OLAP Manager Project

Ilya Pistryakov

Regis University

MSSE 682

Ed Harrold

March 2, 2014

**Table of contents**

Part I: Problem Description

Part II: Project Priorities

Part III: Use Case Templates

Part IV: Use Cases

Part V: Business Rules to Handle

Part VI: Domain Diagram

Part VII: Service Diagram (static)

Part VIII: Sequence Service Diagram

Part IX: Use case dynamic UML

Part X: Patterns and Algorithms to implement

Part I: Problem Description

The company has developed Online Analytical Processing tools to provide business and finance personnel with a one stop shot for their reporting and analysis needs. With expansion of the user base, the users must be able to submit access requests, which are reviewed by controllers, after which the technical team grants the access. As user base increases, users must be able to submit bugs (or potential bugs) to the technical team. In addition to being able to submit bugs, users must be able to submit change requests (new features, changes to existing features, etc). Technical personnel should be able to view a list of user submissions (bugs), list of user access requests, and change requests. Technical personnel must be able to review the tickets, update tickets, and mark tickets as closed. Technical management must be able to review change requests or requests for new features. Controllers must be able to review user access requests.

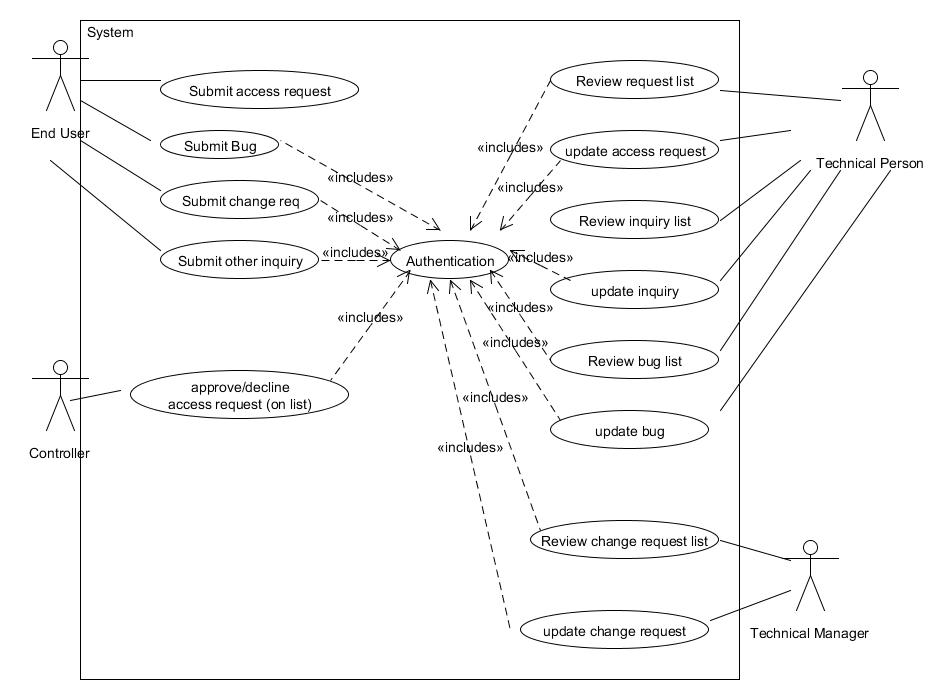
Part II: Priorities

1. End User: Submit access request (for controller review)
2. End User: Submit bug (to technical team)
3. End User: Submit change request (for technical management)
4. End User: Submit other inquiry (for technical team)
5. Controller: approve/decline access request
6. Technical Person: Review access request list
7. Technical Person: Update access request
8. Technical Person: Review inquiry list
9. Technical Person: Update inquiry
10. Technical Person: Review bug list
11. Technical Person: Update bug
12. Technical Manager: Review/update change request

Part III: Use Case diagram

Actors:

1. End user: these are people who use OLAP reporting solutions
2. Controller: these are people who use the solutions, and can approve/decline access requests
3. Technical Personnel: Receiving side of requests submitted by users
4. Technical Managers: Review/complete change requests to be changed by technical personnel



Use case 1: End User: Submit access request (for controller review)

User can submit a request for OLAP tool access without being authenticated. The web form must have links where the end user can find the form to make the request. Once the end user clicks the link, the form is displayed. The user completes the form, and clicks submit. All fields must be complete, otherwise, the form provides an error message upon submission. On the bottom, the form will have “submit” and “cancel” buttons. Only “submit” buttons are available, a user cannot update an existing request. The controller is e-mailed with a link where he/she can approve or decline the request.

Use case 2: End User: Submit bug (to technical team)

* Main Success Scenario
  + Once authenticated, the end user can submit a bug to the technical team. When the user clicks on the “submit bug” link, a form is displayed. The authenticated user completes the form, and clicks “submit”. When a form is submitted, a technical person receives an e-mail that a bug is pending his/her review.
* Alternate Scenarios (extensions)
  + End user is not authenticated. The user is informed that he/she must have access in order to submit bugs. The user is then re-directed to the main page, where he/she can submit an access request form
  + Error with submission. User is notified to contact the technical team directly through a pop up message.

Use case 3: End User: Submit change request (for technical management)

**Primary Actor:** End User

**Stakeholders and interests:**

* + End user must be authenticated
  + Requests must be stored centrally
  + Requests must be reviewable by technical management (through a list)

**Preconditions:** End user is authenticated and is on corporate network

**Success Guaranteed:** Request in the database and ready for technical management review

**Main Success Scenario:**

1. End user clicks “submit change request” on main menu
2. Link redirects the end user to the submission form
3. End user completes the form
4. Form saved and ready for technical management to review

**Extensions:**

1a. User not in system

1. User is redirected to main menu, where he/she can request access

1b. Invalid character in field

Web form provides a validation error

**Special requirement:** Compatible browser

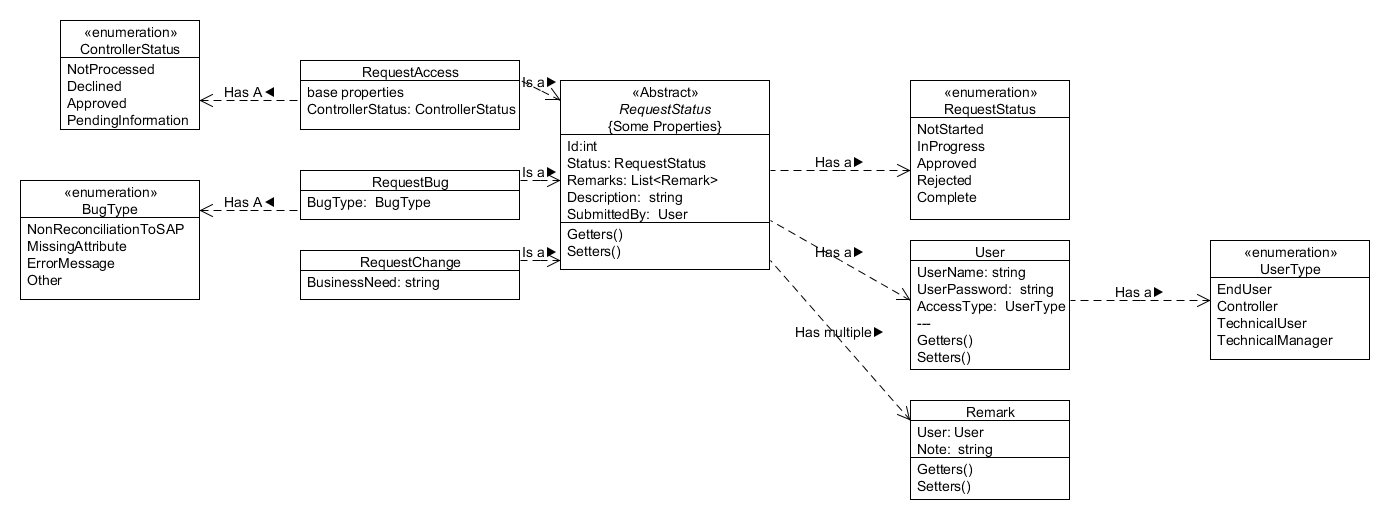
**Technology and Data Variation List:** Web form must appear the same on all browsers

**Frequency of occurrence:** Multiple times per day

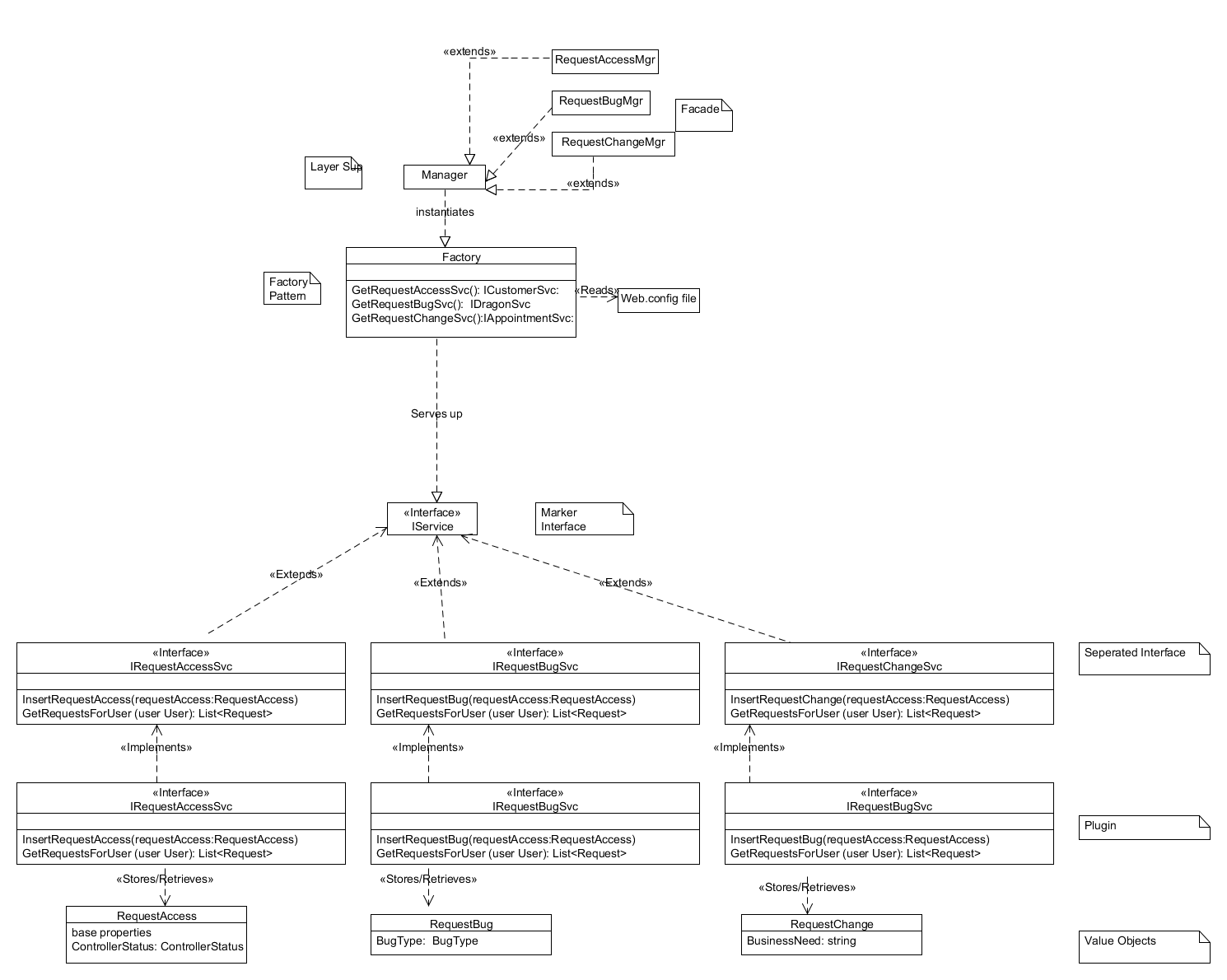
Part V: Business Rules

1. Depending on user access request, form must notify the correct controller of pending approval request
2. Upon submission and update of a request, notification must be made to the appropriate approver/technician

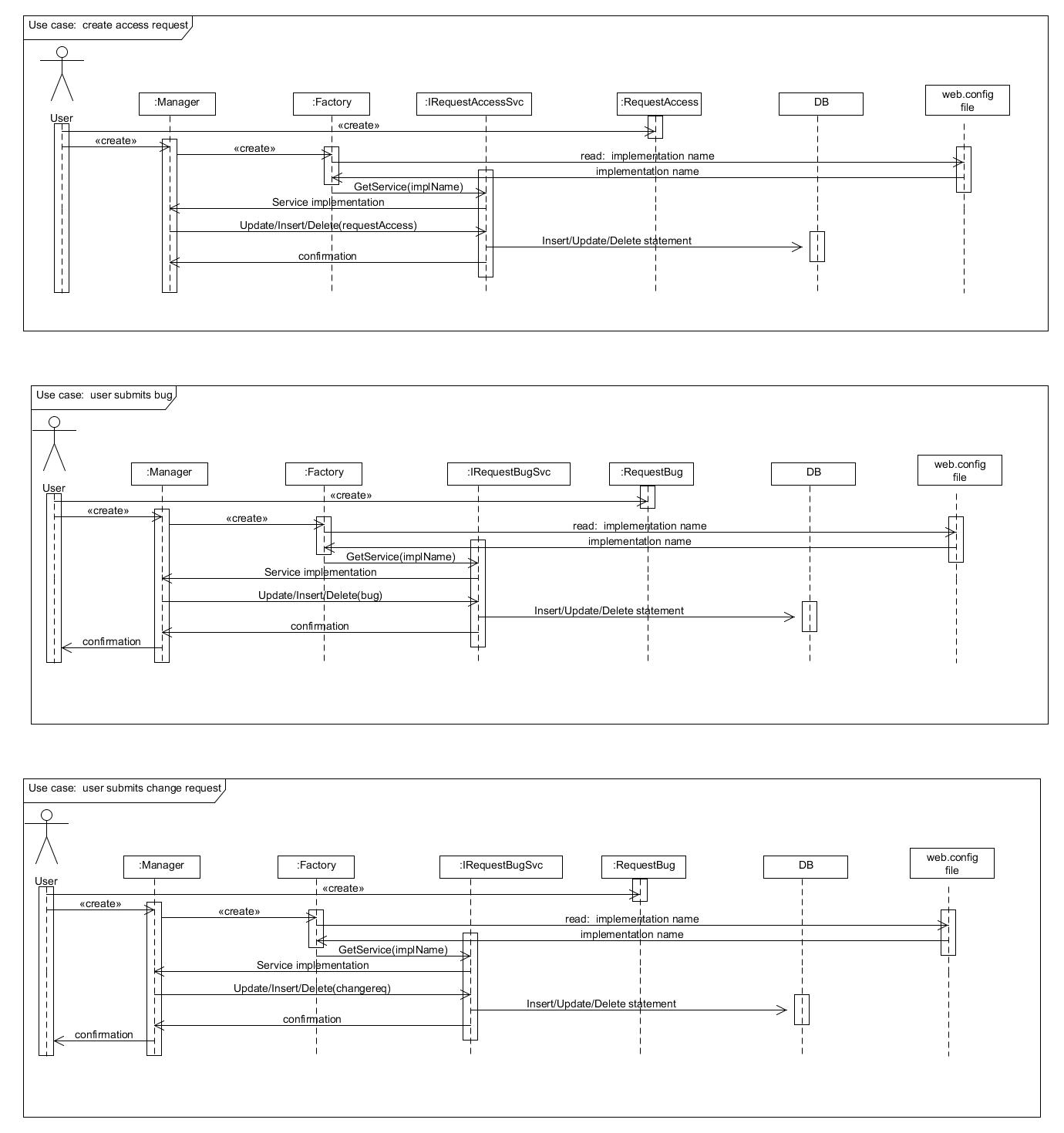
Part VI: Domain Diagram



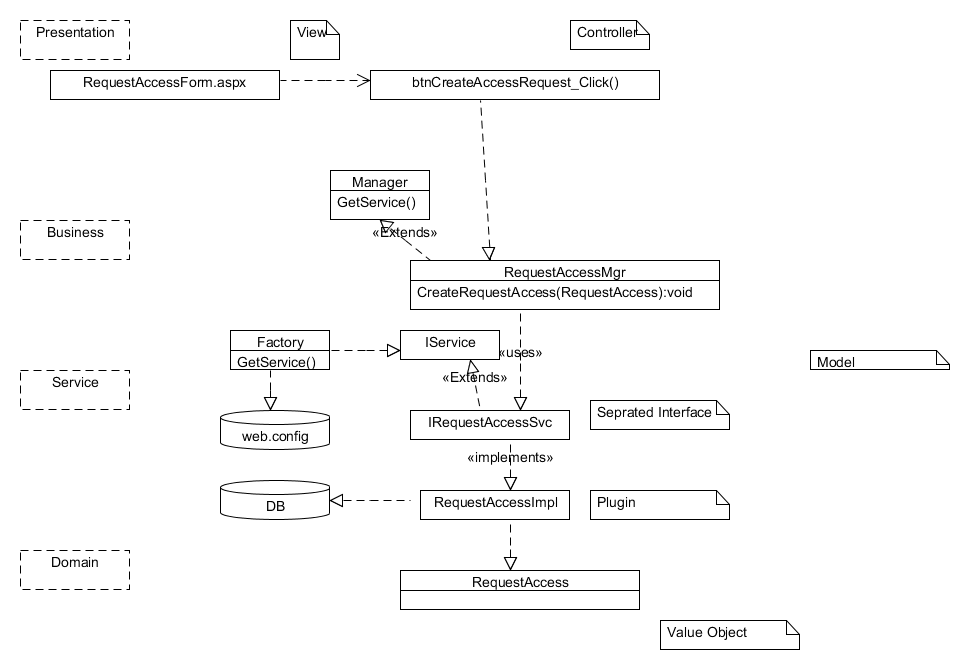
Part VII: Service Diagram (static)

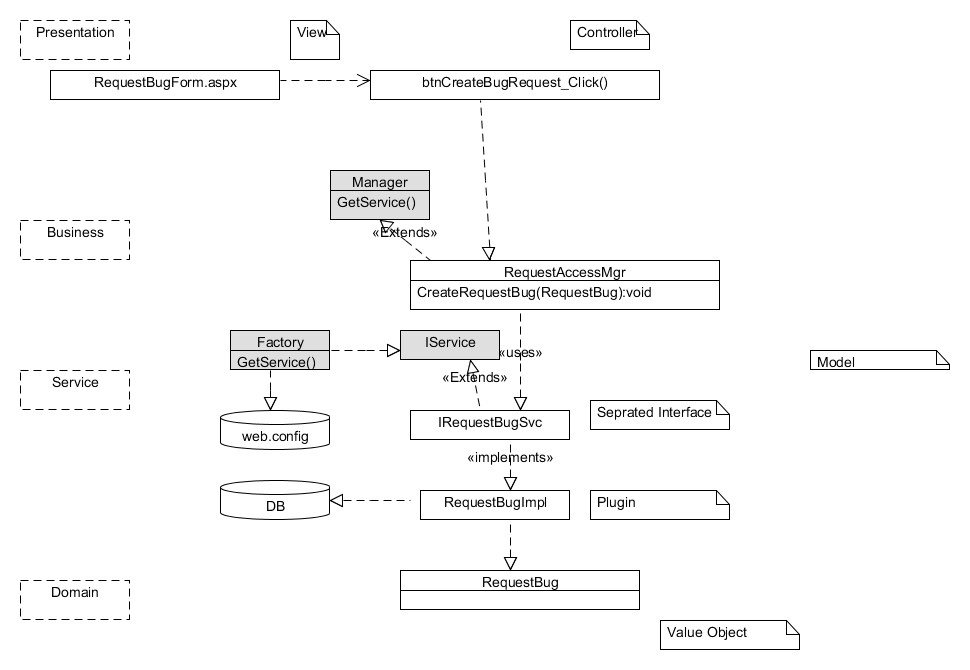


Part VIII: Service Sequence Diagrams

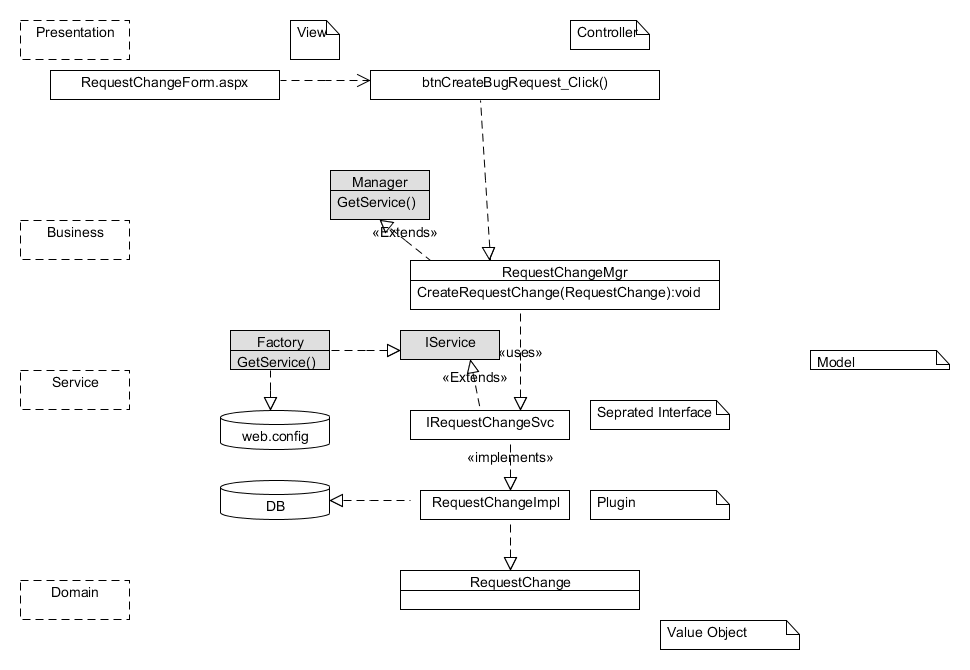


Part IX: Use Case Dynamic UML

Use case 1: End User: Submit access request (for controller review)

Use case 2: End User: Submit bug (to technical team) 

Use case 3: End User: Submit change request (for technical management)



Part X: Patterns to use

– Façade

Freeman & Freeman (2004) defined the façade pattern as an interface to a subsystem. The business layer acts as a façade to the presentation layer, while using the service layer to move value objects in and out of the application. The project will have a business layer.

– Marker Interface

A marker interface has no methods or members. The marker interface provides and defines special behavior to classes at runtime (Kalundai, n.d.). In our application, the IService interface is the parent class of the other service interfaces.

– Separated Interface

In object-oriented languages, it is wise to program to a supertype (Freeman & Freeman, 2004). In our application, we have a separate interface for each of our services, and we code the application to the interface, rather than the implementation. A separated interface provides an abstraction layer for other layers of the application. Separated interface pattern makes concrete implementations dependent on abstractions (interfaces) (Gervasio, 2012). The service layer contains the separated interfaces.

– Plugin

Plugin pattern calls for concrete implementations which are easily swappable. Code is written to the interface (Separated Interface Pattern), rather than the concrete implementations. Configuration files are used to easily swap the implementation. The service layer contains the plugin implementations.

– Factory

The Factory pattern is one of the most important, and most widely used patterns in object oriented programming. Systems (software) must be designed with change in mind, and the factory pattern allows us to build software which can be easily changed (Purdy, 2002). The factory pattern lets subclasses decide which class to instantiate, and let a class defer instantiation to subclasses (Freeman & Freeman, 2004). The service layer contains the factory.

– Value object

A value object is an object which follows value semantics (Fowner, n.d.). Quite simply, it is a predefined way to store data. Dates, integers, money, or more complex data (Customer, Account, etc) can all be considered value objects. In our application, the domain layer, along with any custom enumerations (ENUM) is the “value objects” pattern.

References

Fowler, M. (n.d.) *Value Object*. Retrieved from <http://martinfowler.com/bliki/ValueObject.html> on February 5th, 2014

Freeman, E., Freeman, E. (2004). Head *First Design Patterns.* Sebastopol, CA: O’Reilly Media

Gervasio, A. (2012, June 13). Decoupling Interfaces from Implementaitons –Using Separated Interfaces. *PHP Design Patterns*. Retrieved from <http://www.sitepoint.com/decoupling-interfaces-from-implementation/> on February 5th, 2014

Kalundai, J. (n.d.). Java Marker Interface. Java Papers. Retrieved from <http://javapapers.com/core-java/abstract-and-interface-core-java-2/what-is-a-java-marker-interface/> on February 5th, 2014.

Purdy, D. (2002). *Exploring the Factory Design Pattern*. Microsoft Developer Network. Retrieved from <http://msdn.microsoft.com/en-us/library/ee817667.aspx> on February 5th, 2014.