

**REST in peace**

**G.H.Lakmal - 12000779**

**M.S. Tharindu - 12001384**

**R.M.K.E. Kumarasinghe - 12000762**

**Supervisor : Dr. D.A.S Atukorale**

**December 2016**

Submitted in partial fulfillment of the requirements of the

B.Sc(Hons) in Software Engineering 4th Year Project

(SCS4123)



## Declaration

We certify that this dissertation does not incorporate, without acknowledgment, any material previously submitted for a degree or diploma in any university and to the best of our knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. We also hereby give consent for our group dissertation, if accepted, be made available for photocopying and for interlibrary loans, and for the title and abstract to be made available to outside organizations.

|  |  |  |
| --- | --- | --- |
| **Group Members** | **Signature** | **Date** |
| G.H. Lakmal |  |  |
| M.S. Tharindu |  |  |
| R.M.K.E. Kumarasinghe |  |  |

This is to certify that this dissertation is based on the work of Mr. G.H. Lakmal, M.S. Tharindu and R.M.K.E. Kumarasinghe under my supervision. The dissertation has been prepared according to the format stipulated and is of the acceptable standard.

Supervisor’ (s) Name: Dr. D.A.S Atukorale

……………………………………………… ……………………….

Signature of Supervisor Date:

## Abstract

## Acknowledgements

## Table of Contents

[Declaration iii](#_Toc470868828)

[Abstract iv](#_Toc470868829)

[Acknowledgements v](#_Toc470868830)

[Table of Contents vi](#_Toc470868831)

[List of Figures vii](#_Toc470868832)

[List of Tables viii](#_Toc470868833)

[List of Acronyms ix](#_Toc470868834)

[Chapter 1 - Introduction 1](#_Toc470868835)

[1.1 Background 1](#_Toc470868836)

[1.2 REpresentational State Transfer (REST) 2](#_Toc470868837)

[1.3 REST API 3](#_Toc470868838)

[1.4 Issues and Challenges 4](#_Toc470868839)

[1.5 Remedial Approaches 5](#_Toc470868840)

[1.6 Current Solutions 6](#_Toc470868841)

[1.7 Motivation 8](#_Toc470868842)

[1.8 Goal and Objectives 8](#_Toc470868843)

[1.8.1 Goal 8](#_Toc470868844)

[1.8.2 Objectives 8](#_Toc470868845)

[1.9 Scope and Limitations 9](#_Toc470868846)

[1.10 Expected Contribution 9](#_Toc470868847)

[1.11 Project Outline 9](#_Toc470868848)

[Chapter 2 - Background/Literature Review 10](#_Toc470868849)

[Chapter 3 - Analysis and/or Design 11](#_Toc470868850)

[Chapter 4 - Implementation 12](#_Toc470868851)

[Chapter 5 - Results and Analysis 13](#_Toc470868852)

[Chapter 6 - Evaluation and Testing 14](#_Toc470868853)

[Chapter 7 – Conclusion 15](#_Toc470868854)

[References 16](#_Toc470868855)

[Appendices 17](#_Toc470868856)

## List of Figures

## List of Tables

## List of Acronyms

API - Application Programmable Interface

CASE - Computer Aided Software Tools

CORBA - Common Object Request Broker Architecture

DCOM - Distributed Object Component Model

ERD - Entity Relationship Diagram

HTTP - Hypertext Transfer Protocol

REST - REpresentational State Transfer

RMI - Remote Method Invocation

SOAP - Simple Object Access Protocols

UML - Unified Modeling Language

WWW - World Wide Web

## Chapter 1 - Introduction

## 1.1 Background

World Wide Web was invented in 1980, by Tim Berners-Lee and it has been accepted as a global medium of information sharing. In connection with this invention, Tim has introduced a number of fundamental technologies and the following three out of them still remains as the foundation of web which is widespread in the present world. **(**[**https://drive.google.com/open?id=0B2gf5XS67I2AU0lEYkFGWmlLU2c**](https://drive.google.com/open?id=0B2gf5XS67I2AU0lEYkFGWmlLU2c) **)**

1. Hypertext Markup Language (HTML) was introduced as universal language for the web.
2. Uniform Resources Identifier (URI) was introduced as a unique type of address to identify each resource on the web
3. Hypertext Transfer Protocol (HTTP) was introduced as a standard protocol which allows the retrieval of linked resource from the web. The latest version of this protocol is HTTP/1.1

As the WWW became popular in the world, web services came into use and it was rapidly grown to address the requirements of its users. The main goal of web services is to exchange information among the application in a standard way. Web services are offered through purpose-built web servers which serves data or information to a client program or an application.

However, there are some alternative technologies such as RMI, CORBA, DCOM which came into use prior to the web services to create client server applications. As these technologies are highly coupled in which both the server and client are interdependent, they create problems on compatibility and security. Moreover, platform and technology used in both the server’s and client’s side should be the same. Therefore, the web services were accepted as a better solution to address the need for loosely coupled and platform independent distributed systems.

**(**[**https://drive.google.com/open?id=0BxADaSj3YdrMSS0tVm9NYV9weGc**](https://drive.google.com/open?id=0BxADaSj3YdrMSS0tVm9NYV9weGc) **)**

There are two types of web services introduced as “BIG” and RESTful web services. BIG web services are based on Simple Object Access Protocol (SOAP) while the RESTful services are based on WWW standards. **(**[**https://drive.google.com/open?id=0B2gf5XS67I2AOWhXTDcwWUR2UlE**](https://drive.google.com/open?id=0B2gf5XS67I2AOWhXTDcwWUR2UlE) **)**

SOAP was invented in 1998 by Winer D et al. for Microsoft. SOAP contains three main entities: service provider, service registry and service consumer. The service consumer finds the service description in the service registry which is published by the serviceprovider. The communication among these entities is based on XML and SOAP where it is secured but demands more bandwidth. Due to heavy bandwidth usage SOAP causes network traffic, higher latency and delays in processing . The RESTful architecture is used to overcome these limitations .

**(**[**https://drive.google.com/open?id=0B2gf5XS67I2AXzZuYzlHTUFZdGs**](https://drive.google.com/open?id=0B2gf5XS67I2AXzZuYzlHTUFZdGs) **)**

## 1.2 REpresentational State Transfer (REST)

The architectural style; REpresentational State Transfer (REST) was introduced by Roy Fielding in 2000. REST is a client server architecture. The process in this architecture is that when the client sends a request to the server, the server in return processes the request and sends the response to the client. REST exhibits four principle design attributes. **(**[**https://drive.google.com/open?id=0BxADaSj3YdrMLUFVRTlNQnNsajg**](https://drive.google.com/open?id=0BxADaSj3YdrMLUFVRTlNQnNsajg) **)**

1. Resource Oriented Design

In this design, each concept is modeled as a resource which is identified by a unique URI.

1. Various types of representations

The resource may have various types of representations such as URIs, URLs, URNs.

1. Usage of HTTP standards

The HTTP offers a standard and uniform interface for all resources.

1. The Hypermedia

Hypermedia stands as an application state engine which semantically links the related resources.

Once the above mentioned attributes are fulfilled by a web service, it falls into the Richardson Maturity Model (RMM) which appears as a way to grade a REST service according to the constraints of REST.

Accordingly, the principles of REST allow simple but efficient technology which can gain the acceptance of a major share of programmable web.

**(**[**https://drive.google.com/open?id=0B2gf5XS67I2AWTJGeXhESEdrWUE**](https://drive.google.com/open?id=0B2gf5XS67I2AWTJGeXhESEdrWUE) **)**

## 1.3 REST API

Application Programmable Interfaces (APIs) are used as a facilitator to the process of REST architecture. An API supports interaction between computer programs by exposing a set of data and functions, and also it allows the computer programs to exchange information.

A web API functions as the web service listening and responding to client requests. In the modern web services, the REST architectural style is commonly applied to the design of APIs. When a web API is conformed to the REST architectural style, it becomes a REST API. A REST API makes a web service RESTful. **(**[**https://drive.google.com/open?id=0B2gf5XS67I2AWTJGeXhESEdrWUE**](https://drive.google.com/open?id=0B2gf5XS67I2AWTJGeXhESEdrWUE) **)**

Client developers can be easily attracted to use web services by well-designed Web APIs. There’s a big competition among the web services in the open market for the attention of client developers. Therefore a well designed REST API is a feature which should be available.

## 1.4 Issues and Challenges

Although RESTful web services are accepted to be useful, its development may lead to many issues and challenges such as high costs, late delivery, inability to address all the requirements and project failure. As development process contains with several phases: designing, coding, testing and maintenance, the above mentioned issues can be a result of the problems which may arise at each of these different phases.

**(**[**https://drive.google.com/open?id=0B2gf5XS67I2AZThpVENUM0FUTkk**](https://drive.google.com/open?id=0B2gf5XS67I2AZThpVENUM0FUTkk) **)**

In a typical software design phase UML and ER diagrams are used to model the business process and the relationship between data entities. When designing RESTful web services it is necessary to gather several other entities and relationships than that a typical software gathers such as resources, URIs, requests and responses. Due to complexity of identifying a number of entities, RESTful web services may miss some of the important design details which may cause severe issues in subsequent phases of the development.

**(**[**https://drive.google.com/open?id=0B2gf5XS67I2AUElQVHFkWjBMQ2c**](https://drive.google.com/open?id=0B2gf5XS67I2AUElQVHFkWjBMQ2c) **)**

With regard to the coding phase, a RESTful web service requires numerous layers such as REST API layer, service layer, data access layer etc. which depends on the design. Though the quality of the web service improves through this complex process, the advancement of the layers may require lengthy coding time and much effort to complete the project. As the concentration on time and effort increases, the attention on the requirements of the project and the overall business process would be decreased.

**(**[**https://drive.google.com/open?id=0B2gf5XS67I2AZm53ejJYSnZPWWs**](https://drive.google.com/open?id=0B2gf5XS67I2AZm53ejJYSnZPWWs) **)**

The next stage is the maintenance phase, in which the project would be improved with some important updates and bug fixes. To improve the project, proper documenting of the web service design and code base is essential. Proper maintenance in quality of the design and code base will also be vital to extend the product. However, the large code base and the complexity of documenting would challenging to the whole web service. Developers may miss, adhering to proper code conventions. **(**[**https://drive.google.com/open?id=0BxADaSj3YdrMYi1RbjFONEhZY28**](https://drive.google.com/open?id=0BxADaSj3YdrMYi1RbjFONEhZY28) **,** [**https://drive.google.com/open?id=0BxADaSj3YdrMWTVwTThiVTJaRkk**](https://drive.google.com/open?id=0BxADaSj3YdrMWTVwTThiVTJaRkk) **)**

Moreover, the technology changes such as programming language and frameworks, will inspire the developers to migrate from old technologies to the new, and this may lead them to an extra challenge as they have to rewrite the whole code under the new technology.

## 1.5 Remedial Approaches

Research in the literature forward suggestions to overcome the issues and reduce the risks related to RESTful web services.

One of the suggestions is to automate the development process of the RESTful web services. Different approaches are introduced in this regard such as Rule Based, Property Based, Behavioural Driven and Model Driven to automatomate the development process. Out of these approaches, the Model Driven Architecture (MDA) announced by Object Management Group (OMG) has been widely accepted as a model driven engineering instance which changes the attention of the developer from programming language code to models and transformation.

**(**[**https://drive.google.com/open?id=0B2gf5XS67I2AZThpVENUM0FUTkk**](https://drive.google.com/open?id=0B2gf5XS67I2AZThpVENUM0FUTkk) **,** [**https://drive.google.com/open?id=0B2gf5XS67I2AYmN0SjlidWthNWs**](https://drive.google.com/open?id=0B2gf5XS67I2AYmN0SjlidWthNWs)**,** [**https://drive.google.com/open?id=0B2gf5XS67I2AUElQVHFkWjBMQ2c**](https://drive.google.com/open?id=0B2gf5XS67I2AUElQVHFkWjBMQ2c) **)**

When considering the main goal of MDA, it focuses on raising the level of abstraction by keeping the developer away from implementation details, to allow him focus on the real problem at hand.

For the automation of the development process of REST, different tools are available which adhere MDA principles.

## 1.6 Current Solutions

With regard to the design process Computer Aided Software Engineering (CASE) tools have been introduced related to REST web services, to overcome the design issues. Visual Paradigm REST Modeling tool is one of the REST CASE tools which allows the user to design the REST API through diagramming. This tool doesn’t generate the source code and its sole function is allowing the user to model his project.**(**[**https://www.visual-paradigm.com/support/documents/vpuserguide/276/3420/85154\_modelingrest.html**](https://www.visual-paradigm.com/support/documents/vpuserguide/276/3420/85154_modelingrest.html)**)** The Swagger UI tool on the other hand, provides a IDE to manually type the design specifications of the project using YAML (A human readable language use for configurations of software projects ) or JSON notations **(**[**http://swagger.io/swagger-ui/**](http://swagger.io/swagger-ui/) **)**. However when using these tools the user is obliged to learn the syntax of notations and typing will be time consuming.

In the coding phase related to the RESTful web services many automated tools are used. WSO2 Data Services Server **(**[**http://wso2.com/products/data-services-server/**](http://wso2.com/products/data-services-server/)**)** facilitates integration of data stores, creation of composite data views, and hosts data in REST Style web resources. This will automatically generate the REST API code through the data stores and expose it. However this tool will avoid a developer from adding and modifying advanced functionality, or from deploying the product in his own server. On the other hand, API Workbench **(**[**http://apiworkbench.com/**](http://apiworkbench.com/) **)**, API Blueprint **(**[**https://apiblueprint.org/**](https://apiblueprint.org/) **)** and Swagger Codegen **(**[**http://swagger.io/swagger-codegen/**](http://swagger.io/swagger-codegen/)**)** are tools which generate REST API by allowing the user to input specifications in REST API Modeling Language (RAML) or JSON format. But the user doesn’t acquire the overall picture of the project by only defining specifications without a proper model.**(**[**https://drive.google.com/open?id=0B2gf5XS67I2AUElQVHFkWjBMQ2c**](https://drive.google.com/open?id=0B2gf5XS67I2AUElQVHFkWjBMQ2c) **)**

Considering the test phase, automated testing tools can be identified related to REST testing such as SOAPUI **(**[**https://www.soapui.org/**](https://www.soapui.org/) **)** and vREST **(**[**https://vrest.io/**](https://vrest.io/) **)**. In spite of the availability of these tools the user have to make effort to configure them manually before testing the project. Nevertheless these tools do not test the whole project except only the REST API.

**(**[**https://drive.google.com/open?id=0B2gf5XS67I2AUmszd0RGdjR0ZVU**](https://drive.google.com/open?id=0B2gf5XS67I2AUmszd0RGdjR0ZVU) **)**

SpyREST **(**[**http://www.spyrest.com/?from=@**](http://www.spyrest.com/?from=@) **)** and Swagger **(**[**http://swagger.io/**](http://swagger.io/) **)** Documentation are some of the tools which generate the documentations for RESTful web services. As these tools are limited to the function of generating documentations only for REST API, they are unable to cover to document the remaining code base.

Thus it is revealed that the tools which were discussed above are limited only to specific phases, but do not facilitate the whole development process. Regarding these limitations some other commercial tools, which support the entire process have been invented and they appear to be much more advanced than the former tools. Some of these tools are IBM Rational Rose and Swagger Tool Collection. IBM Rational Rose can be functioned from the design phase to the maintenance phase with documentation and tests. In the design phase it models diagrams which takes static and dynamic process. Though this tool appears to be advanced with these benefits, there are some limitations in relation to it. Most specifically this tool supports Java based projects only and it does not provide data access layers.

Swagger toolkit is an open source alternative for the IBM Rational Rose. This tool stands beyond the former by supporting both the server side (14 programming languages) and client side (21 programming languages) technologies that helps to generate a RESTful Web service, including Java, PHP and JavaScript. The constraint on this toolkit is that it generates only the REST API and the other layers such as data access and SQL scripts are not created by this toolkit.

## 1.7 Motivation

In conclusion RESTful web service development meets with several challenges and limitations: late delivery, complex code structure, high costs, more effort, lengthy project life cycles, problems in testing and maintenance. Even the currently used tools to overcome these challenges do not provide support to automate the development of whole RESTful web service from client end to server end. Thus implementing an innovative solution addressing all these challenges is a timely concern.

## 1.8 Goal and Objectives

## 1.8.1 Goal

The main goal of this research is to propose a solution and develop a software tool which automates the development process of projects, based on RESTful web services through a Model Driven Approach. Specific objectives have been identified to achieve this main goal are given below.

## 1.8.2 Objectives

1. To introduce a Computation Independent Model (CIM) which identifies the abstract domain entities of a RESTful web service without any design or implementation details.
2. To introduce a Platform Independent Model which acquires design specification of the REST project without any implementation details
3. To introduce a Platform Specific Model generate REST project with concrete implementation details of PIM
4. To introduce transformation mechanisms which converts CIM to PIM, PIM to PSM and PSM to project source code.
5. To implement a software tool which adhere MDA based concepts

(The development of this tool will deliver fast and quality code, reduce development costs, reduce effort on development and shorten the project life cycle.)

## 1.9 Scope and Limitations

## 1.10 Expected Contribution

This research will contribute to the industry in applying MDA concepts to RESTful web services development in order to increase automation, faster delivery and quality code.

## 1.11 Project Outline

The organization of the dissertation is as follows:

Chapter 2 provides background and the literature review of the research.The literature is organized under 5 topics; model driven architecture, REST modeling, automated REST source code generation, automated REST API testing and automated REST API documentation.

Chapter 3 explains the methodology used and the adaptation of the methodology to the system

Chapter 4 of the dissertation comprises the analysis and the design of the proposed system.

Chapter 5 of the dissertation provides the implementation details, technologies used and the standards followed.

Chapter 6 contains the evaluations conducted with results and the testing approach used.

Chapter 7; the final chapter conclude the dissertation with recommendations on future research.

## Chapter 2 - Background/Literature Review

# Different approaches are introduced for software automation such as Rule based (Ref), Property based (Ref), Behavioural Driven (Ref) and Model Driven Engineering (Ref). According to Zolotas et al. (Ref) , use of MDE has been suggested as one of the advanced engineering practices for automated API generation. MDE aims at creating domain models which represents the conceptual models for related entities to a specific domain, and making maximum use of it. Zolotas further states that out of different approaches in the MDE domain, MDA is an outstanding architecture ideal for automating the REST APIs.

## 2.1 Model Driven Architecture (MDA)

MDA is used as a [software design](https://en.wikipedia.org/wiki/Software_design) approach in the development process of software which is announced by OMG. It contains a set of principles to define the specifications of a model. **(Ref)** MDA has been applied to several domains and has appreciated for its increased automation, formalism and higher productivity. On the other hand, it has criticized for the complexity of modeling itself, varying automation, varying code generation and less coherence among tools which adhere MDA. **(Ref)**  However with all these **pros and cons** it is accepted that this methodology can be successfully applied on REST **(Ref)**

## 2.2 REST Modeling

In the MDA domain, to capture requirement specifications of RESTful web services an annotated description language was used previously. Kopecky et al. has invented hREST **(Ref),** Maleshkova et al. has extended SAWSDL **(Ref)** as an annotation description language. Later Pagliarecci et al. introduced SWSAL. **(Ref)** Taking an alternative approach Tavares et al. introduced a meta-model for semantic annotation languages for RESTful web services. **(Ref)** Their main focus was to achieve increased interoperability among them.

Addressing this problem in an abstract manner, S. Schreier **(Ref)** presented a meta-model for RESTful services which is based on Ecore meta-model **(Ref)** introduced by OMG. Applying the meta-model concept, Visual Paradigm **(Ref)** introduced a REST API design tool which provides a diagramming method to acquire the RESTful web service specification. According to the official site of Visual Paradigm REST Modeling tool, it is limited only to provide solutions for the design phase of REST APIs. Thus it is obvious that Visual Paradigm does not provide functionalities to generate API source code through the drawn design. Rational Software Architect **(Ref)** on the other hand introduced by IBM acquires both static and dynamic model design of RESTful web service using several UML diagrams such as Class, Activity and Sequence. As mentioned in the official site of Rational Software Architect, it does not provide models to gather data store structure and relationships of a project.

## 2.3 Automated REST Source code generation

Several API specifications were introduced to gather REST constraints to generate API source code. WADL **(Ref)**  was the initial practice to describe web services. As an extension, more human readable description languages such as REST API Modeling Language (RAML) **(Ref)** was introduced.

RAML presents the concept to identify a REST API as a tree structure where a URI or a Reference appeared as a node of the tree while entry point of the REST API is identified as the root. Children are identified as Resources which are accessible through parents. A URI may be associated with one or more access methods. **(Ref)**

Since aforementioned languages are designed for REST API specification only, developers face problems specifying other entity details essential for the project such as database details and relations. **(Ref)**

Based on RAML, Swagger **(Ref)** introduced an Open API Specification to define REST constraints in an advanced way. Parallel to Swagger, API Blueprint **(Ref)** introduced an alternative API Specification based on same language - RAML. Both Swagger and API Blueprint extended their work by introducing tools (Swagger Codegen **[Ref]** by Swagger and APIMATIC **[Ref]** by API Blueprint) to generate REST APIs based on their own specifications. Both APIMATIC and Swagger Codegen automatically generates REST API source code by allowing user to code API constraints using their specifications by an IDE. However APIMATIC is limited to generate REST APIs only for 3 client side and 7 server side languages. **(Ref)** Comparatively Swagger Codegen supports 21 client side and 14 server side languages.

Nevertheless both Swagger Codegen and APIMATIC do not give any facilities to include data layer and data access layer details and relations that are needed to generate a complete Restful web service. Due to unavailability of a modeling diagramming interface in both tools, user may be unable to conceptualize the total picture of the business process and entity relations. **(Ref)**

In addition to these Swagger and API Blueprint there are some other tools with very similar features such as REST United and RESTlet Studio which cause the same issues appear in previous tools. **(Ref)**

Rational Software Architect presented an advanced diagramming methodology to capture static and dynamic business process of a web service through UML diagrams. Using the drawn models it produces REST API with business logic functionalities. Even this tool does not support data and data access layer code generation. **(Ref)**

Addressing the unavailability of data layer and data access layer in REST API code generation, WSO2 Data Services Server **(Ref)** proposed an alternative approach to publish a RESTful service. Acquiring a database specification it generates data resources and publish them as a web service. However modification or addition of business logics cannot be achieved through this server.

Introducing a far more different approach Zolotas et al. **(Ref)** proposed to integrate database specifications into REST API specifications. Tools such as MySQL Workbench **(Ref)**, SQLLite **(Ref)**, RazorSQL **(Ref)** are available to generate database script from a PIM. But it is costly to integrate the generated script manually to the REST API source code through a data access layer. **(Ref)**

## 2.4 Automated REST API Testing

Several Approaches are available to simplify the testing mechanism of RESTful APIs. Tools which are commonly used, such as JUnit **(Ref)**, NUnit (Ref) and other xUnit (Ref) frameworks focus on unit-testing. Due to the dependency on implementation language, these tools cannot be used for all scenarios. Due to this problem several testing approaches were suggested to test RESTful services in an automated way. One of these approaches is known as Haleby’s approach **(Ref)** for rapidly writing test cases for any RESTful API. However this approach only speed up the manual testing process. Chakrabarti et al. **(Ref)** introduced an approach to configure an XML file to specify the test cases. This work proved that an automated test case approach can be used for RESTful APIs. However the advanced configuration of the XML file will lead to several difficulties in writing test cases. **(Ref)** Following the approach of Chakrabarti, Fertig and Braun **(Ref)** suggests, a model driven testing approach to automate the testing.

## 2.5 Automated REST API Documentation

Several research **(Ref)** propose best practices for API documentation that can also be applied to REST APIs. According to Robillard et al. **(Ref)** lack of documentation raises severe obstacles in maintenance and learning API structures. He recommends that a good API documentation must: include good examples, be completed, be simple and include system design models with visual images.**(Ref)** But Stepalina **(Ref)** realized that writing a comprehensive document manually for an API is costly and time consuming. She proposed an automated document generation approach as a solution. Tools such as JavaDoc **(Ref)**, RDoc **(Ref)**, Jadeite **(Ref)** are intended to automatically generate documentation for local APIs. Due to the absence of HTTP information documentation above tools cannot be directly applied for REST API. **(Ref)** For documenting a REST API Mayers et al. **(Ref)** recommended to document all starting points, end points and overall map of the API including all REST entity specifications. With concern to automated document generation Swagger toolkit, API Blueprint and RESTlet Studio produce documentation for REST APIs. But unavailability of high level design models such as UML and ER diagrams in the documentation does not satisfy the requirements related to the principles presented by Robillard.

Accordingly, the above information on different tools and concepts reveals that they do not provide solutions to automate the whole RESTful web service development. **(table no)** displays a comparison of aforementioned tool usage against RESTful project automation areas.

# Table 2.1: Existing tool usage against RESTful project automation areas

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Client Side** | **REST API** | **Service Layer** | **Data Access Layer** | **Database Script** | **Test**  **Script** | **Documentation** |
| API Blueprint Toolkit | ✔ | ✔ | ✔ |  |  |  | ✔ |
| RAML | ✔ | ✔ | ✔ |  |  | ✔ | ✔ |
| Rational Software Architect |  | ✔ | ✔ |  |  |  | ✔ |
| RazorSQL |  |  |  |  | ✔ |  |  |
| REST Assured |  |  |  |  |  | ✔ |  |
| Restlet Studio | ✔ | ✔ | ✔ |  |  |  | ✔ |
| REST United | ✔ | ✔ | ✔ |  |  | ✔ | ✔ |
| SpyREST |  |  |  |  |  |  | ✔ |
| SQLines |  |  |  |  | ✔ |  |  |
| Swagger Toolkit | ✔ | ✔ | ✔ |  |  |  | ✔ |
| Test the REST |  |  |  |  |  | ✔ |  |
| WSO2 Data Services Server |  | ✔ | ✔ | ✔ |  |  | ✔ |

## Chapter 3 - Methodology

As explained in Chapter 2, MDE approach attracts developer’s attention towards business process implementation through model transformation by sheltering implementation details.

Computation Independent Model(CIM), Platform Independent Model(PIM), Platform Specific Model(PSM) and transformation mechanisms between models are necessary to conform with the MDA approach.

## 3.1 Domain Models

With respect to the REST API Modeling CIM identifies the following abstract domain entities:

1. **Resource Type** (a model to represent a concept or object and its properties)
2. **Resource Identifier Pattern** (a URI representation)
3. **Attribute** (specifies a property of a Resource Type such as ‘optional’ while conforming to a defined Data Type)
4. **Data Type** (represents Primitive Data Type such as integer, string or a Collection Type such as list or array)
5. **Method** (responsible for defining the behaviour of a Resource Identifier Pattern and determines the set of produced and consumed Media Types)
6. **Method Type** (an HTTP verb such as GET, POST)
7. **Parameter** (a numerical or other measurable factor conform to a Data Type)
8. **Media Types** (enable content negotiation and identified by a name such as application/json, application/xml)
9. **Representations** (uniquely identified by a name, defines its Media Type and models the data sent by the server

**RDBMS Table**, **Column**, **Primary Key** and **Foreign Key** stereotype are used to model relational database schema concepts.

To integrate the database schema to the REST API it is necessary to build up a connection between REST API Modeling entities and Database Schema Modeling entities. A **Table-Resource Connector** extension is used to interweave the separate layers.

Platform Independent Model(PIM) illustrates the abstract design of the visualized system neglecting the implementation details. In PIM REST API design, relational database schema concepts and project configuration details such as programming language, build tool are composed into a machine readable structured notation.

Platform Specific Model(PSM) enriches the PIM with concrete implementations by applying envisioned technologies and designs.

## 3.2 Model Transformations

Model transformation from CIM to PIM is done through an object notation mapping. The properties or attributes of the abstract domain entities are mapped into a structured notation.

Transformation from PIM to PSM model is done through mappings between PIM constructs and platform specific constructs. Eg:- Data Type transformation from “string” to “var” or “String” depends on the programming language; Javascript or Java.

Finally the source code generation is done by applying PSM to a set of predefined templates.

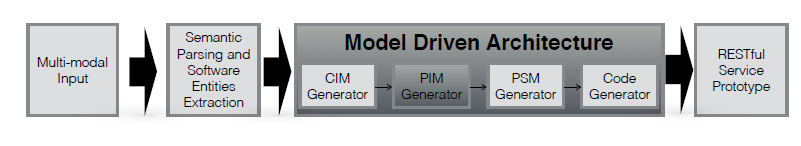
An abstract view of the concept discussed above is depicted by Figure 3.1. 

Figure 3.1. Main phases of RESTful services development **(Ref)**

## Chapter 4 - Analysis and Design

## 4.1 System Analysis

## The design of the system is basically based on the main concepts of model driven approach which was discussed in Chapter 02 & 03. To execute this approach, a software application should be designed and provided to the user. Prior to design a software application, it is necessary to concern the user requirements when it is applied to the real world situations. The software should be contained with a GUI for the user to specify the project model , constraints and some other components to generate REST based project by them. In this process two types of users were identified: client developer (user) and system developer (admin).

Based on these both types of user scenarios, it was specified what the system should do and how it should be done. The user requirements related to client developer (user) and system developer (admin) are captured in **Figure 4.1 and Figure 4.2 respectively**.

Similarly the quality requirements of the proposed system and generated REST project are listed in **Table 4.1, Table 4.2 respectively**.

Table 4.1 - Quality requirements of the proposed system

|  |  |
| --- | --- |
| **Requirement** | **Rationale** |
| Availability | * System downtime should be minimized by following proper fault tolerance mechanisms. In case of a component crash, system should be able to recognize the fault and apply a proper recovery mechanism. |
| Security | * User access levels should be set and the system should only allow privileged users. * Privileges should be granted to the roles and each user may act one or more roles. So if a user does not has authorization, he cannot perform that task. * Auth security levels  should be used to ensure the security in communication. * Database and server security mechanisms have to be used to protect projects and other sensitive data. |
| Maintainability | * Proper coding standards should be followed * Proper documentation should be produced * System components should be maintainable |
| Performance | * The software should be able to support up to 50 simultaneous users. * Project should be available to download within 5 minutes. * Service should be available in 24x7 hours for the users. |
| Usability | * System GUI should be simple and easy to use |
| Understandability | * Standard coding and naming conventions should be used. * Documentation and system user manual should be understandable |
| Reliability | * Implemented software should pass proper test cases |

Table 4.2. - Quality requirements of the generated REST project

|  |  |
| --- | --- |
| **Understandability** | * The generated source code of the product should be easy to understand * REST API documentation should be understandable |
| **Maintainability** | * Proper coding standards should be followed * Proper documentation should be produced * Generated product should follow proper software design abstraction principles. Eg:- REST API, API services, DAO. * A comprehensive documentation about the REST API and Data Objects should be provided. |
| **Reliability** | * Proper testing documentations should be provided for the generated REST project * The client should be able to rely on the generated product. Proper testing mechanism is needed. Importable testing suit and a test report should be provided |

Moreover the proposed system should take **REST API Specification with RDBMS schema** **Technical Specification, Project Specification** as inputs and  should produce Source code of **REST API, Service and Data Access Layer, RDBMS script, Test script and report, REST API documentation** as outputs.

## 4.2 System Design

The system design and architecture was proposed relying on the above analysis. If it is intended to provide the whole system to the user as a single component, user machine should be very powerful to operate it due to the complex functionality of the system. This will be a limitation to the aforementioned performance quality requirements. Moreover application server should operate on the same user machine with several configurations. It would increase the complexity of system configuration and also it would degrade the performance of user machine. Thus a monolithic system would not be suitable for proposed system.

It emphasize that this system should be decoupled and system functionalities have to be exposed as a service while client application consumes the services. It would support to minimize the limitations of the monolithic system such as maintainability and complex configuration issues. Nevertheless some other challenges such as simultaneous handling of many user requests and maintaining high availability may still remain unsolved in client-server architecture.

Considering these functional and nonfunctional requirements, **Service Oriented Architecture (SOA)** was suggested as the ideal solution for the system design. SOA is an architectural style in which systems consists of service users and service providers. **(Ref)**

Adhering to SOA the user needs an interface to access the system and interact with it. Client application will provide the components to handle the user interaction with the system. REST server side and client side source code, test script and database script generations can be done by using independent services in the service layer. This kind of a structure will improve the parallel execution and reduce the project generation time. Also it will ease the maintainability of the system.

To communicate and control the independent services ( for service orchestration ) and user activities a component is used in the **Enterprise Service Bus layer( should use another word to describe the framework)**.

Based on the above rationale, it was decided to use SOA as the base architecture of the full system.

Design assumptions and dependencies

1. Availability of a working internet connection
2. Availability of the servers online
3. Availability of JRE 1.7 or above for the client side
4. Static modeling is the only consideration for modeling CIM (REST API+ RDBMS schema) (dynamic behaviour is not considered)

Proposed system was titled as **REST in Peace (RIP)**. Since it was expected that this system would allow the user to generate their REST projects in a **peaceful** and flexible manner.

Figure 4.3. depicts the abstract component layer structure which is specifically related to RIP system based on SOA.

Figure 4.3 - RIP System Architecture

Complying to the layers of SOA, RIP substitutes some specific components which represents different layers as follows:

* The **application layer** is substituted by **RIP UI** which consumes the service.
* The **service layer** is substituted by 4  components: **RIP client source code generator, RIP REST source code generator and RIP SQL generator, Test environment manager** which exposes as services.
* The **orchestration layer** is substituted by **RIP framework** which handles the communication between different services and client application.

## 

## Chapter 5 - Results and Analysis

## Chapter 6 - Evaluation and Testing

## Chapter 7 – Conclusion

## References

## Appendices