual MVC with related Web based programming approaches for MVC and MVP which are the more used for Java programming. An Integration code index is defined and used to measure the amount of components that are required to build an application. Although the qualitative analysis may lead to think that the controller development might require more coding efforts, the index is the lowest as 29%, followed by Spring MVC. The factors that influence this result are two. First, most of the integration effort is abstracted from the developer in the framework’s controller. Second, the framework is realized to facilitate the pattern’s implementation. Therefore, is expected that Spring framework has also lower integration index, as the MVC model is facilitated in the platform design. Finally, although Virtual-MVC requires more components developed in the model, this do not affect the complexity since the classes do not need to keep hard coded references among the service classes. This is confirmed by the evaluation study where 83% of students disagree or were neutral to the increase of complexity besides the amount of steps (Figure 7.5.e), as well as 83% of them considered easy to understand the relations between the components (Figure 7.5.f).

The comparison with SCA model shows two main differences. First, in the Virtual-MVC model, the atomic unit is the endpoint service, which is similar to the SCA component concept. However, in Virtual-MVC the data values or input/output parameters are not declared in the business logic as is the case of the parameters in SCA. They are mapped through the Request and Response value objects. The second difference in the model, is that in Virtual-MVC it does not contain external dependencies within the business logic, as is the case with SCA. The service is decoupled of any hard coded dependencies by introducing DI in the controller. All references to other services in a composition are handled by the controller. The service does not keep a dependency among the other services in the domain. Hence, it minimize the need to modify the business logic for new compositions and the complexity to keep the relationships among the services.
Chapter 8

Summary and Conclusions

This research study focused to reduce the development complexity of SOA based applications. In the first part of the study, a literature review is undertaken to identify the implementation complexity challenges of SOA solutions. SOA platforms follow a layered approach to organize the functionality to support such complex interactions in Enterprise Application Integration. Therefore, a set of diverse tools and technologies are involved. Therefore, different programming and design strategies need to be combined to enable a flexible architecture, adding a toll on the implementation complexity. Among the main approaches to decrease such implementation complexity are the introduction of middleware such as the ESB, and the Service Virtualization strategy. In the ESB the physical destination and connection information is abstracted from the service, and can be setup at runtime. This characteristic improves the capabilities of traditional tightly coupled and distributed software components. Using service virtualization, the service attributes related with integration can be reconfigured instead of changing the service code. The main benefits of service virtualization are code reduction, greater reuse and flexibility. Both of the approaches extract low level details from the services that can be configured externally and injected at runtime. Ideally this functionality should be facilitated by the infrastructure. Service Component Architecture, focus on reducing complexity for the service design and composition, providing a standardize framework to assemble disparate SOA components into higher level composites.

A set of SOA metrics were studied to identify which factors influence SOA development and implementation complexity. Structural complexity can be measure at design level based on the messages and operations described in the interface. Therefore, reducing complexity on services interface can have a positive impact. Additional layer introduced to increase flexibility of the solution such as using virtualization layers, are easier to maintain but more difficult to setup increasing complexity. However, the increase on complexity is not an indicator of poor design, unless flexibility achieved is low. From the implementation side complexity indicators are related to the number of products involved, the number of steps and time needed to accomplish an operation, the complexity of the task itself, and the specialization level required. Based on the preliminary analysis, the focus is on reducing complexity of SOA implementation, by simplifying the integration workload from the programmer. A key aspect for the simplification is decoupling the components to facilitate reusability, and reduce dependencies.
Development platforms that enforce SOA design principles can have a positive impact decreasing the implementation complexity.

An important contribution of the MVC pattern is reusability of software components due to organization of functionality based on the roles of model, view, and controller. However, the related patterns, enhance reusability of software components under an object-oriented paradigm, but have limitations supporting service-oriented computing. Although MVC achieves higher reusability, it introduces certain complexity for simple applications due to the additional integration coding for the controllers.

The proposed approach, is on extending MVC design pattern main concepts of reusability and loose coupling for SOA. The proposed pattern is called Virtual Model-View-Controller, it facilitates the development of service-oriented applications by incorporates the design concepts of MVC and enhance the applicability for SOA, addressing the design of services, flow of information, and components to realize such services. The given study proposes a standardized terminology for a more structured comparison between MVC related patterns.

The proposed Virtual-MVC pattern supports Service-Orientiation, comprising in the controller the integration functionality of data transformation, service virtualization, intelligent routing, orchestration, and security. The ESB controller is enhanced with the DI pattern for services composition. It removes the hard-coded dependency of the services collaborator’s lookup. In the Orchestration Engine, the DI pattern is induced to form a connectivity channel for processing engines that choreograph the flow of activities between loosely coupled services. The controller abstracts from the developer’s efforts the implementation of data transformation, protocol and service interface definition, related to the service contract. It reduces the complexity of implementation, and allows the developer to concentrate on the business logic, reducing the development time.

To validate Virtual-MVC pattern, a programming framework was constructed, and it’s corresponding programming approach. The example presented are developed for E-Learning applications, which can benefit of adopting a SOA development approach. A study was undertaken by assigning a programming project to students in a Software Engineering course. The findings of the evaluation demonstrated that the application of the Virtual-MVC pattern was understandable even for novice programmers. Students were able to implement the project with their current skills in programming. The cumulative time studying the pattern took less than a day. Although their application is factored in MVC components, it was not difficult to understand the relations and the data flow among them, which is sometimes difficult with the MVC pattern and related versions. Finally, their complexity reduction evaluation was positive, achieving the pattern goal. By following the pattern the programming framework supports a structured approach, and SOA principles.

The Virtual MVC roles were analyzed comparing with related design patterns. The major changes introduced to MVC extending the controller to perform functionality of an ESB, and the model which additionally to data, holds the processing logic in the form of service inventory. As the view and model components are decoupled, is necessary the independent testing is performed in the model. A comparison of Virtual MVC with
related Web based programming approaches MVC and MVP was performed. Those were selected because are the more used for Java programming. An Integration code index was defined and used to measure the amount of components that are required to build to realize the application. Although the qualitative analysis may lead to think that the controller development might require more coding efforts, the result of the index calculation is the lowest as 29%, followed by Spring MVC. The main factors influencing this result are: first, most of the integration effort is abstracted from the developer in the framework’s controller. And second, the framework is realized to facilitate the pattern’s implementation. Therefore, is expected that Spring framework has also lower integration index, as the MVC model is facilitated in the platform design. Finally, although Virtual-MVC requires more components developed in the model, this do not affect the complexity since the classes do not need to keep hard coded references among the service classes. This is confirmed by the evaluation study where 83% of students disagree or were neutral to the increase of complexity besides the amount of steps, as well as 83% of them considered easy to understand the relations between the components.

The comparison with SCA model shows two main differences. First, in the Virtual-MVC model, the atomic unit is the endpoint service, which is similar to the SCA component concept. However, in Virtual-MVC the data values or input/output parameters are not declared in the business logic as is the case of the parameters in SCA. They are mapped through the Request and Response value objects. The second difference in the model, is that in Virtual-MVC it does not contain external dependencies within the business logic, as is the case with SCA. The service is decoupled of any hard coded dependencies by introducing DI in the controller. All references to other services in a composition are handled by the controller. The service does not keep a dependency among the other services in the domain. Hence, it minimize the need to modify the business logic for new compositions and the complexity to keep the relationships among the services.

The Virtual-MVC programming approach and framework is adopted in a research team environment. The framework is used by students in the Active Knowledge Engineering Laboratory to construct their applications based on services. Therefore, the code developed can be reused by new students in new composition, or extend the applications with additional services. An E-learning computational cloud environment is being designed to expose e-learning services that can be used in LMS such as Moodle. The programming approach has been taught in class during the experimental part, and can be adopted in the future for Software Engineering classes.
Appendix A

A.1 Abbreviations

- **AOP** Aspect Oriented Language
- **AJAX** Asynchronous JavaScript and XML
- **API** Application Program Interface
- **AT** Atomic Task
- **B2B** Business to Business
- **B2C** Business to Client
- **BPEL4WS** Business Process Execution Language for Web Services
- **CFES** Composite Formula Expression Subjects
- **COTS** Comercial off-the-shelf
- **CSV** comma-separated values
- **CVS** Concurrent Version System
- **DAO** Data Access Object
- **DTO** Data Transfer Object
- **DI** Dependency Injection Design Pattern
- **EAI** Enterprise Application Integration
- **ebXML** Electronic Business using eXtensible Markup Language
- **EJB** Enterprise Java Bean
- **ESB** Enterprise Service Bus
8. APPENDIX A

- **FTP** File Transfer Protocol
- **GUI** Graphical User Interface
- **GWT** Google Web Toolkit
- **IDE** Integrated Development Environment
- **IoC** Inversion of Control Design Pattern
- **JCA** Java Connector Architecture
- **JDBC** Java Database Connectivity
- **JSF** Java Server Faces
- **JSON** JavaScript Object Notation
- **JSP** Java Server Pages
- **LMS** Learning Management System
- **LOC** Lines of Code (Metric)
- **MMEPAD** Movie-Based Multimedia Environment for Programming and Algorithms Design
- **MVC** Model View Controller Design Pattern
- **MVP** Model View Presenter Design Pattern
- **MVVM** Model-View View Model Design Pattern
- **PaaS** Platform-as-a-Service
- **PM** Presentation Model Design Pattern
- **QoS** Quality of Service
- **REST** Representational State Transfer
- **SCA** Service Component Architecture
- **SLA** Service Level Agreement
- **SOA** Service Oriented Architecture
- **SOAP** Simple Object Access Protocol
- **TC** Task Collection
- **TVE** Tsunami Visualizing Engine
• **UDDI** Universal Description Discovery and Integration
• **WS-*** Web Services specifications, refering to the many Web services standards.
• **WS-CDL** Web Services Choreography Description Language
• **WSDL** Web Service Definition Language
• **XPath** XML Path Language
• **XSLT** Extensible Stylesheet Language
A.2 Virtual-MVC Qualitative evaluation study. Survey to evaluate Developer’s perception of complexity
Virtual-MVC Design Pattern – Developer usability study.
During the course project, you worked developing application’s components in the form of services. It is important for us to understand what is your experience using Virtual-MVC for the development process.

Section I
This section aims to understand your experience working with Web applications. Please, circle the most appropriate answer.

1) How would you rate your expertise developing web applications?
   - None
   - Poor
   - Fair
   - Strong
   - Expert

2) If you have worked with web service’s applications, how do you rate your expertise developing web services?
   - None
   - Poor
   - Fair
   - Strong
   - Expert

3) If you have worked on SOA development projects, how do you rate your expertise developing Service-Oriented applications?
   - None
   - Poor
   - Fair
   - Strong
   - Expert

4) Mention the programming languages that you normally use for Web development?

_________________________________________________________________________________________
_________________________________________________________________________________________
_________________________________________________________________________________________

_________________________________________________________________________________________
**Section II**
This section aims to understand your perception of development complexity using Virtual-MVC platform.

5) What is the most appropriate time for you in order to learn the pattern and development steps?
   - 1 week or fewer
   - 2 weeks
   - 3 weeks
   - 4 weeks
   - 5 weeks or more

6) Do you think that all development steps are appropriate?
   - Strongly disagree
   - Disagree
   - Neutral
   - Agree
   - Strongly Agree

7) Do you consider that is difficult to develop Service-Oriented applications using Virtual-MVC?
   - Strongly disagree
   - Disagree
   - Neutral
   - Agree
   - Strongly Agree

8) If your rate was disagree or strongly disagree, please explain the reason/s.
   
   ________________________________________________________________
   
   ________________________________________________________________
   
   ________________________________________________________________

9) Does the number of steps required to implement a service makes the implementation complex?
   - Strongly disagree
   - Disagree
   - Neutral
   - Agree
   - Strongly Agree

10) Do you consider the time required to develop services is appropriate?
    - Strongly disagree
    - Disagree
    - Neutral
    - Agree
    - Strongly Agree

11) Can you specify the most difficult component needed to develop the application? Put the number from 1 (easier) to 3 (most difficult) to evaluate the level of difficulty.
    
    Model ( )  View ( )  Controller ( )

12) Would you consider easy to understand the relations between model, view and controller components, in order to implement a Web service?
    - Strongly disagree
    - Disagree
    - Neutral
    - Agree
    - Strongly Agree

13) From 1 (easier) to 5 (most difficult), rate the difficulty of debugging your application?
    - 1
    - 2
    - 3
    - 4
    - 5
Section III
This section aims to understand your perception of services reusability by applying SOA principles.

14) Once you have developed the services, if you need to modify the function in the model, and update the correspondent views and controllers, do you think this will be a complex task compared with the same tasks in traditional web applications (JSP, ASP.NET, PHP, etc)?

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
</table>

15) Once you have developed the services, if you need to add a new service in the sequence, do you think this will be a complex task compared with the same task in traditional web applications (JSP, ASP.NET, PHP, etc)?

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
</table>

Section IV
Please answer this section only if your expertise of developing web services is between Fair and Expert. This section aims to get your perception of Virtual-MVC vs. Point-to-Point Web services integration, and Service-Oriented applications.

16) From your experience working with Web services, do you consider that Virtual-MVC facilitates the development?

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
</table>

17) Compared with your usual approach developing and integrating Web services, mention what do you think are advantages of using Virtual-MVC platform?
____________________________________________________________________________________________
____________________________________________________________________________________________
____________________________________________________________________________________________
____________________________________________________________________________________________

18) Compared with your usual approach developing and integrating Web services, mention what do you think are disadvantages of using Virtual-MVC platform?
____________________________________________________________________________________________
____________________________________________________________________________________________
____________________________________________________________________________________________
____________________________________________________________________________________________
Section V
This section we would like to know your opinion of the course.

19) Were the topics covered by the course likely to be used in your own research?
- Strongly disagree
- Disagree
- Neutral
- Agree
- Strongly Agree

20) In overall, was the course useful for you?
- Strongly disagree
- Disagree
- Neutral
- Agree
- Strongly Agree
References


REFERENCES


[22] GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. Design Patterns Elements of Reusable Object Oriented Software. Addison-Wesley, New York, US, 1994. 31

REFERENCES


109
REFERENCES


[38] MING-XIA, G., AND KEMING, T. Comparative analysis of webforms mvc and mvp architecture. In Environmental Science and Information Application Technology (ESIAT), 2010 International Conference on (2010), vol. 2, pp. 391–394. 25


[44] OKA, T., VAZHENIN, D., AND VAZHENIN, A. Web-based task management tools in e-learning environment. In International Conference on Computational Technologies in Electrical and Electronics Engineering Sibircon (July 2008), IEEE, pp. 100–105. 60, 63


[48] PAUTASSO, C., ZIMMERMANN, O., AND LEYMANN, F. Restful web services vs. ”big” web services: Making the right architectural decision. In Proceedings of the 17th International Conference on World Wide Web (New York, NY, USA, 2008), WWW ’08, ACM, pp. 805–814. 11, 12, 13, 14


[55] SpringSource. Spring, java/j2ee application framework. [http://static.springsource.org/spring/docs/2.5.x/spring-reference.pdf](http://static.springsource.org/spring/docs/2.5.x/spring-reference.pdf), 2004. 25, 26, 35


REFERENCES
