Canvas

Software Architecture Document

周植，胡子木，朱任翔，王登泰

Version 1

2019.10.21

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Description of Versions / Changes** | **Responsible Party** | **Date** |
| 1.0 | Add 1.2.3 | Zhu Renxiang | 19.10.17 |

Table of Contents

[1. 简介 1](#_Toc22545255)

[1.1. 目的 1](#_Toc22545256)

[1.2. 范围 1](#_Toc22545257)

[1.3. 定义，首字母缩写词和缩写 1](#_Toc22545258)

[1.4. 参考文献 2](#_Toc22545259)

[1.5. 概观 2](#_Toc22545260)

[2. 架构表现 2](#_Toc22545261)

[3. 架构目标与约束 3](#_Toc22545262)

[4. 用例视图 3](#_Toc22545263)

[4.1. 角色 3](#_Toc22545264)

[4.2. 用例实现 3](#_Toc22545265)

[4.2.1. Login 3](#_Toc22545266)

[4.2.2. Request Analysis (Get Report) 5](#_Toc22545267)

[4.2.3. Retrieve Last Report 8](#_Toc22545268)

[4.2.4. Print Report 9](#_Toc22545269)

[4.2.5. Email Report 10](#_Toc22545270)

[5. 逻辑视图 11](#_Toc22545271)

[5.1. Overview 11](#_Toc22545272)

[5.2. Interface Definitions 13](#_Toc22545273)

[6. 数据视图 22](#_Toc22545274)

[7. 部署视图 24](#_Toc22545275)

Software Architecture Document

# 简介

本文档提供了一个高级概述，并解释了Canvas的体系结构。

该文档定义了体系结构的目标，系统支持的用例，体系结构样式和已选择的组件。 该文档为从概念构思到实现的体系结构和设计决策提供了理论依据。

## 目的

软件体系结构文档（SAD）提供了Canvas系统的全面体系结构概述。 它提供了许多不同的体系结构视图来描述系统的不同方面。

为了尽可能准确地描述软件，本文档使用了基于Philippe Kruchten的“4 + 1”体系结构模型视图[Kruchten]结构。



“ 4 + 1”视图模型允许各种利益相关者找到他们在软件体系结构中需要的东西。

## 范围

该SAD的范围是解释分布式开发监视和挖掘系统的体系结构。

本文档描述了Canvas系统设计的各个方面，这些方面在体系结构上很重要。 这些要素和行为对于指导Canvas系统的构建以及从整体上理解该项目至关重要。 鼓励对 Canvas 系统有技术了解的利益相关者从阅读为该系统开发的项目提案，操作概念和软件要求规范文档开始[PP，ConOps，SRS]。

## 定义，首字母缩写词和缩写

* **Apache** – Web Server
* **ASP.NET** - Microsoft web platform
* **HTTP** – Hypertext Transfer Protocol
* **WWW** – World Wide Web
* **SAD -** Software Architecture Document
* **UML** – Unified Modeling Language
* **User -** This is any user who is registered on the DMM website

## 概观

为了完整记录体系结构的所有方面，软件体系结构文档包含以下小节。

第2节：介绍每种视图的用法

第3节：描述系统的体系结构目标和约束

第4节：描述最重要的用例实现

第5节：描述系统的逻辑视图，包括接口和操作定义。

第6节：描述了重要的持久性元素。

第7节：介绍如何部署系统。

# 架构表现

本文档使用“ 4 + 1”模型[Kruchten]中定义的视图详细说明了体系结构。 用于记录Canvas系统的视图为：

**用例视图**

**受众**: 系统中的所有利益相关者，包括用户终端。

**领域**: 描述代表系统某些重要的核心功能的场景和/或用例集。 描述了系统的参与者和用例，此视图提出了用户的需求，并在设计级别上进行了详细说明，以更详细地描述离散的流程和约束。 该领域词汇表与任何处理模型或表示语法无关。

**相关工件** :用户模型，用例文档。

**逻辑视图**

**受众**: 开发者

**领域**: 功能需求：描述设计的对象模型。 还描述了最重要的用例实现和系统的业务需求。

相关工件: 设计模型。

**数据视图**

**受众**: 数据专家，数据库管理员。

**领域**: 持久性：描述数据模型中在体系结构上重要的持久性元素以及数据如何流经系统。

**相关工件:** 数据模型

**部署视图**

**受众**: 部署经理。

**领域**: 拓扑：描述软件到硬件的映射，并显示系统的分布式方面。 通过在架构中包括已知和预期的部署方案来描述潜在的部署结构，我们允许实施者对网络性能，系统交互等做出某些假设。

**相关工件:** 部署模型。

# 架构目标与约束

有一些关键要求和系统约束对体系结构有重要影响。他们是：

1. 该系统旨在作为将来要构建的更完整的项目预测系统的概念证明。因此，本文档和整个系统的主要利益相关者之一是未来的架构师和设计师，而不是通常的用户。因此，本文档的一个目标是对将来的建筑师和设计师有用。
2. 该系统将使用Microsoft .NET技术编写，但将使用开源RDBMS系统（MySQL）进行数据持久化，并将部署到运行Mono的Linux Web服务器上。这些特殊的部署要求在体系结构的开发中需要额外考虑。
3. 系统必须与多个第三方API，Assembla和Google Predictive通信。定义系统如何​​与这些第三方系统接口是体系结构的主要问题。
4. 软件需求规范的第3.3节概述了应用程序可能会随着时间推移而面临的许多预期更改。系统体系结构的主要目标之一是通过最小化实现这些代码而需要修改的代码量来最小化这些更改的影响。该体系结构试图通过使用模块化和信息隐藏来做到这一点，以将可能会更改的组件与系统的其余部分隔离开来。

# 用例视图

用例视图的目的是为系统的使用及其组件之间的交互提供额外的上下文。 在本文档中，每个组件都被视为用例参与者。第4.1节列出了当前的参与者，并在系统的整体使用环境中对每个参与者进行了简要说明。在第4.2节中，使用UML用例图和序列图概述和说明了最常见的用例，以阐明组件之间的交互。

## 角色

用户：用户是使用系统的载体，一共被分为三类用户——学生、教师以及管理员。用户与操作界面进行交互并使用整个系统。

## 用例实现

### 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



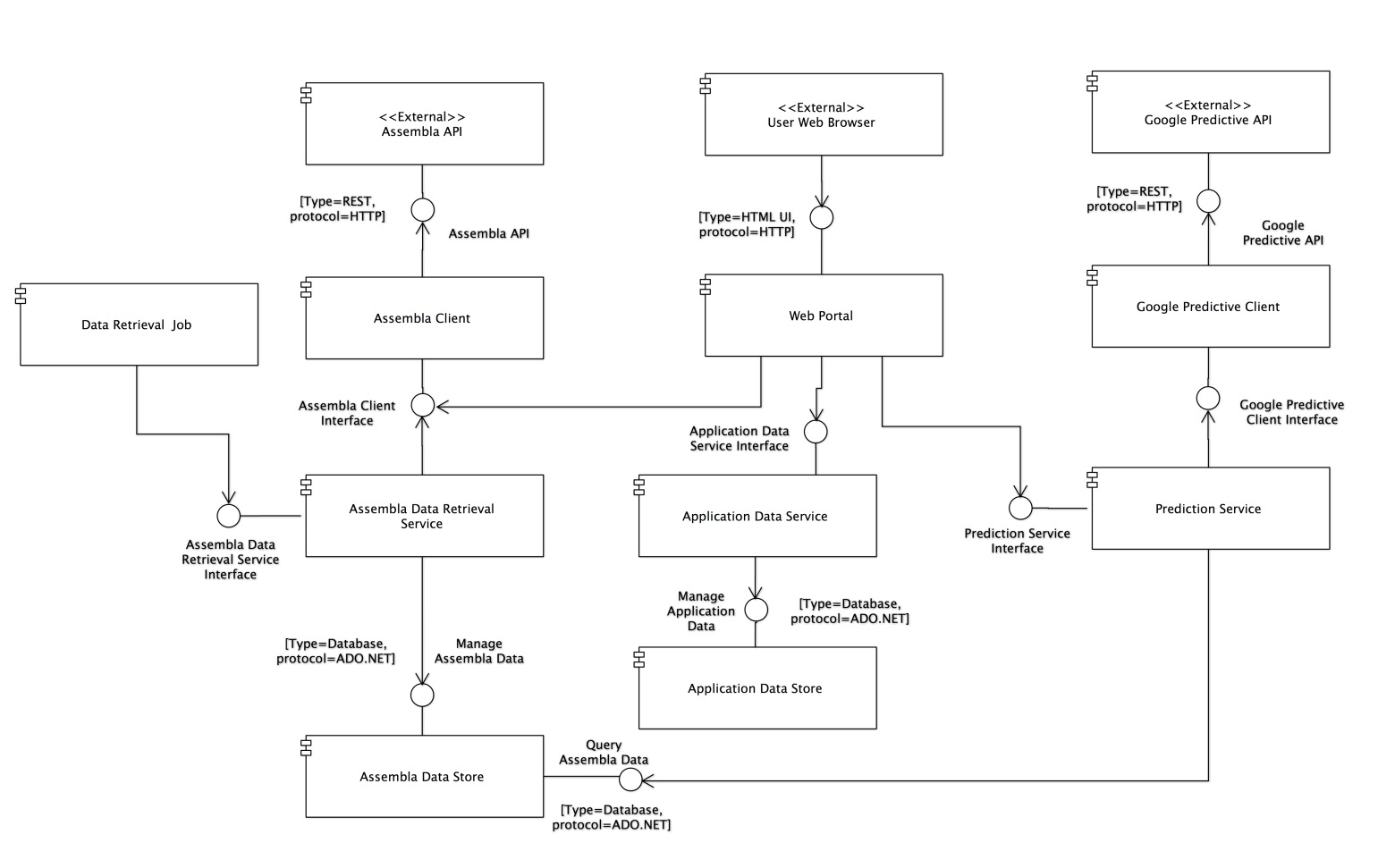
# 逻辑视图

## 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.

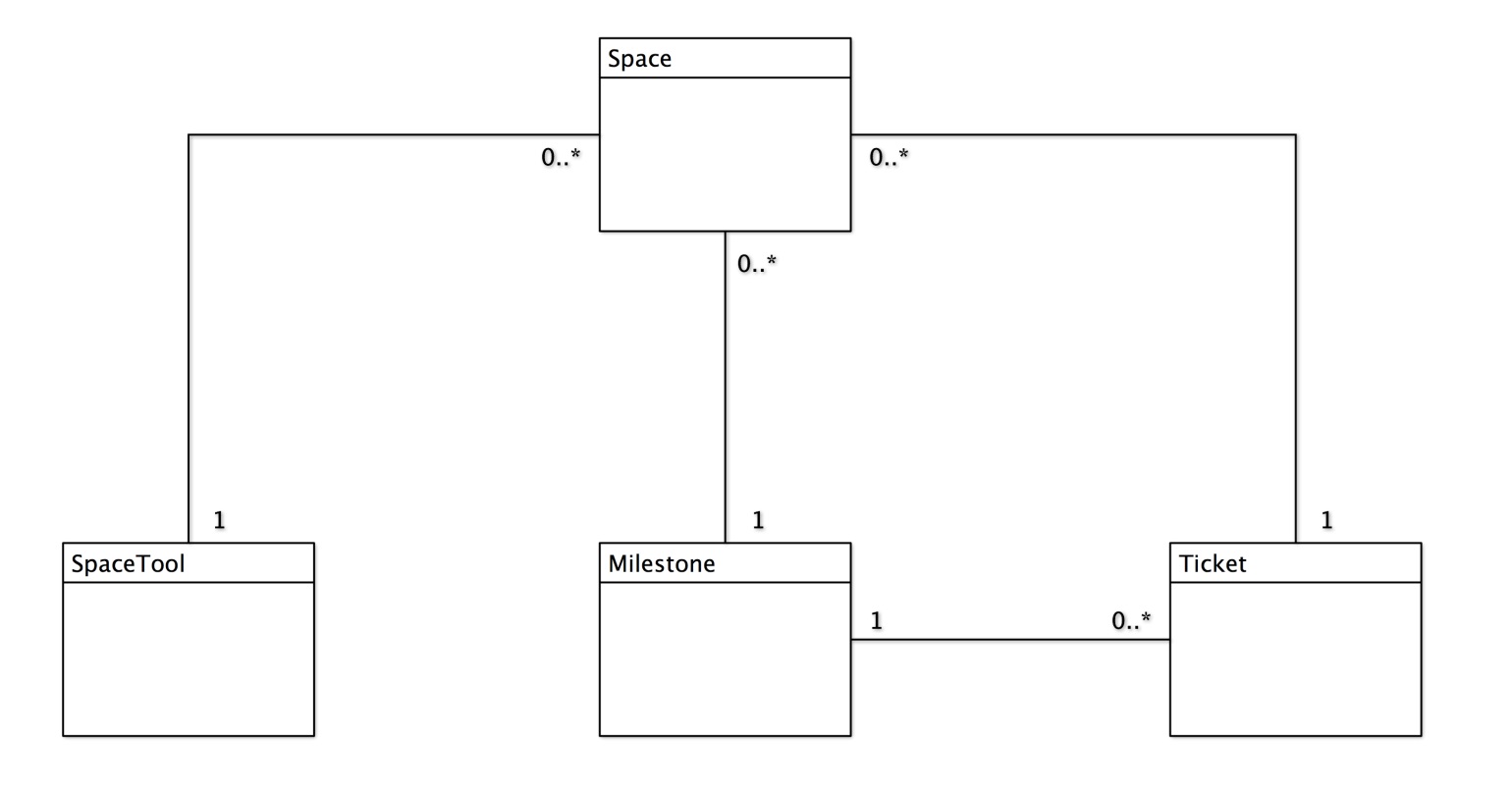
*Precondition:* User Id needs to be passed. The model logic will look for the last report for the specific user Id into the database.

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

# 数据视图

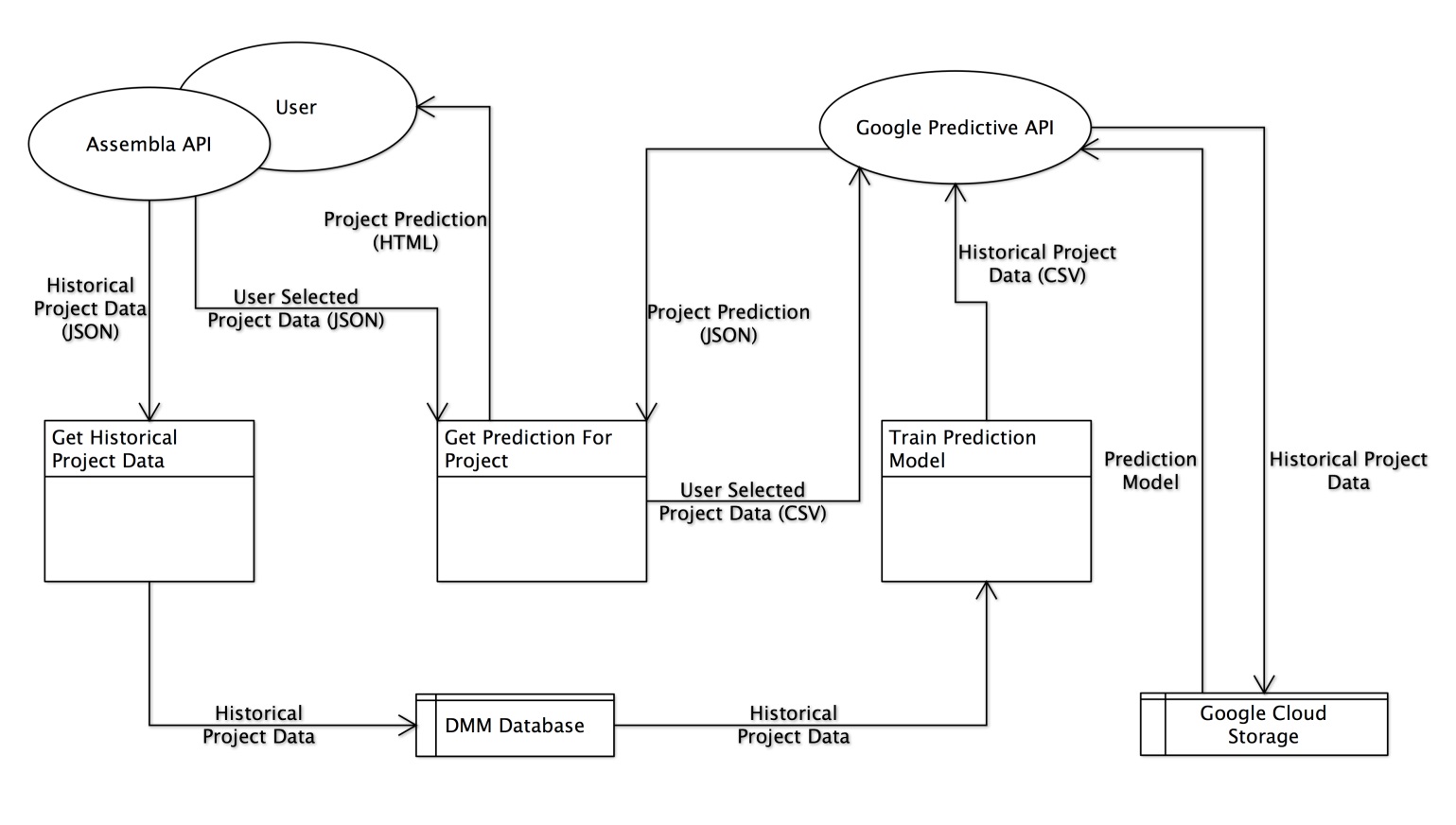
**Figure 8.1** Static Data Structure Diagram

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.



# 部署视图

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

