OMSE 555/556 Software Engineering Practicum I & II

Software Architecture Document

Distributed Development Monitoring and Mining

Tom Mooney, Shailesh Shimpi, Ahmed Osman, Isaac Pendergrass

Version 1.7

04/11/2013

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Description of Versions / Changes** | **Responsible Party** | **Date** |
| 1.0 | Initial version | Shail Shimpi | 2/16/13 |
| 1.1 | Start of functional view | Tom Mooney | 3/10/13 |
| 1.2 | Add the Prediction functional view | Ahmed Osman | 3/19/13 |
| 1.3 | -Added Entity Data Model diagram  -Added interface specifications to get all  Spaces a user has access to and to get  Space data for a specific Space and user. | Tom Mooney | 4/6/13 |
| 1.4 | - Added Web Portal interface definitions.  - Added GetLastUserReport methods.  - Changed operation signatures. | Shail Shimpi | 4/9/13 |
| 1.5 | - Enhanced the Prediction module  - Added PredictionClient and PredictionClientService modules | Ahmed Osman | 4/9/13 |
| 1.6 | - Web Portal Interface definition modified | Shail Shimpi | 4/10/13 |
| 1.7 | Added Use Case View and Deployment View | Isaac Pendergrass | 4/11/13 |

**Approval Block**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Comments** | **Responsible Party** | **Date** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1. Introduction 1](#_Toc353470033)

[1.1. Purpose 1](#_Toc353470034)

[1.2. Scope 1](#_Toc353470035)

[1.3. Definitions, Acronyms, and Abbreviations 2](#_Toc353470036)

[1.4. References 2](#_Toc353470037)

[1.5. Overview 2](#_Toc353470038)

[2. Architectural Representation 3](#_Toc353470039)

[3. Architectural Goals and Constraints 4](#_Toc353470040)

[4. Use-Case View 4](#_Toc353470041)

[4.1. Actors 4](#_Toc353470042)

[4.2. Use-Case Realizations 5](#_Toc353470043)

[4.2.1. Login 5](#_Toc353470044)

[4.2.2. Request Analysis (Get Report) 7](#_Toc353470045)

[4.2.3. Retrieve Last Report 9](#_Toc353470046)

[4.2.4. Print Report 10](#_Toc353470047)

[4.2.5. Email Report 11](#_Toc353470048)

[5. Logical View 12](#_Toc353470049)

[5.1. Overview 12](#_Toc353470050)

[5.2. Interface Definitions 14](#_Toc353470051)

[6. Data View 23](#_Toc353470052)

[7. Deployment View 25](#_Toc353470053)

# Introduction

This document provides a high level overview and explains the architecture of the Distributed Development Monitoring and Mining system.

The document defines goals of the architecture, the use cases supported by the system, architectural styles and components that have been selected. The document provides a rationale for the architecture and design decisions made from the conceptual idea to its implementation.

## Purpose

The Software Architecture Document (SAD) provides a comprehensive architectural overview of the Distributed Development Monitoring and Mining system (DMM). It presents a number of different architectural views to depict the different aspects of the system.

In order to depict the software as accurately as possible, the structure of this document is based on Philippe Kruchten’s “4+1” model view of architecture [Kruchten].



The “4+1” View Model allows various stakeholders to find what they need in the software architecture.

## Scope

The scope of this SAD is to explain the architecture of the Distributed Development Monitoring and Mining system.

This document describes the various aspects of the DMM system design that are considered to be architecturally significant. These elements and behaviors are fundamental for guiding the construction of the DMM system and for understanding this project as a whole. Stakeholders who require a technical understanding of the DMM system are encouraged to start by reading the Project Proposal, Concept of Operations and Software Requirements Specification documents developed for this system [PP, ConOps, SRS].

## Definitions, Acronyms, and Abbreviations

* **Apache** – Web Server
* **ASP.NET** - Microsoft web platform
* **HTTP** – Hypertext Transfer Protocol
* [**Mono**](http://www.mono-project.com/ASP.NET)– open source implementation of Microsoft’s Common Language Infrastructure
* **WWW** – World Wide Web
* **SAD -** Software Architecture Document
* **UML** – Unified Modeling Language
* **User -** This is any user who is registered on the DMM website

## References

[PP]: Project Proposal

[SPMP]: Software Project Management Plan

[ConOps]: Concept of Operations

[SRS]: Software Requirements Specification

[MedBiquitous]: Sample SAD, <http://medbiq.org/std_specs/techguidelines/softwarearchitecture.pdf>

[Kruchten]: The “4+1” view model of software architecture, Philippe Kruchten, November 1995, <http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/Pbk4p1.pdf>

## Overview

In order to fully document all the aspects of the architecture, the Software Architecture Document contains the following subsections.

Section 2: describes the use of each view

Section 3: describes the architectural goals and constraints of the system

Section 4: describes the most important use-case realizations

Section 5: describes logical view of the system including interface and operation definitions.

Section 6: describes significant persistence elements.

Section 7: describes how the system will be deployed.

# Architectural Representation

This document details the architecture using the views defined in the “4+1” model [Kruchten]. The views used to document the DMM system are:

**Use Case view**

**Audience**: all the stakeholders of the system, including the end-users.

**Area**: describes the set of scenarios and/or use cases that represent some significant, central functionality of the system. Describes the actors and use cases for the system, this view presents the needs of the user and is elaborated further at the design level to describe discrete flows and constraints in more detail. This domain vocabulary is independent of any processing model or representational syntax (i.e. XML).

**Related Artifacts** : Use-Case Model, Use-Case documents

**Logical view**

**Audience**: Designers.

**Area**: Functional Requirements: describes the design's object model. Also describes the most important use-case realizations and business requirements of the system.

**Related Artifacts**: Design model

**Data view**

**Audience**: Data specialists, Database administrators

**Area**: Persistence: describes the architecturally significant persistent elements in the data model as well as how data flows through the system.

**Related Artifacts**: Data model.

**Deployment view**

**Audience**: Deployment managers.

**Area**: Topology: describes the mapping of the software onto the hardware and shows the system's distributed aspects. Describes potential deployment structures, by including known and anticipated deployment scenarios in the architecture we allow the implementers to make certain assumptions on network performance, system interaction and so forth.

**Related Artifacts**: Deployment model.

# Architectural Goals and Constraints

There are some key requirements and system constraints that have a significant bearing on the architecture. They are:

1. The system is meant as a proof of concept for a more complete project prediction system to be built in the future. Therefore one of the primary stakeholders in this document and the system as a whole are future architects and designers, not necessarily users as is normally the case. As a result, one goal of this document is to be useful to future architects and designers.
2. The system will be written using Microsoft .NET technologies but will use an open source RDBMS system (MySQL) for data persistence and will be deployed to a Linux webserver running Mono. These special deployment requirements require additional consideration in the development of the architecture.
3. The system must communicate with multiple third-party APIs, Assembla and Google Predictive. Defining how the system interfaces with these third-party systems is a primary concern of the architecture.
4. Section 3.3 of the Software Requirements Specification outlines a number of anticipated changes that the application could face over time. One of the primary goals of the system architecture is to minimize the impact of these changes by minimizing the amount of code that would need to be modified to implement them. The architecture seeks to do this through the use of modularization and information hiding to isolate components that are likely to change from the rest of the system.

# Use-Case View

The purpose of the use-case view is to give additional context surrounding the usage of the system and the interactions between its components. For the purposes of this document, each component is considered a use-case actor. Section 4.1 lists the current actors and gives a brief description of each in the overall use context of the system. In section 4.2, the most common use-cases are outlined and illustrated using UML use-case diagrams and sequence diagrams to clarify the interactions between components.

## Actors

|  |  |
| --- | --- |
| **User** | |
|  | The user will drive all operation of the software. No distinction is made in regards to type of user. The user interacts with all available interfaces to initiate and monitor all application operations. |
| **Assembla Client** | |
|  | The Assembla client serves to aid in user authentication and general assembla accessibility. |
| **Assembla Data Retrieval Service** | |
|  | The Assembla Data Retrieval Services main purpose is to act as the communication link between the Assembla API and our application. |
| **Assembla Data Store** | |
|  | The Assembla Data Store handles all storage and retrieval of saved Assembla information. |
| **Web Portal** | |
|  | The web portal is the main user interface for the system. |
| **Google Predictive Client** | |
|  | The Google Predictive client is the gateway into all functionality of the predictive service. |
| **Prediction Service** | |
|  | The Prediction Service is the link between our application and the Google Predictive API. |

## Use-Case Realizations

### Login

User credentials are authenticated and user is redirected to application home page.

**Figure 4.1** Login Use Case Diagram



**Figure 4.2** Login Sequence Diagram



### Request Analysis (Get Report)

User requests a report for a user-specified project and report is displayed.

**Figure 4.3** Request Analysis Use Case Diagram



**Figure 4.4** Request Analysis Sequence Diagram



### Retrieve Last Report

User requests to view the last generated report.

**Figure 4.5** Retrieve Last Report Use Case Diagram



**Figure 4.6** Retrieve Last Report Sequence Diagram



### Print Report

User requests to print report.

**Figure 4.7** Print Report Use Case Diagram



**Figure 4.8** Print Report Sequence Diagram



### Email Report

User requests a report be sent to a user-specified list of recipients.

**Figure 4.9** Email Report Use Case Diagram



**Figure 4.10** Email Report Sequence Diagram



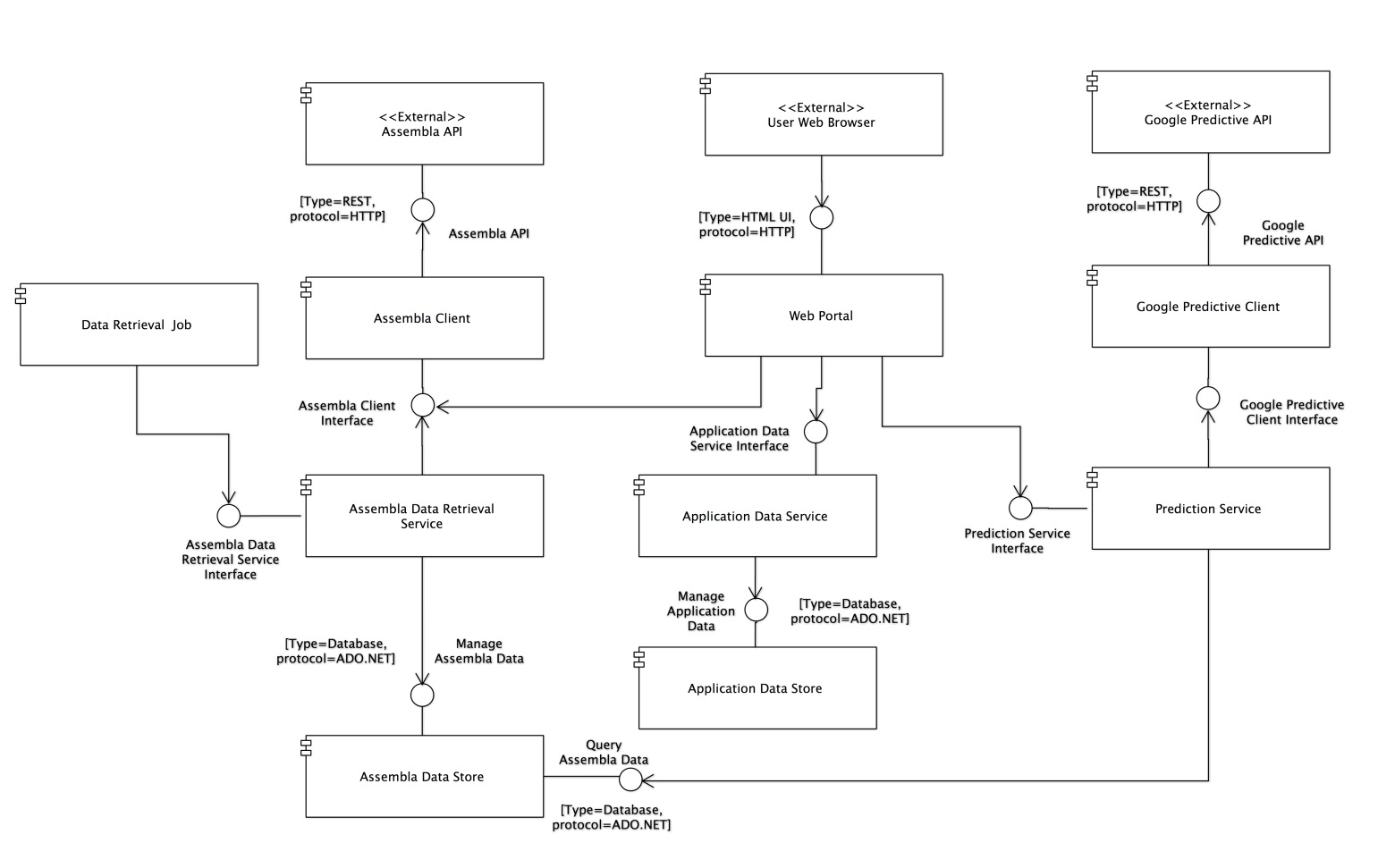
# Logical View

## Overview

The main goal of the logical view is to define the components that will make up the system and to define the interfaces through which they will communicate and interact with one another. The primary decision-making factor behind defining the system components is the need to isolate the components that are likely to change from the rest of the system. By clearly defining the interfaces of these components and hiding their internal implementations from the rest of the system, the impact of expected changes can be minimized. Section 3.3 of the Software Requirements Specification outlines the changes that are likely to be made to the system. A summary of these changes and how the logical decomposition of the architecture addresses them is as follows:

1. Changes to the Assembla API
   1. The architecture addresses this by implementing the calls to the Assembla API in an Assembla Client component (see figure 5.1). The rest of the application will communicate with Assembla only through the interface exposed by this component. Therefore any changes to the system to deal with changes in the Assembla API need only be made in the internal implementation of this component.
2. Changes to the Google Predictive API
   1. Similar to the above, this is addressed by implementing calls to the Google Predictive API in a Google Predictive Client component (see figure 5.1). Changes required to deal with changes to the Google Predictive API need only be made in the internal implementation of this component and not to the rest of the system.
3. Changes to the metrics used to construct the predictive model
   1. All business logic dealing with Google Predictive, including what metrics are sent to it to construct it’s model are isolated in a Prediction Service component (see figure 5.1). Changes to the metrics used to construct the model need only be made in this component without affecting the rest of the system.
4. Changes to the content of the prediction report generated for the user
   1. This report is generated in the Web Portal module (see figure 5.1). As such changes to its content need only be made in this module.

**Figure 5.1** Logical Component Diagram.



**Table 5.1** Element Responsibilities

|  |  |
| --- | --- |
| **Element** | **Responsibilities** |
| Assembla Client | * Provide an interface for Assembla OAuth authentication. * Handle all communication with the Assembla REST API * Provide a native .NET interface for other components to use to access the Assembla API. |
| Assembla Data Retrieval Service | * Provide an interface for retrieving data from Assembla * Manage the saving of data retrieved from Assembla |
| Assembla Data Store | * Persist data to MySQL data store. * Provide query interface to the MySQL data store. |
| Web Portal | * Present users with an HTML-based user interface accessible through a web browser. * Interact with other components in the system to allow users to authenticate with Assembla, choose an Assembla project for analysis, and analyze the chosen project. |
| Google Predictive Client | * Handle all communication with the Google Predictive API. * Provide a native .NET interface for other components to use to access the Google Predicitive API. |
| Prediction Service | * Provide an interface to get a prediction for a given Assembla project. * Provide an interface for providing training data to Google Predictive. |
| Application Data Service | * Provide an interface to save and retrieve application specific data, for example past prediction reports. |
| Application Data Store | * Persist application specific data such as past prediction reports to the MySQL data store. * Provide a query interface to the application specific MySQL data store |

## Interface Definitions

**Assembla\_Client::Assembla\_Client\_Interface**

Interface Signature

public interface IAssemblaClient

{

HttpWebResponse Milestones(string spaceId);

HttpWebResponse PublicSpaceNamesForPage(int pageNumber);

HttpWebResponse PublicSpaces();

HttpWebResponse SpaceTools(string spaceId);

HttpWebResponse TicketsByMilestone(string spaceId, string milestoneId);

HttpWebResponse UserSpaces(string accessToken);

HttpWebResponse Space(string accessToken, string spaceId);

HttpWebResponse AuthorizeUser();

HttpWebResponse GetAccessToken(string authorizationCode);

}

Operation Definitions

**Milestones(string spaceId)**

*Description*: This operation connects to the Assembla API and downloads all Milestones associated with a particular Space as identified by the spaceId parameter.

*Precondition:* A valid Assembla application ID and application secret must be sent in the request headers in order to authenticate with the Assembla API.

*Postcondition:* An HttpWebResponse object is returned to the caller. This object contains the JSON formatted response containing the Milestone data. The caller is responsible for deserializing the response.

**PublicSpaceNamesForPage(int pageNumber)**

*Description*: This operation connects to the Assembla website and downloads a specific HTML page containing publicly available space IDs. The page to download is specified by the pageNumber parameter.

*Precondition:* None

*Postcondition:* An HttpWebResponse object is returned to the caller. This object contains the HTML content of the page requested.

**PublicSpaces()**

*Description*: This operation connects to the Assembla website and downloads the HTML page that lists all publicly available space IDs.

*Precondition:* None

*Postcondition:* An HttpWebResponse object is returned to the caller. This object contains the HTML content of the page requested.

**SpaceTools(string spaceId)**

*Description*: This operation connects to the Assembla API and downloads all Tools associated with a particular Space as identified by the spaceId parameter.

*Precondition:* A valid Assembla application ID and application secret must be sent in the request headers in order to authenticate with the Assembla API.

*Postcondition:* An HttpWebResponse object is returned to the caller. This object contains the JSON formatted response containing the Tool data. The caller is responsible for deserializing the response.

**TicketsByMilestone(string spaceId, string milestoneId)**

*Description*: This operation connects to the Assembla API and downloads all Tickets associated with a particular Space and Milestone as identified by the spaceId and milestoneId parameters.

*Precondition:* A valid Assembla application ID and application secret must be sent in the request headers in order to authenticate with the Assembla API.

*Postcondition:* An HttpWebResponse object is returned to the caller. This object contains the JSON formatted response containing the Ticket data. The caller is responsible for deserializing the response.

**UserSpaces(string accessToken)**

*Description*: Returns a list of spaces that the currently logged on user has access to.

*Precondition:* User has successfully logged on through Assembla and a valid access token has been obtained.

*Postcondition:* An HttpWebResponse object is returned to the caller. This object contains the JSON formatted response containing the Spaces data. The caller is responsible for deserializing the response.

**Space(string accessToken, string spaceId)**

*Description*: This operation connects to the Assembla API and downloads all data associated with a particular Space and user as identified by the spaceId and accessToken parameters.

*Precondition:* User has successfully logged on through Assembla and a valid access token has been obtained. The user must also have access to at least one Assembla space.

*Postcondition:* An HttpWebResponse object is returned to the caller. This object contains the JSON formatted response containing the Space data. The caller is responsible for deserializing the response.

**AuthorizeUser()**

*Description*: Calls the Assembla API web service authorization method.

*Precondition:* A valid Assembla client ID must be sent in the request query string in a parameter named “client\_id”.

*Postcondition:* An HttpWebResponse object is returned to the caller. This object contains a redirect to the callback URL configured for the application with Assembla. The redirect URL will contain a query string parameter called “code” that contains the user’s authorization code. The caller is responsible for executing this redirect and capturing the authorization code.

**GetAccessToken(string authorizationCode)**

*Description*: Calls the Assembla API method to generate an access token for an authorized user.

*Precondition:* User has been authorized and an authorization code obtained through the AuthorizeUser() method. A valid authorization code must be passed in to the authorizationCode parameter.

*Postcondition:* An HttpWebResponse object is returned to the caller. This object contains a JSON formatted response containing the access token and refresh token for the authorized user to be used in subsequent Assembla API requests made on the user’s behalf. The caller is responsible for deserializing this response and handling the tokens.

**Assembla\_Data\_Retrieval\_Service::Assembla\_Data\_Retrieval\_Service\_Interface**

public interface IAssemblaDataService

{

void GetAllPublicSpaces();

void GetMilestones();

void GetTicketsForMilestones();

List<Space> GetSpacesForUser(string accessToken);

Space GetSpace(string accessToken, string spaceId);

}

Operation Definitions

**GetAllPublicSpaces()**

*Description*: This operation gets all public space IDs from Assembla and saves them to the database.

*Precondition:* None

*Postcondition:* All publicly accessible space IDs are saved to the database.

**GetMilestones ()**

*Description*: This operation gets milestone data for each of the space IDs saved in the database and saves the milestone data to the database.

*Precondition:* A list of space IDs is available in the database.

*Postcondition:* Milestone data for each space ID present in the database is saved to the database and associated with the appropriate space ID.

**GetSpacesForUser (string accessToken)**

*Description*: This operation gets a list of all spaces that a user has access to.

*Precondition:* The user has successfully logged on to Assembla and a valid access token has been obtained to pass to this method.

*Postcondition:* A list of Space objects is returned to the caller.

**GetSpaceForUser (string accessToken, string spaceId)**

*Description*: This operation gets data for a specific Space that a user has access to.

*Precondition:* The user has successfully logged on to Assembla and a valid access token has been obtained to pass to this method. The user also has access to at least one Space.

*Postcondition:* A Space object is returned to the caller.

**GetTicketsForMilestones ()**

*Description*: This operation gets ticket data for each of the milestones saved in the database.

*Precondition:* A list of milestones and associated space IDs is available in the database.

*Postcondition:* Ticket data for each milestone present in the database is saved to the database and associated with the appropriate milestone ID and space ID.

**Application\_Data\_Service::Application\_Data\_Service\_Interface**

public interface IApplicationDataService

{

void SavePredictionReport(PredictionReport predictionReport);

PredictionReport GetLastPredictionReport(string spaceId);

}

Operation Definitions

**void SavePredictionReport(PredictionReport predictionReport)**

*Description*: This operation saves a prediction report to the application data store.

*Precondition:* A prediction report has been generated by Google Predictive.

*Postcondition:* A prediction report is saved to the database.

**PredictionReport GetLastPredictionReport(string spaceId)**

*Description*: This operation gets the last saved prediction report for a particular space.

*Precondition:* At least one prediction report has been previously saved to the application data store.

*Postcondition:* A PredictionReport object is returned to the caller for further processing or display.

**Prediction\_Client::Prediction\_Client\_Interface**

Interface Signature

public interface IPredictionClient

{

public HttpWebResponse Analyze (string modelId)

public HttpWebResponse Delete (string modelId)

public HttpWebResponse Get (string modelId)

public HttpWebResponse Insert (string modelId)

public HttpWebResponse List ()

public HttpWebResponse Predict (string modelId)

public HttpWebResponse Update (string modelId)

public HttpWebResponse AuthorizeUser()

}

Operation Definitions

**HttpWebResponse Analyze (string modelId)**

*Description*: This operation connects to the Google Predictive API and get analysis of the model and the data the model was trained on associated with a particular Model as identified by the modelId parameter.

*Precondition:* A valid model ID and application secret must be sent in the request headers in order to authenticate with the Google Predictive API.

*Postcondition:* An HttpWebResponse object is returned to the caller. This object contains the JSON formatted response containing the analysis data. The caller is responsible for deserializing the response.

**HttpWebResponse Delete (string modelId)**

*Description*: This operation connects to the Google Predictive API and Delete a trained model associated with a particular Model as identified by the modelId parameter.

*Precondition:* A valid model ID and application secret must be sent in the request headers in order to authenticate with the Google Predictive API.

*Postcondition:* An HttpWebResponse object is returned to the caller. This object contains the JSON formatted response containing the delete operation status. The caller is responsible for deserializing the response.

**HttpWebResponse Get (string modelId)**

*Description*: This operation connects to the Google Predictive API and check training status of model associated with a particular Model as identified by the modelId parameter.

*Precondition:* A valid model ID and application secret must be sent in the request headers in order to authenticate with the Google Predictive API.

*Postcondition:* An HttpWebResponse object is returned to the caller. This object contains the JSON formatted response containing the status of the trained model if it is done or still in progress. The caller is responsible for deserializing the response.

**HttpWebResponse Insert (string modelId)**

*Description*: This operation connects to the Google Predictive API and begin training the model associated with a particular Model as identified by the modelId parameter.

*Precondition:* A valid model ID and application secret must be sent in the request headers in order to authenticate with the Google Predictive API .

*Postcondition:* An HttpWebResponse object is returned to the caller. This object contains the JSON formatted response containing the model data. The caller is responsible for deserializing the response.

**HttpWebResponse List ()**

*Description*: This operation connects to the Google Predictive API and list available models.

*Precondition:* A valid model ID and application secret must be sent in the request headers in order to authenticate with the Google Predictive API.

*Postcondition:* An HttpWebResponse object is returned to the caller. This object contains the JSON formatted response containing the list of models. The caller is responsible for deserializing the response.

**HttpWebResponse Predict (string modelId)**

*Description*: This operation connects to the Google Predictive API and submit model id and request a prediction associated with a particular Model as identified by the modelId parameter.

*Precondition:* A valid model ID and application secret must be sent in the request headers in order to authenticate with the Google Predictive API.

*Postcondition:* An HttpWebResponse object is returned to the caller. This object contains the JSON formatted response containing the prediction. The caller is responsible for deserializing the response.

**HttpWebResponse Update (string modelId)**

*Description*: This operation connects to the Google Predictive API and add new data to a trained model associated with a particular Model as identified by the modelId parameter.

*Precondition:* A valid model ID and application secret must be sent in the request headers in order to authenticate with the Google Predictive API.

*Postcondition:* An HttpWebResponse object is returned to the caller. This object contains the JSON formatted response containing the model update data. The caller is responsible for deserializing the response.

**HttpWebResponse AuthorizeUser()**

*Description*: This operation connects to the Google Predictive API using the secrets for the user.

*Precondition:* the user has already registered at Google Predictive to use its API.

*Postcondition:* An HttpWebResponse object is returned to the caller. This object contains the JSON formatted response containing the authnitcation ID. The caller is responsible for deserializing the response.

**Prediction\_Client\_Service::Prediction\_Client\_Service\_Interface**

Interface Signature

public interface IPredectionClientService

{

public PredictionFeedback Predict()

public int UpdateModel(string modelId)

}

Operation Definitions

**PredictionFeedback Predict()**

*Description*: This operation is doing the actual prediction by sending the data to Google Predictive Client and retrieving the results

*Precondition:* All parameters are set such as action, model name and prediction data

*Postcondition:* The model feedback is stored in PredictionFeedback object

**int UpdateModel(string modelId)**

*Description*: This operation updates the model at Google predictive with new data

*Precondition:* The model already exists at Google Predictive

*Postcondition:* The model has been updated from the data gathered from the collaboration database and it will return status of failure or success

**Web\_Portal::Web\_Portal\_Interface**

Interface Signature

public interface IWebPortal

{

HttpWebResponse AuthorizeUser(string url)

HttpWebResponse GetSpacesForUser(string accessToken)

HttpWebResponse RunPrediction(string spaceId)

HttpWebResponse GetLastUserReport(string userId)

}

Operation Definitions

**AuthorizeUser(string url)**

*Description*: User first visits DMM’s login page. Upon clicking ‘Login’; this operation redirects user to the Assembla login page for authentication.

*Precondition:* Assembla authentication API requires client Id. The Client Id will be stored into application’s configuration file.

*Postcondition:* Once user enters User Id and Password; Assembla API authenticates the user and redirects the user to DMM’s home page.

**GetSpacesForUser(string accessToken)**

*Description*: This operation gets the project space Ids associated with the user.

*Precondition:* User needs to be authenticated. User Id is passed for getting the spaces.

*Postcondition:* User associated spaces are returned back to the user interface and those get populated into the drop-down box. The return object is HttpWebResponse. The JSON formatted response will be parsed at client.

**RunPrediction(string spaceId)**

*Description*: Once the user selects the project space to be analyzed and clicks the button ‘Analyze’, this operation is triggered.

*Precondition:* The Space Id that needs to be analyzed should be selected in the drop-down box.

*Postcondition:* Google Predictor API does the analysis depending upon the past training data supplied and returns back with the analysis report that gets displayed on the user report interface.

**GetLastUserReport(string userId)**

*Description*: User is able to see the last analysis report that s/he has run previously. Clicking the menu item, calls this operation.

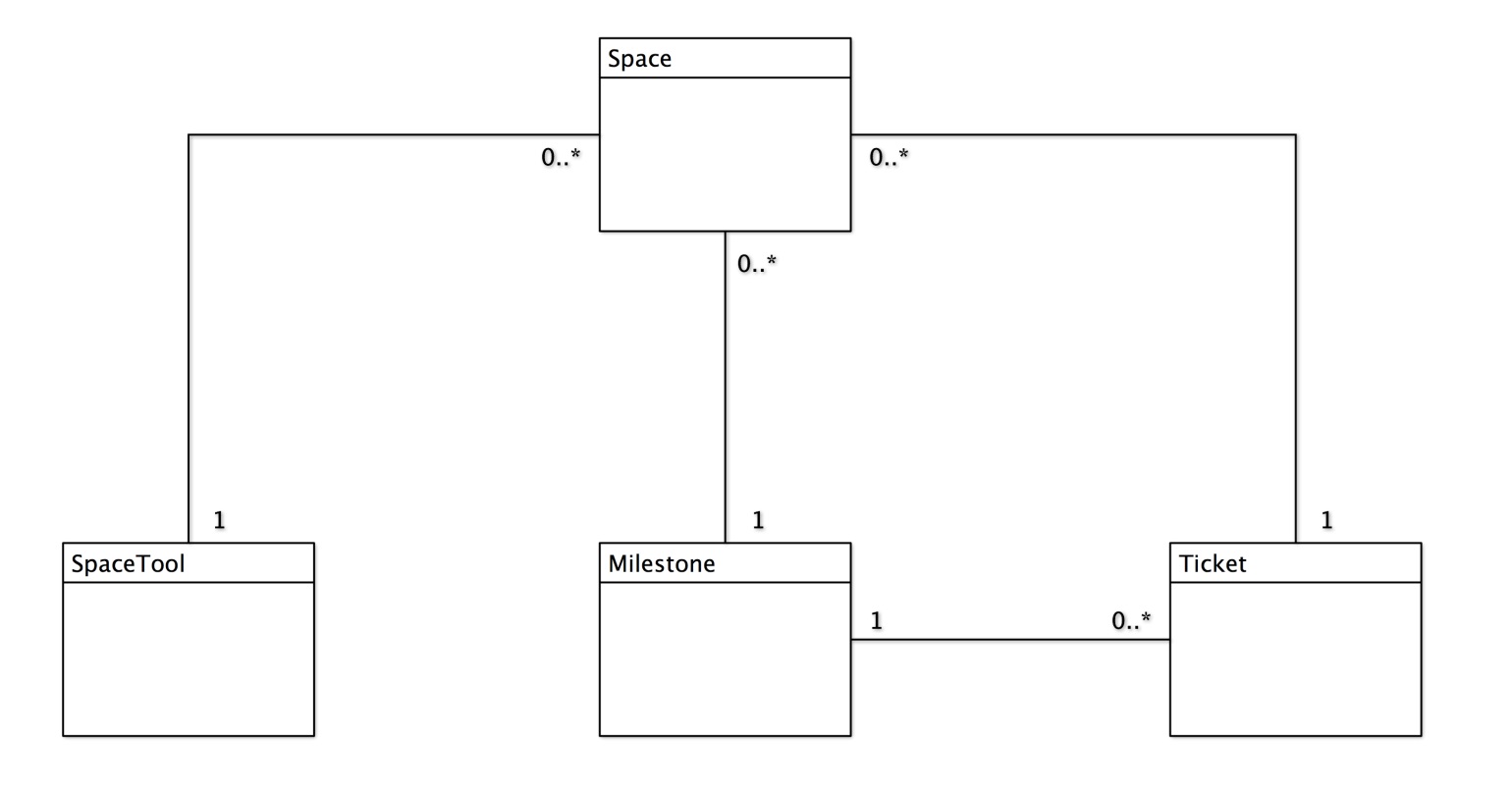
*Precondition:* User Id needs to be passed. The model logic will look for the last report for the specific user Id into the database.

*Postcondition:* The last analysis report gets displayed on user interface.

# Data View

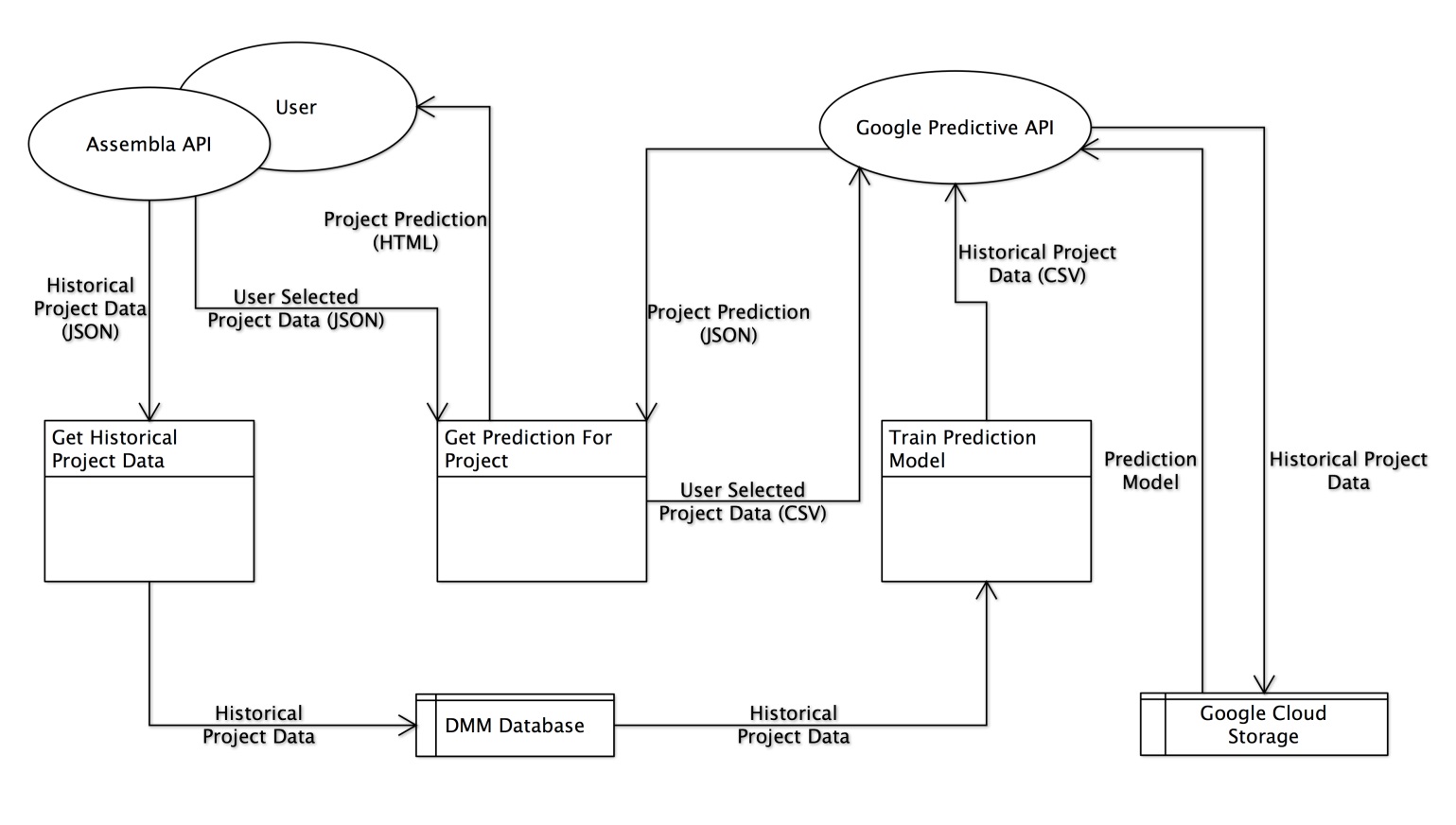
**Figure 8.1** Static Data Structure Diagram

This diagram illustrates the static data structure and relationships of the main entities that will be stored by the application in its database. Each element nominally represents a database table. Relationship cardinality is denoted with UML multiplicity notation.



**Figure 8.2** Data Flow Diagram

This diagram illustrates how data will flow between external entities and the DMM application. Ovals represent external entities, tall boxes represent points where data is processed, arrows show the direction of data flow, and short boxes represent persistent data stores.



# Deployment View

The web application will be hosted on a single physical server. An Apache webserver running a mono module will be used to serve the application pages. In addition, a MySql Server instance will also be hosted on the physical server to aid the application in persisting data.

The application will interface with external APIs (Assembla, Google Predictive), of which the deployment scenarios are not known.

The application’s deployment specifics can be seen below.

**Figure 7.1** Deployment View Diagram

