An Integrated REST API for ALM Tool Suite

An Internship Report

Submitted in Partial Fulfilment for the Award of

M.Tech in Information Technology

By

M.Yamini

MT2012168

To

International Institute of Information Technology

Bangalore - 560100

June 2014

Certificate

This is to certify that the internship report titled **‘ An Integrated REST API for ALM Tool Suite ’** submitted by **M.Yamini (MT2012168)** is a bona fide work carried out under my supervision at **Infosys Limited** from **10th November, 2013** to **13th June, 2014** in partial fulfilment of **M.Tech** course at **International Institute of Information Technology, Bangalore**.

Her performance and conduct during the internship period was satisfactory.

Dr.Srinivas Padmanabhuni

Principal Research Scientist & Associate Vice President,

Infosys Labs,

Infosys Limited, Bangalore-560100.

Date: 13th June, 2014

Place: Bangalore.

Acknowledgements

I take this opportunity to express my gratitude and regards to my mentor Mr.Allahbaksh Asadullah for his exemplary monitoring and constant encouragement throughout this internship period. The help and guidance given by him, time to time, shall carry me a long way in the journey of life on which I am about to embark. I also owe my gratitude to my college International Institute of Information Technology Bangalore(IIIT-B) and Dr. Srinivas Padmanabhuni,Principal Research Scientist & Associate Vice President, Infosys Labs, for giving me an opportunity to become a part of Infosys Limited which is one of the most renowned organization. I am obliged to my team members, especially Nikita Jain and Avitash Purohit, for their valuable suggestion and cooperation. Lastly, I thank the Almighty, my family and friends for their constant encouragement without which this assignment would not be possible.

Contents

List of Tables

List of Figures

Abstract

Application Lifecycle Management tools (ALM) are the product lifecycle management (governance, development and maintenance) software. They encompass requirement management, software architecture, computer programming, software testing, software maintenance, change management, project management and release management etc. A software enterprise develops suites of products which use different ALM tools. As developers move from one project to another, the underlying ALM tools also change. This makes development process difficult because each ALM tool has different UI and terminology even though the underlying functionality remains the same. So, this work aims at providing common integrated APIs for different ALM tools and its features. These APIs are capable of communicating with tools that manage the project under work. The APIs are exposed as web services designed on REST Architecture Style. The service based architecture is a rightful design choice because a web service provides interoperability and can be remotely invoked across networks. These features make it useful for developing a completely decoupled client-server design. Also, REST services are based on light-weight HTTP stateless protocol. This eliminates the complications involved in the usual WS*-protocol stack thereby creating effective public APIs.

Chapter 1

Application Lifecycle Management

1.1 Introduction

Application Lifecycle Management (ALM) consist of a collection of practices that refer to the capability to integrate, coordinate and manage different phases of a software delivery process. ALM's goal is to help IT define business needs and realize them in cost-effective manner[1]. ALM approaches the development lifecycle with an end-to-end perspective i.e., from the initial idea drafting to the end of product delivery. A Forrester study[2] of various ALM tools defines ALM as synchronization between SDLC activities like requirement elucidation, design modelling, development and testing through interactions between development artifacts by enforcing processes that span across these activities.

1.2 Aspects of ALM

ALM can be broadly categorized into three aspects namely governance, development and operations [3].Governance consists of the decision making process made throughout the

SDLC. Development encompasses the process of creating the application. This phase is recurring depending on the type of development model used. Operations involve support and maintenance of the software after deployment.Some of the core features[4] of ALM solutions are:traceability of relationships between artifacts, automation of high-level processes, continuous reporting etc. Traceability of artifacts and their interdependencies varies with the number, size and scope of projects. Compliance requirements make traceability a necessity. ALM solution's frequent reporting and monitoring of project development provides real-time status updates of all life cycle related information.

1.3 Organizational Benefits of ALM

ALM provides consistent, current and an enterprise level-view of the development project. It provides decision makers with information that supports future investments by following standards using a consolidated metrics repository. It also provides information needed to identify, understand, track and mitigate development risks by identifying trends and adopting iterative development practices.A typical ALM approach increases control, visibility, and predictability for building modern applications. It also enables change management, risk-based prioritization, and communication across disparate teams.ALM solutions reduce delivery cycle time of an application release by increasing the business flow and by adopting agile practices. These solutions minimize downtime in production environment by using tools for streamlining communication and capturing diagnostic information. They also increase the ability of IT departments to rapidly build and adapt applications to support dynamically changing business requirements.

1.4 Tools for ALM

With off-shoring and outsourcing common within application development, the challenge of communicating across geographical, organizational and technical sectors are increasing. This causes hindrance and hampers productivity among distributed teams. Also, appropriate performance to maximize the business value across sectors becomes difficult to achieve. A proper tool set or software suite incorporated with ALM practices can provide a suitable solution to this issue.

1.4.1 Issues of ALM Tools

A number of different vendors today provide tools that are integrated in unidirectional manner i.e., tools that work well on only one aspect .As an example, Microsoft Visual Studio Team System provides range of tools supporting development process[3], but the support for other project processes are comparatively less.Most of the traditional ALM tool offered by vendors tend to have management support for processes within their disciplines. As an example[1], Build Management tools may have embedded processes for maintaining the build discipline, but most of it won't have support for Test and Validation Tool. The efforts involved in integrating ALM tools across multiple directions i.e., different disciplines, have resulted in a rather expensive and unstable business management design. These integrations are often susceptible to changes when one of the tool's features are modified. Also, the cost of upgrading them are difficult and expensive, thereby making them unsuitable for evolving business needs.The process of traceability and reporting across disciplines has also become complicated since each tools has its own internal data representation. Vendors have attempted to fix these issues by suggesting the enterprise to operate on single reposi-

tory and move to a single vendor for all ALM tools. However this approach won't have any large scale impact because multiple teams work on different platforms at different locations making the single repository solution difficult to achieve. Also, there is no single vendor who has every aspect of ALM embedded in his tool.

1.5 Agile ALM

Agile ALM combines software engineering practices with business management tactics i.e., it combines agile strategies with ALM principles. It targets processes and tools working together without any constraints. Agile ALM combines four major fundamentals namely collaboration, integration, automation and continuous improvement. Before agile was affixed with ALM, there was only sub-optimal collaboration making the processes and work-flows unreliable and inconsistent. Agile ALM can be distinguished as functional view and technical view. In functional view, Agile ALM focuses on assigning and tracking the implementation of requirements and effective release management. In technical view the focus is shifted to integration management and increasing the productivity.Agile ALM combines flexible processes with lightweight tools in a comprehensive and practical approach to building, testing, integrating, and deploying software. Taking an agile approach to ALM improves product quality and reduces time to market it.

1.6 Conclusion

Most of today's ALM software suites have to factor the agile principles without compromising conventional development disciplines like waterfall. Agile models suggest minimal

documentation and more implicit way of knowledge exchange. The ALM processes and suites should therefore allow work-flow creations with low artifact content [2].Even though lot of commercial vendor tools don't have an integrated support, there are still efforts being made to achieve this. The following chapter discusses about some of the commercially available popular ALM tools.

Chapter 2

Commercial ALM Software Suite

2.1 ALM Solution Providers

A good ALM package has software capable of monitoring all the phases of design, development, testing, release and other ongoing enhancements. With an effective ALM solution in place, organizations can significantly improve and adapt their software to dynamic requirements. The increasing complexity in the development process has resulted in a myriad of more sophisticated tools. Some top vendors were able to successfully harness the ALM key aspects like task tracing, issue management, release iterations into their tools. An ALM software suite can be considered as full-fledged ALM platforms that try to combine support for all application SDLC phases into one product or family of products. This chapter discusses about three such software suite **Microsoft's *Team Foundation Server(TFS)*, ThoughtWork's *Mingle* and Atlassian's *JIRA*.**

2.2 Foundation Server(TFS)

Microsoft's TFS [5] provides core collaboration functionality for development teams in an integrated product. Its major functionalities includes Project Management, Work item tracking(WIT), Version Control, Test case management, Build Automation, Reporting, lab management and feedback management. TFS is an extensible server product designed for members contributing to software project development. It can integrate with .NET APIs and has set of events that allow integration with outside tools also.

2.2.1 TFS Architecture

Logically, TFS architecture is made of two tiers Application Tier and Data Tier.

Application Tier: This tier comprises of collection of web services with which the clients can communicate. It also has a web access site through which interactions with server can happen. This eliminates the need to install a separate client suite like Visual Studio.

Data Tier: This tier makes up the back-end of TFS. It has a SQL Server database which maintains data of a TFS instance. This data is utilized for report generation. All the data stored in TFS is also stored as a backup in the SQL. The figure 2.1 shows an effective and a simple pictorial representation of TFS architecture.

2.2.2 TFS Features

Project Management: Users of TFS can monitor the progress of a project by tracking and reporting the work item status. Additionally, with TFS template concept, the project process can be synced with the development environment. A process template [1] defines a set

[1] `http://msdn.microsoft.com/en-us/library/ms364062(v=vs.80).aspx`

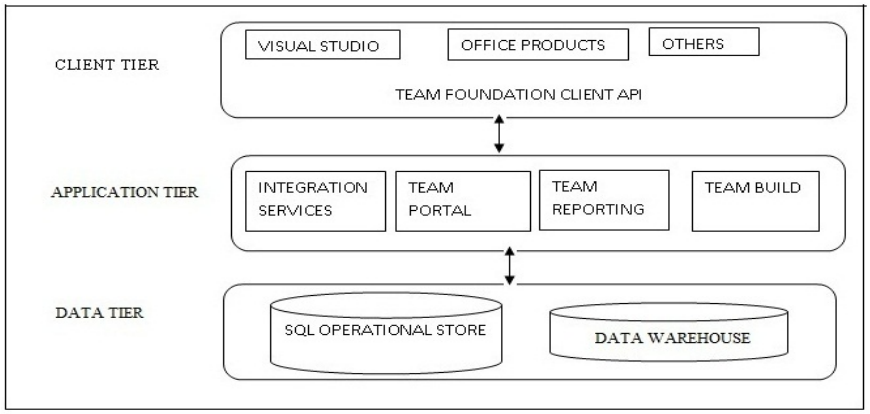

Figure 2.1: TFS Layered Architecture

of instructions for creating a new project and assigning work item types, roles, permissions, report template definitions etc. Default process templates for Framework Agile and Formal process are installed with TFS.

Version Control: TFS supports two different types of version control inbuilt control engine called Team Foundation Version Control (TFVC) and Git (included in TFS 2013 release). TFVC is a centralized repository with two different types of workspace namely server and local. To improve performance for remote clients, TFS has ability to include Proxy Servers [2].Proxy serves allow contents to be cached at a site closer to the developer thereby avoiding network latency.

Work Item Tracking: The WIT system of TFS stores and monitors bugs, requirement, scenarios, tasks, stories and other work item types. WIT is stored as XML, hence allowing users to easily create, extend and modify work items. The tracking system is also integrated other Microsoft Office products like Excel, Project etc. to enable viewing and editing.

Team Build: This build server application of TFS has two component MSBuild which

[2] `http://en.wikipedia.org/wiki/Team_Foundation_Server`

is declarative XML and Windows Workflow Foundation (WF). It augments the build engine with Team System-specific tasks and can be configured for various types of builds like scheduled builds, Continuous Integration etc. This process is also part of traceability mechanism and has the ability to report on changes occurring during each build of the project.

Reporting: Reporting infrastructure consists of data warehouse which is data cube of relational database and a SQL server Analysis Services. Since this structure is standardized, any tool which points to these data sources can report from it. The report set generated by this feature are test result and progress, project management, agile reports like Backlog overview, Burndown Charts for sprint and release cycle, velocity etc. TFS 2013 has a new feature called ”light-weight reporting”, which supports real time creation reports based on query results apart from data at warehouse and cubes.

2.3 Mingle

ThoughtWork's Mingle is an agile project management solution that offers unparalleled visibility into software initiative and unprecedented collaboration [3] among business and delivery teams. It has comprehensive project reporting mechanisms that provide the outlook needed to identify risk, make decisions and respond to changes swiftly.Mingle was developed by considering the following ALM features[4] Different working teams, best practices, card representation, realistic reporting and integrations.

[3] `http://open-services.net/software/thoughtworks-mingle/`

[4] `http://www.thoughtworks.com/products/mingle-agile-project-management/`

2.3.1 Mingle Features

Integrations: Agile organizations design their own process cycle. To accommodate different processes, mingle allows customization with an open and extensible architecture. This feature allows users to work with varied technology and ALM integrators. Even though Mingle has its own verticals of integrations available, it allows organizations to build their own workflow possibilities via APIs and Macro development tool-kit.

Reporting: Mingle's reporting features captures realistic metrics that can monitor and track accurate progress and provide forecasts. The fixed date burn-up chart generated by mingle uses historic trends to forecast work completion on date basis. The reporting feature is customizable and provides solution dynamically on-the fly. It has forecasts program objectives and alerts when the goal is not delivered as expected.

Customization: Unit of work in the project that is to be tracked and managed is represented as a card. Agile development revolves around these cards and their transition to different phases. Mingle provides easy and attractive web access card wall through which enhanced and up-to-date collaborative experience can be achieved by a simple drag-drop action. The card wall is also customizable, making it easy to create phases as per requirements. This feature makes sprint planning, prioritizing and estimating tasks easier.

Workflow: Mingle provides template for any software development methodology chosen. Workflow is the lifecycle i.e., status and transitions of each card type in the project. Workflow customization allows different teams to define their own stages and have their own iteration cycles.

2.4 JIRA

JIRA is Atlassian's issue tracking tool with support for several project management functions. It can be considered as a complete platform that supports agile practices through plugins[6] like GreenHopper, Bonfire etc. GreenHopper is built as add-on to JIRA with support for agile methodologies. Its flexible and customizable nature allows users to configure JIRA to map any hybrid agile processes based on the requirements.

2.4.1 JIRA Architecture

At a higher level of abstraction, JIRA is considered to be made of two layers namely Application services and Data storage[7].

Application services: This layer is responsible for rendering all of JIRA's business services and functions. These services comprise of business concept such as workflow and notifications. Other services like REST web service provides integration entry for other application and OGSi service provides the base framework hooks for extension.

Data Storage: This layer adds the persistence nature to JIRA. Most of the data about projects, issues, bugs are stored in a relational database. Contents such as attachments and search indexes are persisted in file system. The underlying database is transparent and can be easily migrated.

2.4.2 JIRA Agile Features

JIRA's agile feature is enabled by plugging add-on called GreenHopper. Some of the features addressed by GreenHopper are as follows[6]:

Backlog Management: This feature provides support for easy, meaningful and quick cre-

ation and ranking of backlog item in the project team. The items may be issues,bugs, tasks etc.

Capacity Management: This provides easy and flexible methods for estimating and time tracking processes for iterations using standard and custom field values.

Iteration Tracking: This feature supports easy updating of regular task status and visualizes the daily status tracking and progress of team and work.

Reporting and Charting: This aspect of Green-Hopper allows quick and detailed reporting on iteration, functionality and backlog items for both short-term and long-term planning.

Customization and Flexibility: This feature supports integration of different teams and their workflows along with various criterion based validation.

2.5 Conclusion

ALM platform successfully executes a project according to planned goals by providing collaboration, workflow and process automation. The main step in achieving a full-fledged ALM platform support is congregate ALM tools and integrate them to build a customized base tailored to the requirements. The platform thus achieved can be exposed as APIs or deployed as services. The following chapter discusses base concepts of one of such approaches namely, Web services with REST.

Chapter 3

The Concept of REST

3.1 What is REST?

REpresentational State Transfer is a collection of principles which, when applied to systems make it scalable and ubiquitous.It is a software architectural style that ignores the actual implementation and protocol syntax associated with the components and focuses only on constraints applied to the data. The constraints[8] can be summarized under the following characteristics:

Addressable Resources: The key abstraction of data in REST is a resource and it should be addressable via a URI.

Constrained Interface: There should be a small collection of methods that are modular and well-defined to manipulate the resources.

Representation Based: A resource referenced by one URI can have different representation formats based on the interaction platform. E.g.: JSON,XML etc.

Stateless: The communication context should not be stored between requests. Each re-

quest can be thought of as a state transition done by the requesting system.

Hypermedia as the engine of application state (HETEOAS): State transitions happen only through dynamically identified hypermedia resource in the server (e.g., hyperlinks). There is no assumption that an action is available beyond those described in prior representations.

3.2 Why REST?

Most of REST services run over HTTP protocol. Non-RESTful technologies like SOAP and WS-* have their own protocol stack and use HTTP only to tunnel through firewalls. So, they don't achieve the light-weight and scalability aspect associated with REST.

3.2.1 REST vs. SOAP for Services

The idea of Service Oriented Architecture (SOA) is to design systems as a set of reusable, decoupled and distributed services that can support remote access. Since these services need to be published on the network, developing a complex service should be easier and universally standard. This resulted in creation of SOAP, UDDI, WSDL and other such protocol standards. These techniques, though standardized, lack the basic support for portability (no uniform resource addressability) and have a burst of unorganized and unnecessary interfaces (no limit on the methods exposed) that makes scalability difficult. REST, on the other end, is purely dependent on HTTP techniques. So they have a URI based addressability and have only four simple methods for CRUD operations namely GET, PUT, DELETE and POST. So, when web services are designed with REST principles, their APIs are [1]

[1] `http://en.wikipedia.org/wiki/REST`

- Identified with a base URI
- Communicated with Internet media type like JSON, XML
- Use standard HTTP methods
- Hypertext links to reference state and related resource

The figure 3.1 clearly depicts the distinguishing features of SOAP based and REST based service architecture.

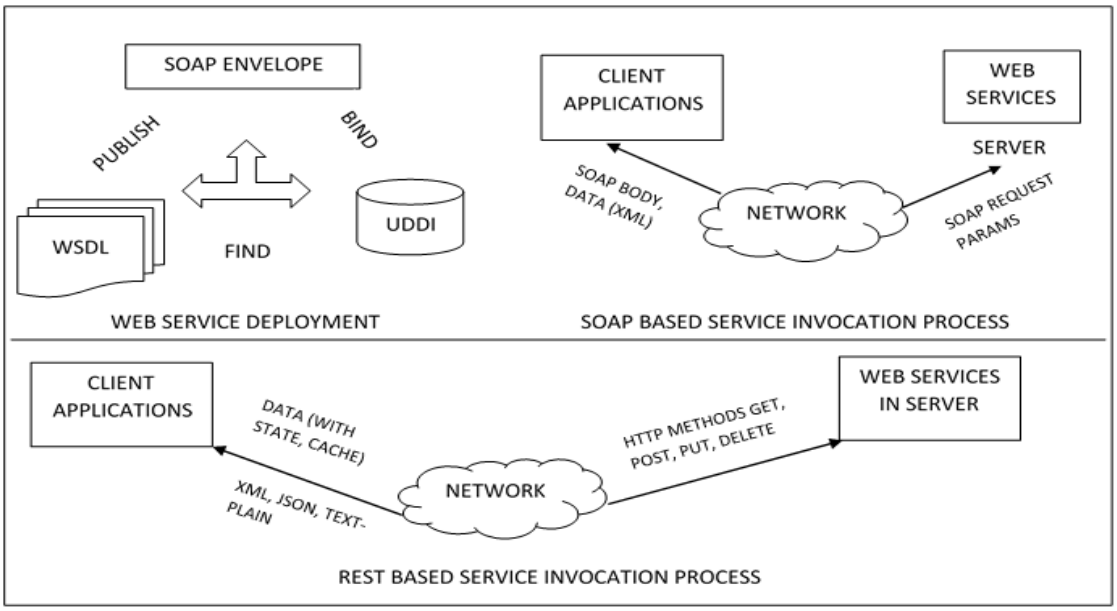

Figure 3.1: SOAP vs. REST Service Architecture

3.3 Designing RESTful Services

The steps involved in designing web services with REST architecture can be grouped as the following tasks[9].

Requirement Gathering: This steps is similar to the regular requirement identification and elucidation process involved in SDLC phase.

Resource Identification: This step involves in identifying possible resources that can be manipulated and operated. This stage is similar to identifying objects in an OOAD based design.

Resource Representation: In REST style of development, resource exchange between client and server happens using some standard representation formats like XML, JSON, and Plain Text etc. This exchange representation has to be decided in this phase.

URI Definition: The API for services are defined via URIs through which client and server will communicate by exchanging representations.

3.3.1 Rules for Good API Design

Designing of any artifact should be efficient and meaningful. So most of the designs are built by validating against some rules and standards. Following are some of the rules used for creating REST APIs in a pragmatic [10] manner.

Base URL should be simple and intuitive: A simple API is easy and self-explanatory. For enforcing this in REST design there should be only two base URLs per resource. One for fetching the entire collection of resource and other for retrieving a specific instance of the resource.

No Verbs should be in base URL. Use only HTTP verbs for resource operations: Most

of web APIs use method-driven approach for designing URLs. These URLs reflect all possible manipulations that can be executed on a resource. This in turn leads to unconstrained interfaces with no consistent pattern. So, REST style has enforced the use of HTTP CRUD operations namely GET, POST, PUT, DELETE on the base URL. This ensures limited and a consistent set of access points. For example, consider a real world resource ***books***. The table 3.1 describes the REST interfaces [10] for performing all actions on the book.

Resource	**GET**	**POST**	**PUT**	**DELETE**
/books	List of all books	Create a collection of books	Bulk update on book list	Delete all books
/books/1290	Get single book with ID 1290	N/A	Update details of book with ID 1290 (if exists)	Delete book with ID 1290

Table 3.1: Sample REST API Design

In above table, posting a single book is possible. But the book id generated is internal to the system. So, a post operation cannot happen with user specified id.

Use plural nouns and concrete names: Using concrete names show cases the APIs functionality better to a developer. Even though abstraction is necessary, at a higher level using the same abstract noun for all different resources becomes rather ambiguous. The form of noun used (plural/singular) should be consistent across all APIs.

Simplify Associations. Hide complexity under ?: Resources of a system always interact with other resources to form associations. E.g., books belong to a specific library. So to create a book or retrieve a list of books two resources namely library and book has to be related. So the GET interface will be */library/1/books*, where '1' is the library id. The URL becomes

bigger as the level of association goes deeper. So there is need to hide the complexity. The intricacies of a resource involving attributes and states that can be queried and manipulated can be defined as options using HTTP '?'. E.g., `GET/books?libraryId=1&author=Joe`. Other trivial and well known design standards include Appropriate error handling, using HTTP status codes, applying versioning, displaying fixed subset of results (paginating) etc. Some commonly used HTTP response codes are: **200 : OK, 400 : Bad Request, 500 : Internal Server Error, 401 : Unauthorized**.

3.4 REST Consumer - Jersey

Java provides lot of libraries and standards for developing a REST client like Jersey with is based on JAX-RS, RESTEasy, and Restlet etc. In this work, Jersey libraries are used for developing REST APIs. Java defines REST support via JSR Java Specification Request[2]. This specification is known as JAX-RS (Java API for RESTful Web Services). Jersey is the reference implementation of this specification. It includes REST server and core client libraries. Annotations are used to simplify the development and deployment process. Figure 3.2 lists some of the annotations used in mapping a POJO as web resource and in method parameters to pull information out of request. Jersey also provides libraries for securing clients using basic HTTP authentication and also provides injection for SSL client context. It has suitable support for transferring multi-part messages like images, file attachments etc. Jersey JSON supports set of extension modules like **MOXy, Jackson** and Java API for JSON processing **(JSON-P)**. These extension modules use three basic approaches while working on JSON representation - **POJO based JSON** binding, **JAXB**

[2] `http://www.vogella.com/tutorials/REST/article.html`

Annotations	Usage
@Path	Specifies relative path of resource class/method
@GET,@PUT,@POST,@DELETE	Specify HTTP request type
@Produces	Specifies Internet Media types used for content transfer
@Consumes	Specifies accepted media type
@PathParam	Binds method to path segment
@QueryParam	Binds method parameter to HTTP query parameter
@MatrixParam	Binds resource method to URI matrix parameter
@HeaderParam	Binds method parameter to HTTP header value
@CookieParam	Binds resource method to cookie value
@FormParam	Binds resource method parameter to a form
@DefaultValue	Specify default value if key in binding is not found
@BeanParam	Allow injection of all other annotations in single bean
@Context	Returns entire object context

Table 3.2: JAX-RS Annotations

based JSON binding and **low-level JSON** parsing and processing support.Jersey framework provides support for filters and interceptors on both client and server sides. Filters modify the headers, entity and other parameters on inbound and outbound requests and responses. Interceptors are primarily used for modifying the input and output entity streams. The interface provided for this are ClientRequestFilter, ClientResponseFilter on the client side, *ContainerRequestFilter, ContainerResponseFilter on the server side and ReadInterceptor, WriteInterceptor*.

3.5 Conclusion

The choice of using SOAP or REST for developing web service API must depend on the purpose for which the web service is designed and the environment in which it's deployed. SOAP is heavy weight but it provides utilities like built-in error handling, pre-build extensibility in the form of WS* standards etc. On the other end, REST is flexible, efficient and fast. REST can be useful when bandwidth and resources are limited, totally stateless operations are sufficient and where caching situations are cheaper. SOAP will be useful in situation where formal contracts with rigid specifications and state based operations are needed.

Chapter 4

Design and Implementation

4.1 Introduction

The focus of this work is to develop an abstract interface for agile users that is capable of communicating to the underlying Agile Management tool variants. The interface should expose methods to create, remove, track, manage and report progress of development artifacts and also provide support for other functionalities delivered by these tools. Most of these tools have their own REST based APIs and web services that can be consumed by a client. So, for creating a single integrated tool interface, it's prudent to develop just a layer of faade that can connect to the tools internal web services.

4.2 Architecture Design

Figure 4.1 depicts the overall sketch of proposed architecture. The individual components and their functionalities are explained in 4.3.

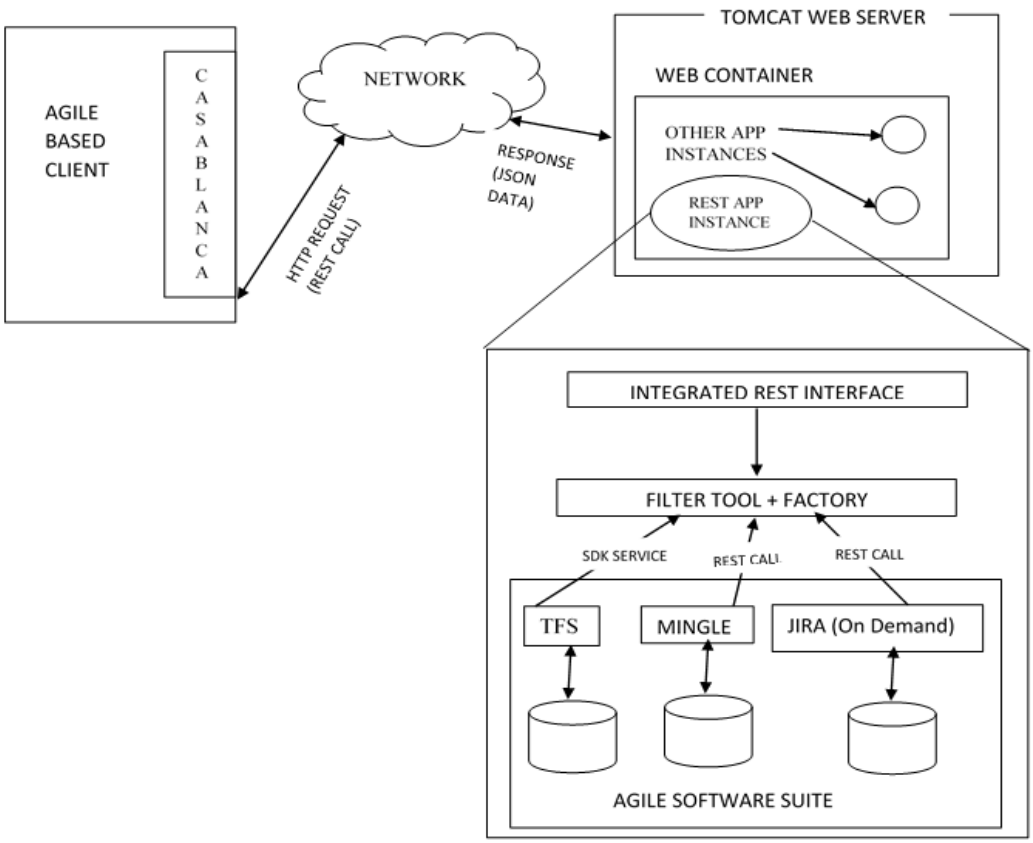

Figure 4.1: Integrated Gateway Architecture

4.3 Detailed Review

4.3.1 Agile Based Client

This component denotes the decoupled client that is capable of making HTTP REST calls to the web services deployed. The client initiates the communication by invoking an API from its agile application. Casablanca is a C++ REST SDK that acts as a middle-ware by taking the method call parameters from the client and invoking the appropriate HTTP method (GET/POST etc.). **Casablanca** adds REST capability to applications whose under-

lying platform doesn't support REST libraries. The client can range from simple HTML to complicated and customizable User Interfaces.

4.3.2 REST App Server

In this work *Apache Tomcat* is used as deployment environment. Tomcat provides web container which allows developers to host their web applications. Tomcat is both a web server (HTTP calls) and a web container (JSP compilation, servlet instantiation). So, the *localhost* system has ability to act like both client and server. The REST services designed will be deployed in this servlet container.

4.3.3 Integrated REST Interface

In agile methodology all the requirements are denoted by story cards. Each card has a well-defined requirement written in it and can be assigned to team members. A team member should be able to perform basic CRUD (Create, Retrieve, Update, and Delete) operations on use-case depicted by this card. This CRUD interface varies depending on the underlying agile tool used. So, the integrated REST interface should expose methods for

- GET single story card
- GET all story cards
- POST single story card
- POST bulk story cards [of an iteration (Sprint Planning)]
- PUT single story card

- PUT bulk story cards [of an iteration]
- Load a project (GET)
- GET all users of a project
- GET transition states

The above methods has only one REST call each from the client, irrespective of the agile management tool used. The CRUD operations are mapped to the HTTP methods as follows: GET - Retrieve, POST - Create, PUT - Update and DELETE - Delete.

4.3.4 Tool Filter

Theoretically, this layer is responsible for re-routing the calls made by the client's agile project to the appropriate tools used for its management. This layer can be viewed as a filter which does the following- takes the method call and the parameters from client, maps the project from which the call is made to the tool used for it, processes the data by performing manipulations using the underlying tool's services and present the result in a proper representation. For Mingle and JIRA OnDemand, the tool vendors provide their REST entry points which can be used. For TFS, the vendors have deployed a SDK, which can be used. Consider a sample REST call for getting a requirement (GET all the Story Cards). The call is processed as follows:

STEP 1 - HTTP Request:

The client invokes REST URL to be called for 'GET story card' from its Agile App using `http://serverID:port/projectName/api/v2/projects/BNSF/?type=card`. In this URL, *projectName* is the Rest Application name, *serverID:port* is the server system's

name and the port in which the Rest Application is deployed, *v2* is the REST API standard versioning, *BNSF* is the project from which the client called and *?type=card* is the query parameter which indicates the work item to be retrieved is a user story.

STEP 2 - Sever Tool Filter:

The server, listening at some specified port, receives this call from the client. It figures out that the project being worked upon is *BNSF*(from path URL). So, it checks the tool map, and filters the call internally to the tool TFS. (Assume BNSF is managed by TFS). So, the filter internally changes the REST URL to `/v2/tfsprojects/BNSF/?type=card` and the resource method whose *@Path* matches this call is invoked. This mapping and instantiating the appropriate service to be invoked is implemented using **SimpleFactory Pattern**. A simple Factory [11] is called directly by the class which wants to create an object (the calling class is referred as client different from HTTP client). All the classes that a simple factory returns either inherit from same parent or implement same interface. The interaction between the resource classes in accordance to this pattern is depicted in the Figure 4.2

Team Foundation Server SDK Service: TFS has provided the 'Team Foundation Sever SDK' for Java, which enables teams using Team Explorer to fully customize their development environment. This SDK [12] has a re-distributable JAR file containing the TFS API's, the native code libraries used by the TFS API, full API documentation in Javadoc format and code samples.

Mingle REST Service: Mingle API provides access to various kinds of resources in Mingle over HTTP. These resources are available as xml documents. The standard HTTP operations are used to support the CRUD on these documents. The REST APIs are provided for attachments, cards, card types (story, bug etc.), comments, transitions, users etc. A sample Mingle REST API for making a GET call to retrieve a story card information is

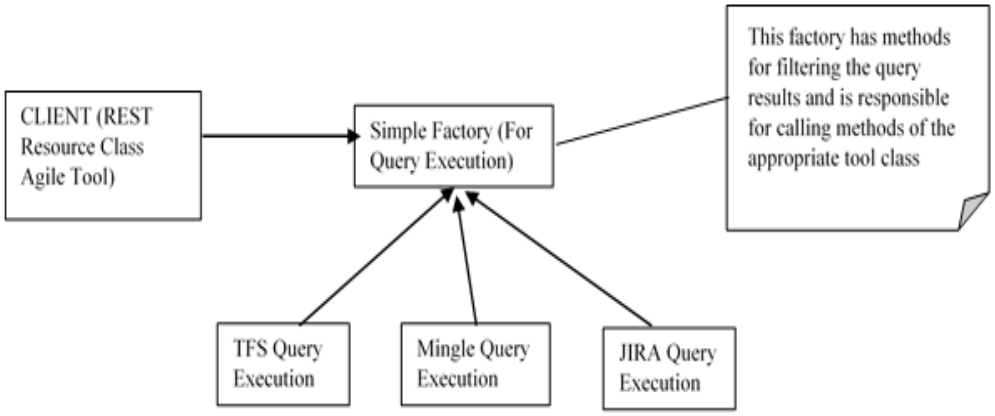

Figure 4.2: Simple Factory Design Pattern

as follows `http://name:password@your.mingle.server:port/api/v2/projects/project_name/cards/no.xml` where *name:password* is the user's authentication to the project, *no.xml* is the card number to be retrieved and 'project_name' is the agile application managed by mingle. The mingle server and port is the system identification in which Mingle is deployed.

JIRA OnDemand REST Service: JIRA's REST APIs provide access to resources (data entities) via URI paths. The application using these APIs must make an authenticated HTTP request and parse the response. The JIRA REST API uses JSON as its communication format. The API's structure is as follows `http://host:port/context/rest/api-name/api-version/resource-name`. Currently, there are two API names available **auth** (for authentication related operations) and **api** (for others).The current API version is 2, which has a symbolic version *latest*.

STEP 3 - HTTP Response:

Once, the Sever Tool Filter maps the appropriate tool, the following steps are executed if the tool is TFS then the SDK methods responsible for retrieving all the work items is

called. The resulting collection is formatted as JSON using the **Google Gson** library. If the tool is mingle, then the call is first checked for Base64 HTTP Authentication. Once it is verified the user is a valid project member, then the Rest App internally calls Mingle's vendor defined REST API. This internal response is obtained in xml format which is then parsed and re-formatted as JSON. The process is same for JIRA, except that the REST call will be directed to JIRAs internal rest API. So, if the client requests for 'GET All Cards':

If the project is managed by TFS: Make call to the TFS Java SDK methods. The returned list is of TFS internal data type definition - *WorkItem*. This collection is then formatted to a resource POJO as the response JSON object.

Else if managed by Mingle: Make another internal rest call to Mingle's vendor provided REST API with URL `https://mingle.thoughtworks.com/api/v2/projects/sampleagile/cards.xml`. The XML is then marshalled with JAXB or parsed with SAX to form JSON.The term 'sampleagile' is the project key from which the cards are retrieved.

Else by JIRA OnDemand: Make another internal rest call to JIRA's vendor provided public API with URL `http://serverID:port/rest/api/2/search?jql=project=SP`. The resulting JSON response fields are formatted to suit the client's response field.The 'serverID' is the system where JIRA is hosted on 'port'.The 'jql=project=SP' is a JIRA query which will give all the story cards of the project with key 'SP'. The resulting JSON response from any of the above call is then set as response entity. Appropriate response codes are added and the resulting HTTP response is sent back to the client.

4.3.5 Representation Format

The format used for data entity exchange is JSON JavaScript Object Notation. JSON denotes the data objects in attribute-value pairs. It is primarily used to transmit data between server and web application, as an alternative to XML [1]. JSON is language-independent and there are lot of libraries in large variety of programming languages that support JSON parsing. The official Internet Media Type for JSON is application/json. In this work, **Google's Gson** jar is used for marshalling and un-marshalling between JSON and POJO objects. The JSON data used for this work is shown in 4.3 The above json will be marshalled to a POJO

{"name": "Edit a contact"," description": "As a user of the contact book, I should be able to edit the contact details of a person","card_type": "User Story","id":"10130","project":{"name": "Sprint Planner","id":"10100"},"iteration":{"name": "Sprint 1","id":"27"}}

Figure 4.3: Sample JSON Data

object whose data members will be mapped to the json's attributes and value will be the corresponding attribute value 4.4. The control flow of this API is shown in 4.5

```
public class CardModel {
    String name;
    String description;
    String card_type;
    String id;
    ProjectModel project;
    IterationModel iteration;
}
```

```
public class ProjectModel {
    String name;
    String id;
}
```

```
public class IterationModel {
    String name;
    String id;
}
```

Figure 4.4: Sample Java Class

[1] http://en.wikipedia.org/wiki/JSON

```
Data: HTTP Request (URL with optional JSON data)
Result: HTTP Response (Status Code with Response Body)
REST_Url ← Map(project, tool);
if REST_Url found then
    if tool == TFS then
        Call appropriate SDK Methods;
        Return Result_Object ;
    else if tool == Mingle then
        Rest Call to Mingle's internal REST API;
        Marshall result XML to Result_Object;
        Return Result_Object ;
    else
        Rest Call to JIRA's internal REST API ;    // Third Tool used is JIRA
        Modify fields of result JSON to match appropriate object ;
        Return Result_Object ;
    end
    Filtered_Object ← Filter(Result_Object, Fields) ;  // Only if Applicable
    Marshal the resulting object to JSON;
    Return the JSON ;
else
    Return Error JSON Message ;
end
```

Figure 4.5: Program Control Flow

4.4 REST Service Architecture

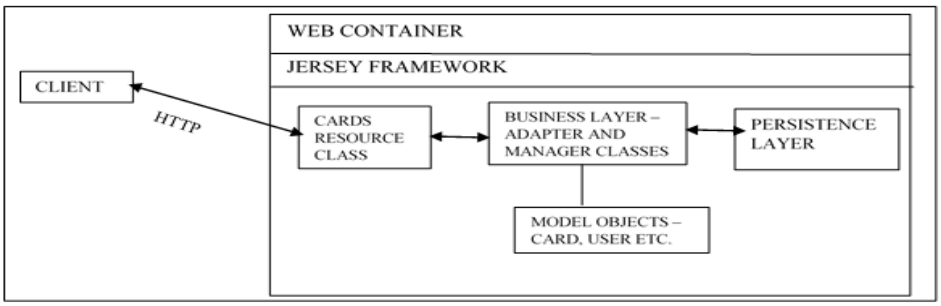

Figure 4.6: Jersey Framework

Figure 4.6 depicts the interactions of a web service developed in Jersey [16]. The resource class receives its response entity by passing the request parameters to the Business Layer. The Business layer interacts with underlying methods and persistent store (if needed) and represents the resulting data as Model Objects. Model Objects indicate the real world project artifacts like Users, Cards, and Iterations etc. The model objects is then marshalled as JSON to be sent across network to client.

4.5 Conclusion

The above architecture in extensible in the fact that any ALM tool can be integrated with it. E.g., if, at later stage, a new project is tracked and managed by Rally, the only change that needs to be done is to add methods that provide CRUD operations using Rally's internal services. The end APIs exposed to client will be the same. So, there is no burst of service end points. The integrated service end point thus follows the REST standards of being constrained and representation-based, with addressable resources.

Chapter 5

Conclusion and Future Work

The Agile Manifesto talks about interaction, collaboration and quick response to change. Adoption of this practice for software development has become mandatory in today's IT world. Consequently, the use of tools for agile project management has become a trend. A good management tool should be one which incorporates the ability to interact and trace the artifacts across various phases of SDLC and support co-operation across different teams. Achieving all these features in a single tool will result in unnecessary sophistication and the task of managing becomes cumbersome. So, this work was taken up as an initiative to provide a combined yet restricted entry point for accessing features of various ALM tools. Different vendors have provided different tool, each with its unique feature set. This work combines JIRA, Mingle and TFS tools. It can be extended further to provide support for other tools like Rally, VersionOne etc.

Appendix A

Glossary

API	Application Programming Interface
JSON	JavaScript Object Notation
POJO	Plain Old Java Object
REST	Representational State Transfer
SDK	Software Development Kit
SDLC	Software Development Life Cycle
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TFS	Team Foundation Server
UDDI	Universal Description Discovery and Integration
URI	Uniform Resource Identifier
WS*	Web Service Protocol Stack
WSDL	Web Service Description Language
XML	Extensible Markup Language

Appendix B

Integrated Gateway REST API

Sample Project Name - BNSF

Localhost Server - Apache Tomcat deployed at port 9080

REST Application Name - ALMToolSuite

API version - 2

Resource	URL	Objective
Project	http://localhost:9080/ALMToolSuite/api/v2/projects/BNSF/?type=project	**Get** details like name, Id, key of the Agile App
Card	http://localhost:9080/ ALMToolSuite /api/v2/projects/BNSF/?type=card	**Get** all the user stories of the chosen project
	http://localhost:9080/ ALMToolSuite / api /v2/ projects / BNSF/ ?type=card & value=130	**Get** a single card from the chosen project with id=130
	http://localhost:9080/ALMToolSuite /api/v2/projects/BNSF/?type=card **JSON DATA EXISTING ITERATION**: {"name": "Put Card"," description": "Card Update","card_type": "User Story","id":"130","project":{"name": "BNSF", "id":"42"},**"iteration" : { "name": "Sprint 1","id":"26"}}** **IN NEW ITERATION**:{"name": "Put Card"," description": "Card Update","card_type": "User Story","id":"130","project":{"name": "BNSF", "id":"42"},"iteration":{ **"name": "Sprint 2","id":"-1"}}**	**Put** a card (in both existing and new iteration). The difference is denoted in the JSON data passed
	http://localhost:9080/ALMToolSuite /api/v2/BNSF/projects/?type=card **JSON DATA EXISTING ITERATION**:{"name": "Post Card"," description": "Card Create","card_type": "User Story","**id":"-1**","project":{"name": "BNSF", "id":"42"},"iteration":{"name": "Sprint 1","id":"26"}} **IN NEW ITERATION**:{"name": "Post Card"," description": "Card Create","card_type": "User Story","**id":"-1**","project":{"name": "BNSF", "id":"42"},"iteration":{"name": "Sprint 2","id":"-1"}}	**Post** a new card (in both existing and new iteration). The difference is denoted in the JSON data passed

User	http://localhost:9080/ALMToolSuite/api/v2/projects/BNSF/?type=users	**Get** all authenticated users of a project
Iteration	http://localhost:9080/ALMToolSuite/api/v2/projects/BNSF/?type=iteration	**Get** all iterations (active/future) of the project
	http://localhost:9080/ALMToolSuite/api/v2/projects/BNSF/?type=iteration&value=26	**Get** all the user stories planned for an iteration whose id = 26.
	http://localhost:9080/ALMToolSuite/api/v2/projects/BNSF/?type=iteration **JSON DATA**: {"cards":[{"name": "Update An existing card" ," description " : "Iteration post","card_type": "User Story", **"id": "131"**,"project":{"name": "BNSF" ,"id":"42"},"iteration":{"name": "Sprint 3","id":"-1"}}, {"name": "Create a new card", "description": "New card in new iteration","card_type": "User Story", **"id":"- 1"**,"project":{"name": "BNSF","id":"42"},"iteration":{"name": "Sprint 3","id":"-1"}}]}	Create/**Post** a new iteration. The iteration can have existing as well as new user stories
	http://localhost:9080/ALMToolSuite/api/v2/projects/BNSF/?type=iteration **JSON DATA**: {"cards":[{"name": "Update An existing card" ," description " : "Iteration put","card_type": "User Story", **"id": "131"**,"project":{"name": "BNSF" ,"id":"42"},"iteration":{"name": "Sprint 2","id":"2"}}, {"name": "Create a new card", "description": "New card in existing iteration","card_type": "User Story", **"id":"- 1"**,"project":{"name": "BNSF","id":"42"},"iteration":{"name": "Sprint 2","id":"2"}}]}	**Put**/update an existing iteration. The iteration will have already existing cards as well as newly created story cards.
File Attachment	http://localhost:9080/ALMToolSuite/api/v2/projects/BNSF/download?value=131	A user story can have files attached with it. This URL downloads the attachments of a card whose id=131
	http://localhost:9080/ALMToolSuite/api/v2/projects/BNSF/upload	This URL posts attachments for a set of existing cards. The data is a zip file with metadata information which maps files to cards.
Transitions	http://localhost:9080/ALMToolSuite/api/v2/projects/BNSF/?type=state	This URL gives a list of all possible transitions possible from each state. It depends on the project workflow.

Bibliography

[1] K. A. Shaw, "Application lifecycle management for the enterprise," http://www.serena.com/docs/repository/alm/alm-for-the-enterpri.pdf, April 2007.

[2] D. Munshi and P. K. Reddy, "Application lifecycle management : In pursuit of best of breed," TATA Services, Tech. Rep., 2007.

[3] D. Chappell, "What is application lifecycle management?" http://www.davidchappell.com/WhatIsALM--Chappell.pdf, December 2008, sponsored by Microsoft.

[4] *Application Platform Optimization*, Microsoft Corporation, August 2007.

[5] E. Blankenship, M. Woodward, G. Holliday, and B. Keller, *Professional Team Foundation Server 2012*, ser. Wrox programmer to programmer. Wiley, 2012. [Online]. Available: http://books.google.co.in/books?id=_m-2k3LRQBMC

[6] J. Malik, *Agile Project Management with GreenHopper 6 Blueprints*, ser. EBL-Schweitzer. Packt Publishing, 2013. [Online]. Available: http://books.google.co.in/books?id=f1v2Vpuh8ZQC

[7] P. Li, *JIRA 5.2 Essentials*, ser. Professional expertise distilled. Packt Publishing, 2013. [Online]. Available: http://books.google.co.in/books?id=jraZndAsf5MC

[8] B. Burke, *RESTful Java with JAX-RS*. O'Reilly Media, 2009. [Online]. Available: http://books.google.co.in/books?id=_jQtCL5_vAcC

[9] J. Sandoval, *RESTful Java Web Services: Master Core REST Concepts and Create RESTful Web Services in Java*, ser. From technologies to solutions. Packt Publishing, Limited, 2009. [Online]. Available: http://books.google.co.in/books?id=NS6FeLs6hwMC

[10] B. Mulloy, *Web API Design Book - Crafting Interfaces that Developers Love*. Apigee, March 2012. [Online]. Available: http://info.apigee.com/Portals/62317/docs/web%20api.pdf

[11] C. Broderick, "Simple factory design pattern," http://corey.quickshiftconsulting.com/blog/first-post (Accessed: 20 May,2014).

[12] B. Harry, "Announcing a java sdk for tfs," http://blogs.msdn.com/b/bharry/archive/2011/05/16/announcing-a-java-sdk-for-tfs.aspx (Accessed: 20 May,2014).