Review Document – Requirements Document

David Klopp Christian Stricker Markus Vieth
Alisha Klein

January 13, 2016

Contents

1 Introduction

Der Gruppe 11 sind einige Fehler beim Formulieren des Low-Level Documents unterlaufen, welche im Folgenden behandelt werde.

2 Low-Level-Design

2.1 Klasse `Object`

Problem:

Der Klassenname `Object` ist irreführend. Es könnte die Java Klasse `Object` gemeint sein, von der abgeleitet werden soll.

Lösung:

Ändere den Klassennamen in `SystemObject` um zu verdeutlichen, dass eine eigene implementierte Klasse gemeint ist und um ungewollte Nebeneffekte zu vermeiden.

2.2 Sicherheit

Problem:

`ClientCrypter` und `ServerCrypter` sind nicht notwendig, da eine verschlüsselte HTTPS-Verbindung zwischen Client und Server und verschiedenen anderen Servern aufgebaut wird. Die `Crypter` würden außerdem innerhalb vom Server miteinander verschlüsselt kommunizieren, wodurch die Performance erheblich sinken würde.

Lösung:

`ClientCrypter` und `ServerCrypter` können, ohne größere Nachteile vollständig entfernt werden.

2.3 Sequenzdiagramm

Problem:

Die Funktion `getModel(id)` gehört zu der `PackageManager` Klasse und nicht zu dem `ModelInterface`. Des weiteren ist das Sequenzdiagramm zur `Package` Generierung in diesem Punkte ebenfalls falsch, da die Lebenszeit der Objekte fehlerhaft eingetragen wurde. (z.B Der `PackageManager` existiert vor dem `ModelInterface` und einige Klassen existieren, bevor diese erstellt werden.)

Lösung:

Die Funktion `getModel(id)` sollte im `PackageManager` implementiert werden. Die Lebenszeit der Objekte muss angepasst werden, so dass der `PackageManager` vor dem `ModelInterface` existiert.

2.4 Weka-Anbindung

Problem:

Eine Anbindung der Weka-Library an das System fehlt vollständig.

Lösung:

Die Weka-Library müsste derart an das System angebunden werden, dass eine Schnittstelle zur Verfügung steht die auf Basis der Klassen des Servers operieren kann.

2.5 Upload und Download

Problem:

Datensätze und Algorithmen können nicht hochgeladen und erstellte Pakete nicht heruntergeladen werden.

Lösung:

Es müssen Funktionen im `PackageManger` und den anderen entsprechenden Klassen bereitgestellt werden, welche die Funktionen für den Datei Up/Download implementieren.

2.6 Rechte

Problem:

Die Rechte für den Zugriff auf Pakete oder Algorithmen werden nicht überprüft und der Login-Mechanismus fehlt.

Lösung:

Die Pakete müssten ein Attribut für die Zugriffsrechte bereitstellen. Über den aktuell angemeldeten Benutzer könnten nun die Berechtigungen auf die jeweiligen Pakete geprüft werden.

2.7 Administrator

Problem:

Eine eigene Klasse für den "‘Administrator"’, der sich von den üblichen Usern abhebt, um seine Rechte und Funktionen bereitzustellen und zu organisieren, fehlt.

Lösung:

Es sollte eine Klasse "‘Administrator"’ implementiert werden, die von `User` erbt oder eine andere Methode, welche eine Identifizierung des Administrators erlaubt. Die Methoden für den Zugriff auf Dateien oder Pakete müssten für diese "‘Administratoren"’ entsprechend angepasst werden.

2.8 Parameter für Algorithmen fehlen

Problem:

Durch die vorhandenen Klassen wird keine Option bereitgestellt, um festzustellen mit welchen Parameter ein Algorithmus umgehen kann.

Lösung:

Man könnte der Klasse `Algorithm` ein Attribut hinzufügen, welches die unterstützten Parameter in einer Liste speichert.

2.9 Serialisierbarkeit

Problem:

Die Klassen `Package` und `Model` sind nicht serialisierbar, obwohl dies explizit im Requirements-Document gefordert ist.

Lösung:

Die Klassen `Package` und `Model` müssen das Interface `Serializable` implementieren und ihre Attribute entsprechend anpassen. Des weiteren sollte eine Methode bereitgestellt werden, die diese Objekte in eine Datei schreibt, um so weiter Operationen mit diesen Dateien zu ermöglichen.

2.10 Datensatz

Problem:

Die Klasse `Dataset` speichert eine Liste Parameter, obwohl diese nicht benötigt wird.

Lösung:

Man lösche das Attribut `Parameter`.

2.11 Generics

Problem:

Im gesamten Dokument werden Generics falsch verwendet. Wenn sie eingesetzt werden, z.B. in `Model`, anschließend wird nicht definiert, um welche Klasse es sich handelt, da `Model` selbst nicht parametrisiert ist. Parametrisierte Klassen wie List dagegen werden durchgehend nicht parametrisiert, welches gegen gängigen Standards (siehe UR024) verstößt.

Lösung:

Man passe die Generics in allen Fällen an.

2.12 Modell

Problem:

Der Klasse `Model` fehlen die Attribute, um sowohl Parameter, als auch Algorithmus und Datensatz zu speichern, welche an seiner Erstellung beteiligt waren. Diese Information wird benötigt, um die Funktion zu gewährleisten, dass der Nutzer informiert wird, wenn dieser ein Modell wiederholt erstellen möchte, welches unter genau den gleichen Bedingungen bereits erstellt wurde und er die Berechtigung besitzt es zu lesen.

Lösung:

Man füge die benötigten Attribute hinzu.

2.13 ID

Problem:

Die Parameter `id` der Klasse `Database` sollte vom Typ `URI` sein und nicht vom Typ `long`, um die Kompatibilität mit der Datenbank zu gewährleisten.

Lösung:

Man ändere das Attribut in den entsprechenden Methoden.

2.14 Konstruktor

Problem:

Zu keiner Klasse ist ein Konstruktor angegeben, welcher spezifiziert, wie eine Klasse erzeugt werden kann. Klassen wie `User`, welchen teilweise Setter für einige Attribute fehlen, müssten allerdings einen anderen, als den Standard-Konstrukter benötigen.

Lösung:

Man ergänze den Konstruktor in den entsprechenden Klassen.