Zentrale Architekturdokumention

Software Architektur

Software zur Verwaltung von Laufdaten

Milos Babic & Tim Schmiedl

15. Mai 2013

Versionsübersicht

Tab. 1: Übersicht der Veränderungen und Versionen

Datum	**Bearbeiter**	**Änderungen**
08.04.13	Milos Babic, Tim Schmiedl	Erste Abgabe des Dokuments
18.04.13	Milos Babic	• Stakeholder in Tabelle eingetragen • Randbedingungen eingetragen • Kontextdiagramm um Beschreibung erweitert
05.05.13	Milos Babic	Refactoring: Klassendiagramm
15.05.13	Tim Schmiedl	Randbedingungen, generelle Überarbeitung

Gliederung

Verzeichnis der Abbildungen

Verzeichnis der Tabellen

1. Einführung und Ziele

Im Folgendem wird das zu erstellende Softwaresystem "RunningEasy" als "System" bezeichnet.

1.1. Aufgabestellung

Zur Verwaltung von Laufveranstaltung soll ein neues Softwaresystem – RunningEasy – erstellt werden.
Es soll Veranstaltungen samt ihren Teilnehmern und Ergebnissen erfassen und auswerten.
Der Läufer kann sich an Veranstaltungen anmelden und Ergebnisse anschauen.
Auch soll der Läufer per SMS oder Email benachrichtigt werden, falls die Startgebühr noch nicht überwiesen wurde.
Zudem sollen Schnittstellen zu externen Systemen wie Bankanwendungen, SMS/Email-Service und Datenimportierung vorhanden sein.

1.2. Architekturziele

Anhand des Lastenhefts (Blatt 1 SWA-Praktikum) sind folgende funktionale und nicht funktionale Anforderungen gegeben:

Funktionale Anforderungen:

- Anlegen von Laufveranstaltungen
- Anmelden eine Läufers zu einer Veranstaltung
- Überweisen der Startgebühr
- Anmeldung bei einer Veranstaltung zurückziehen
- Liste der gemeldeten Läufer, die die Startgebühr (noch) nicht überwiesen haben
- Erinnerung per E-Mail oder SMS zur Gebührbezahlung
- Vereinszugehörigkeit eines Läufers ändern
- Erstellen von Startlisten
- Anzahl von Meldungen zu einem bestimmten Tag vor Anmeldeschluss anfordern
- Zuweisen von Startnummern zu den gemeldeten Teilnehmern
- Importieren der Laufzeiten einer Veranstaltung, die von einem externen Zeitmesssystem erfasst werden
- Erstellen von Ergebnislisten
- Liste der Starter, die aufgegeben haben und nicht im Ziel angekommen sind
- Disqualifikation eines Läufers
- Zeitkorrektur: eine erfasste Laufzeit manuell korrigieren können
- Versand der persönlichen Laufzeit und Platzierung per SMS
- Liste aller Ergebnisse für einen bei mehreren Veranstaltungen gestarteten Läufer
- Zahlungseingänge von einer Bankanwendung für ein Konto anfordern
- Zwischenzeiten aus einem externen Laufzeitsystem anfordern
- Datenimport von Vorgängersystemen

Nicht funktionale Anforderungen:

- Erinnerung per E-Mail oder SMS zur Gebührbezahlung nach 5 Tagen
- Datenimport von Vorgängersystemen in serialisierter Form

1.3. Stakeholder

Tab. 2: Stakeholder

Stakeholder	Anforderungen
Läufer/Teilnehmer	• fordern unkomplizierte Anmeldungen an Veranstaltungen . • fordern eine übersichtliche Statistik der Veranstaltungen und Läufer bzw. Teilnehmer.
Administrator	• Einfache Bedienung zur Änderung/Verwaltung von Informationen zu Läeufer, Laufzeiten, Vereinen, Teilnahen und Ergebnissen. • Schnittstellen zu externen System leicht wartbar und erweiterbar
Veranstalter	• fordern Erstellung und Konfigurierung von Veranstatlungen, und Zahlungskontrolle der Teilnehmer.

Externe Schnittstellen	• müssen kompatibel zu dem System sein
	• Dazu gehören u.A.
	  • Bankanwendung
	  • Laufzeitsystem
	  • Benachrichtigungsdienst

Altdaten	• müssen im vorhanden Format importiert werden können

2. Randbedingungen

Die Randbedingungen ergehen sich durch jeweils durch die in den Aufgabenblättern gestellten Anforderungen.
Übergreifend für das gesamte Projekt sind folgende Randbedingungen vorgegeben.

Auf **technischer** Ebene:

- Java als Programmiersprache
- JUnit als Testunterstützung
- Eclipse als IDE
- JPA 2.0 als Datenbank-Schnittstelle

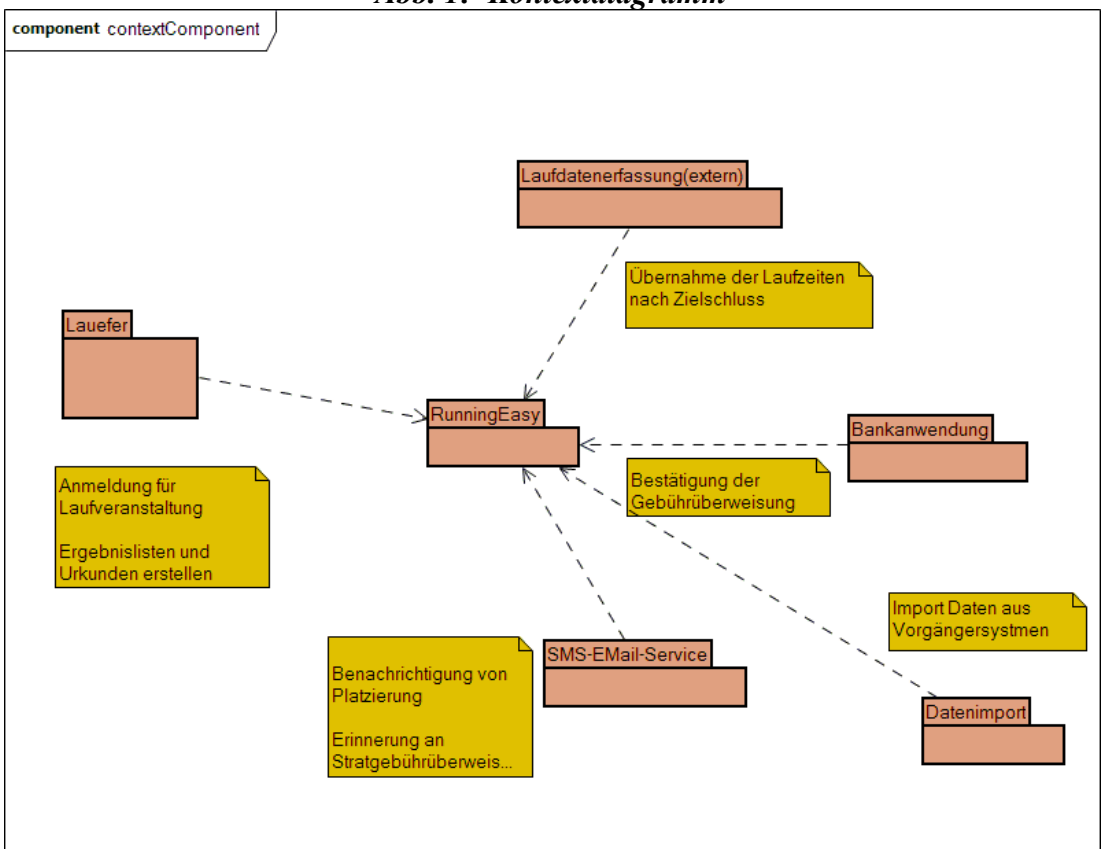
Auf **organisatorischer** Ebene:

- Versionsverwaltung
- feste Termine für die Abgaben

3. Kontextsicht

Im folgenden Diagramm ist das System im Kontext mit angrenzenden Systemen und Akteuren dargestellt.

Abb. 1: Kontextdiagramm



Der Läufer benutzt RunningEasy um sich für eine Laufveranstaltung anzumelden/stornieren, seine Ergebnisse anzufordern.

Das SMS-EMail-Service versendet Emails/SMSs, die vom RunningEasy erstellt wurden an die Läufer oder Teilnehmer gerichtet sind.

Der Datenimport besitzt Daten aus Vorgängersystem bzw. alten Verwaltungssystem für Laufveranstaltungen. Diese sollen in das RunningEasy import werden können.

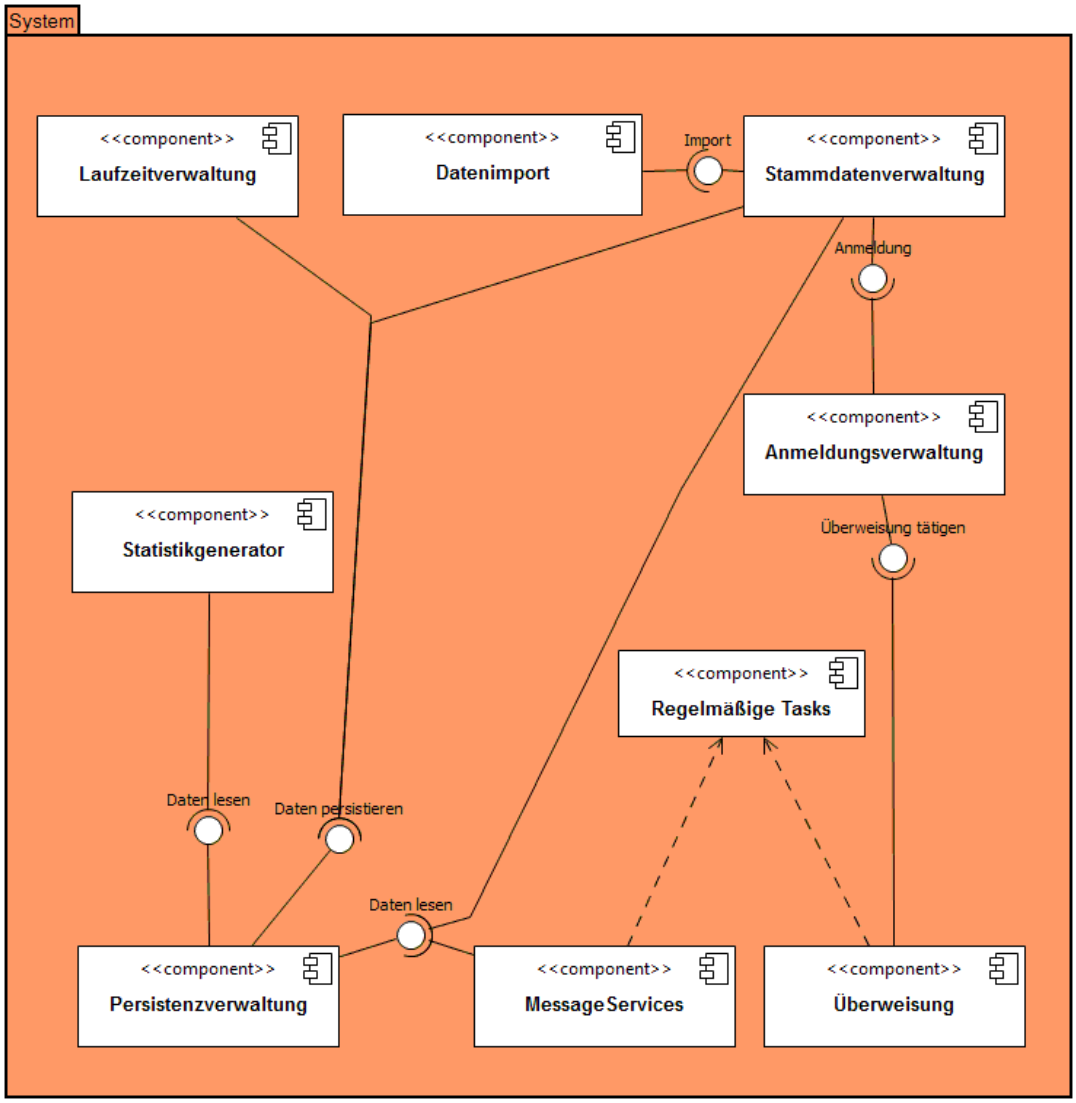
Von der Bankanwendung kann RunningEasy sich Bestätigung der Gebührüberweisung melden lassen. Somit behält das System den Überblick über bezahlte und unbezahlte Gebühren.

Das Laufdatenerfassungssystem enthält Laufdaten von Teilnehmern einer Veranstaltung. Diese benutzt RunningEasy um danach die Ergebnisse der Veranstaltung und Teilnehmer zu erstellen.

4. Bausteinsicht

4.1 Komponentendiagramm

Abb. 2: Komponentendiagramm



Im obigen Diagramm sind die wichtigsten Komponenten mitsamt ihren Schnittstellen und Verbindungen angezeigt.

Besondere Erklärung bedarf die Komponente „Regelmäßige Tasks“, welche hier als abstrakte „Oberkomponente“ von „MessageServices“ und „Überweisung“ dargestellt ist. Nach unserem Lösungsansatz war dies notwendig, da diese Teile der Anwendung nicht direkt durch Benutzereingabe sondern in irgendeiner Form zeitgesteuert getriggert werden müssen.

Beispielsweise muss die Komponente MessageService automatisch nach Ablauf bestimmter Zeiten Mitteilungen an außenstehende Systeme verschicken, ohne dass es einen Input eines menschlichen Akteurs bedarf.
Dies unterscheidet diese Teile der Anwendung vom Restsystem und ist daher wie dargestellt modelliert.

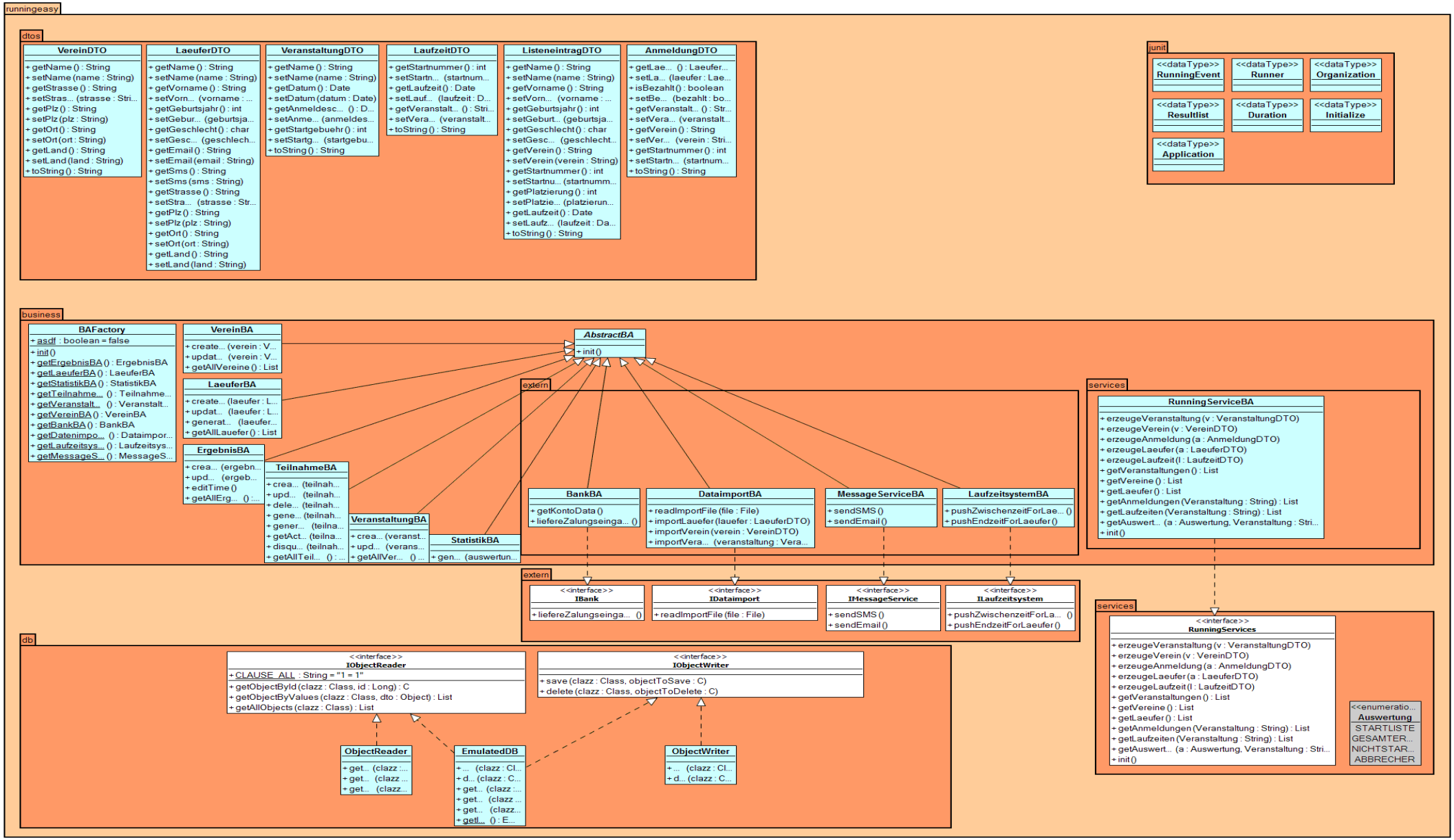
Die Stammdatenverwaltung kümmert sich um jegliche Änderung des Datenbestandes welche durch Eingaben im Client durchgeführt werden.

Die Datenimport Komponente liest die Altdaten ein und legt über eine Schnittstelle mit der Stammdatenverwaltung diese importierten Daten auch im aktuellen System wieder an.

Alle Komponenten mit schreibenden oder lesenden Zugriff auch persistente Daten kommunizieren über vorgegebene Schnittstellen mit einer Persistenzverwaltungs-Komponente welche die Implementierung der Persistenzschicht kapselt.

4.2 Klassendiagramm

Abb. 3: Klassendiagramm



In Abb.3 sind fast alle Klassen abgebildet, die im RunningEasy implementierten werden. Lediglich wenige nicht relevanten (Hilfs-)Klassen, wie Testklassen sind nicht dargestellt.

Die Programmlogik ist wie gefolgt aufgebaut:

Die Data Transfer Objects (**DTO**) sind Datenstrukturen, welche die Grundinformationen beinhalten. Diese werden von den Business Activities (**BA**) verwaltet und später auch persistiert.
Die DTOs haben nur einfache Getter und Setter-Methoden für ihre Attribute und beinhalten sonst keine weitere Logik.

Die **BA-Klassen** werden von der abstrakten Klasse AbstractBA abgeleitet. Jede BA wurde für die einzelnen DTOs definiert, welche für die Verarbeitung dieses DTOs zuständig ist. Die vererbte *init()*-Methode veranlasst das instantiierte BA-Objekt weitere Konfigurationen vorzunehmen, bevor es verwendet werden kann. Diese Methode sollte von jeder Subklasse überschreiben werden.
Die Oberklasse AbstractBA ermöglicht es durch ihre interne Implementierung auch den Zugriff auf die Datenhaltung, diese wird zu den einzelnen BA-Klassen vererbt und ermöglicht auch diesen Zugriff auch die Persistenzschicht.

Die Interfaces **IObjectreader** und **IObjectwriter** sind unsere selbstdefinierte Schnittstellen zur Persistieren der Daten (nur DTOs). Sie definieren einige wenige Methoden zum lesen und schreiben/speichern von DTOs. Sie müssen im Verlaufe der Entwicklung auch von der „echten“ Datenbankschicht implementiert werden.
Damit bieten diese Klassen eine funktionierende Abstraktion der Persistenzschicht welche derzeit mithilfe der Klasse *EmulatedDB* implementiert wird.

Über die **BAFactory** werden die einzelnen BAs generiert und können abgerufen werden. Hier kommt das Factory- sowie das Singleton-Entwurfsmuster zum Einsatz.

Die **RunningServicesBA** ist die wichtigste BusinessActivity, welche die vorgegebene Schnittstelle RunningServices implementiert.
Dabei delegieren die vorgegeben Methoden weiter an die jeweils zuständige BA-Klasse. Damit stellt RunningServicesBA „nur“ das Entwurfsmuster einer Fassade dar.

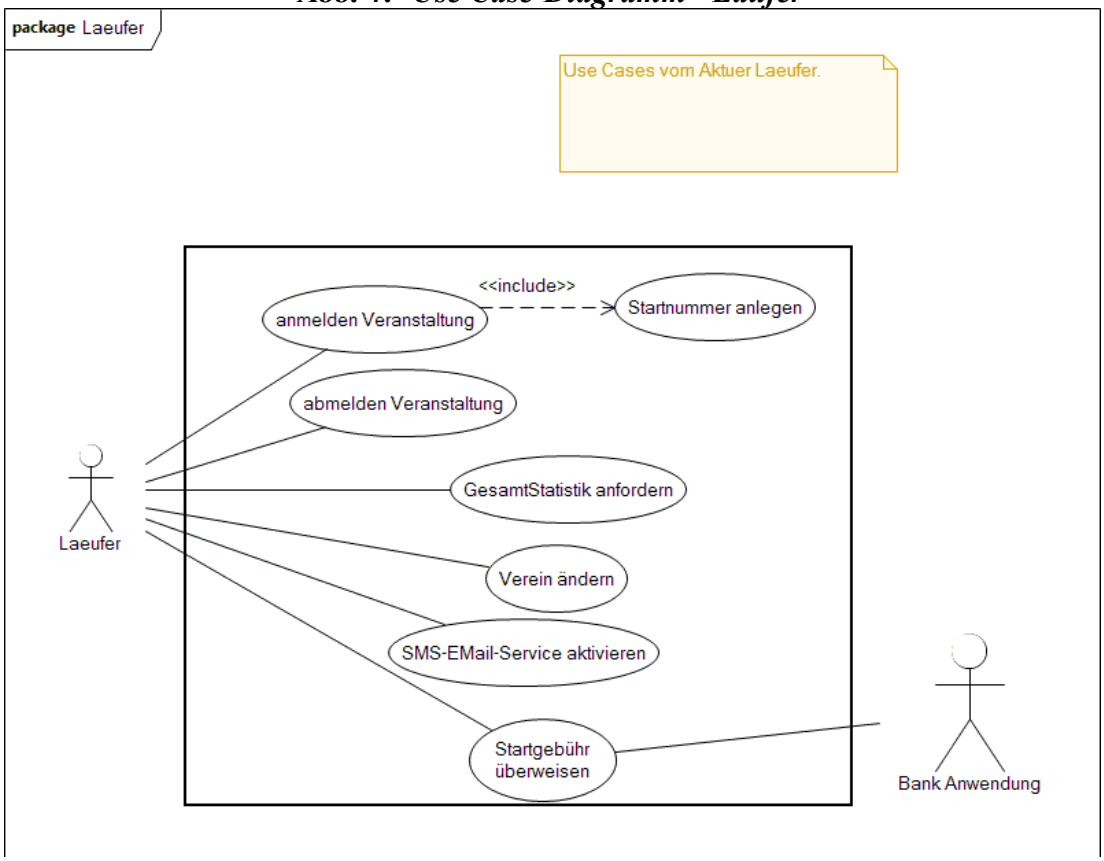
Das **„junit“-Package** beinhaltet die einzelnen JUnit-Tests für die entsprechenden Komponenten der Anwendung.

5. Laufzeitsicht

5.1 Use Case Diagramm

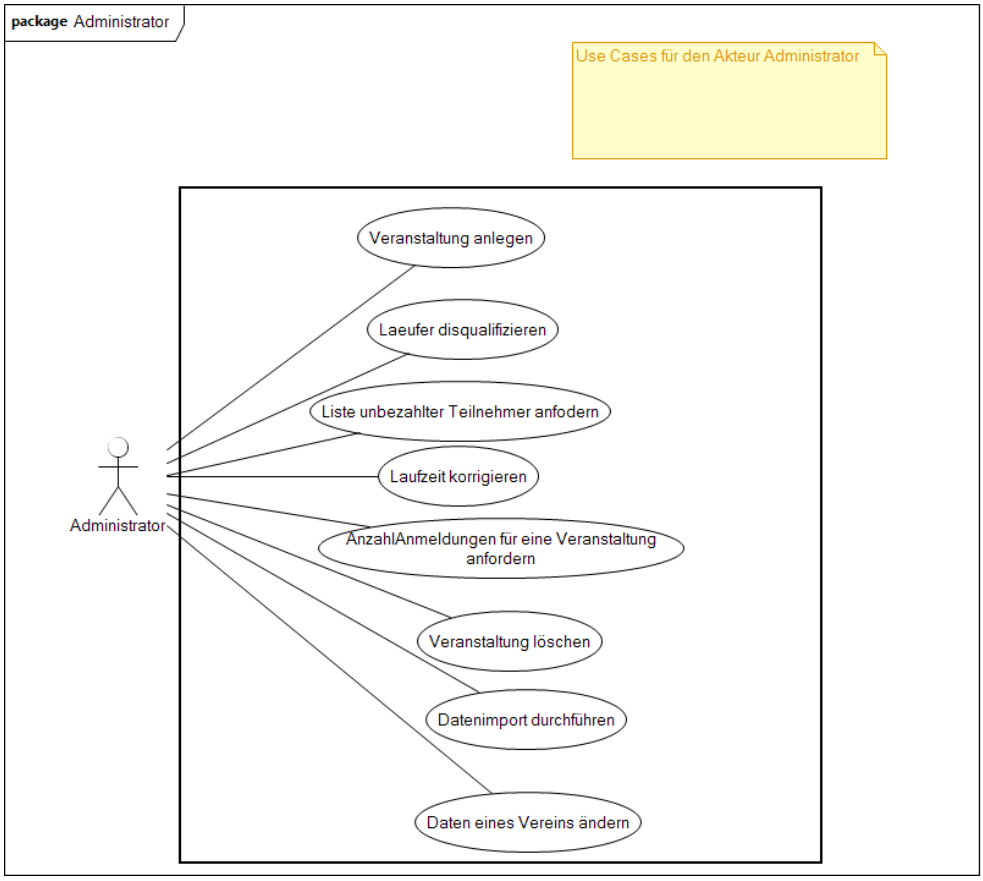
Für eine bessere Übersicht wurden das Use Case-Diagramm in drei separate Diagramme gespalten, welche jeweils um einen zentralen Akteuer angeordnet sind. Dies sind einmal der Läufer bzw. Teilnehmer, ein Administrator/Systemverwalter sowie die externen Schnittstellen.

Abb. 4: Use Case-Diagramm - Läufer



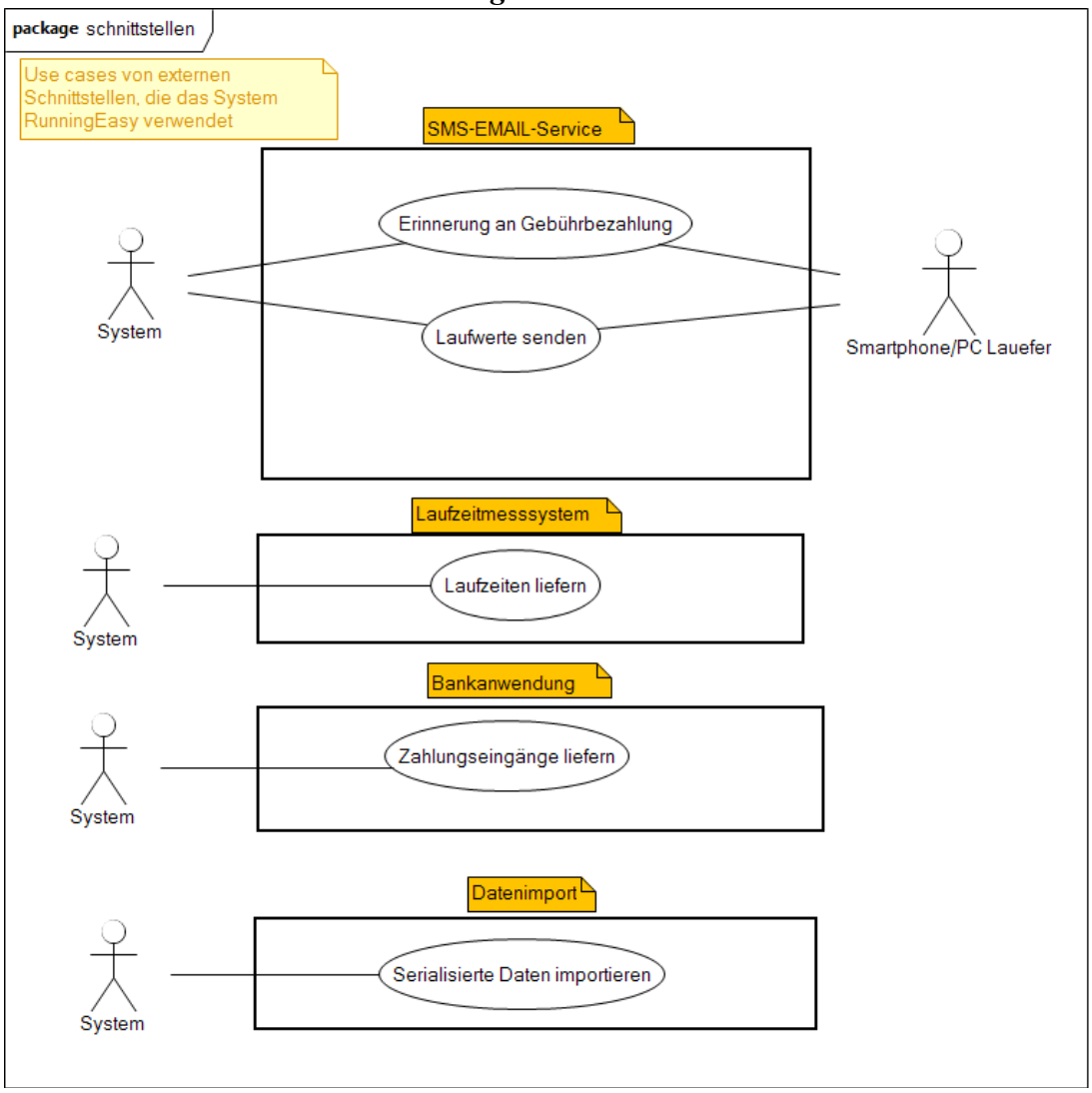
Im obigen Diagramm sind die alle bedeutenden Use Cases des Akteurs "Läufer" dargestellt.

Abb. 5: Use Case-Diagramm - Administrator



Im obigen Diagramm sind die alle bedeutenden Use Cases des Akteurs "Administrator" dargestellt.

Abb. 6: Use Case-Diagramm – externe Schnittstellen



Im obigen Diagramm sind die alle bedeutenden Use Cases der externen Schnittstellen dargestellt.
Hierbei ist jeweils das System "Runningeasy" der Akteur welches mit externen Programmen/Komponenten interagiert.

5.2 Sequenzdiagramm

Um eine grobe Idee von der Abarbeitungskette des Systems von den einzelnen Use Cases zu geben sind hier zwei Sequenzdiagramme modelliert.

Abb. 7: Sequenzdiagramm - Datenimport

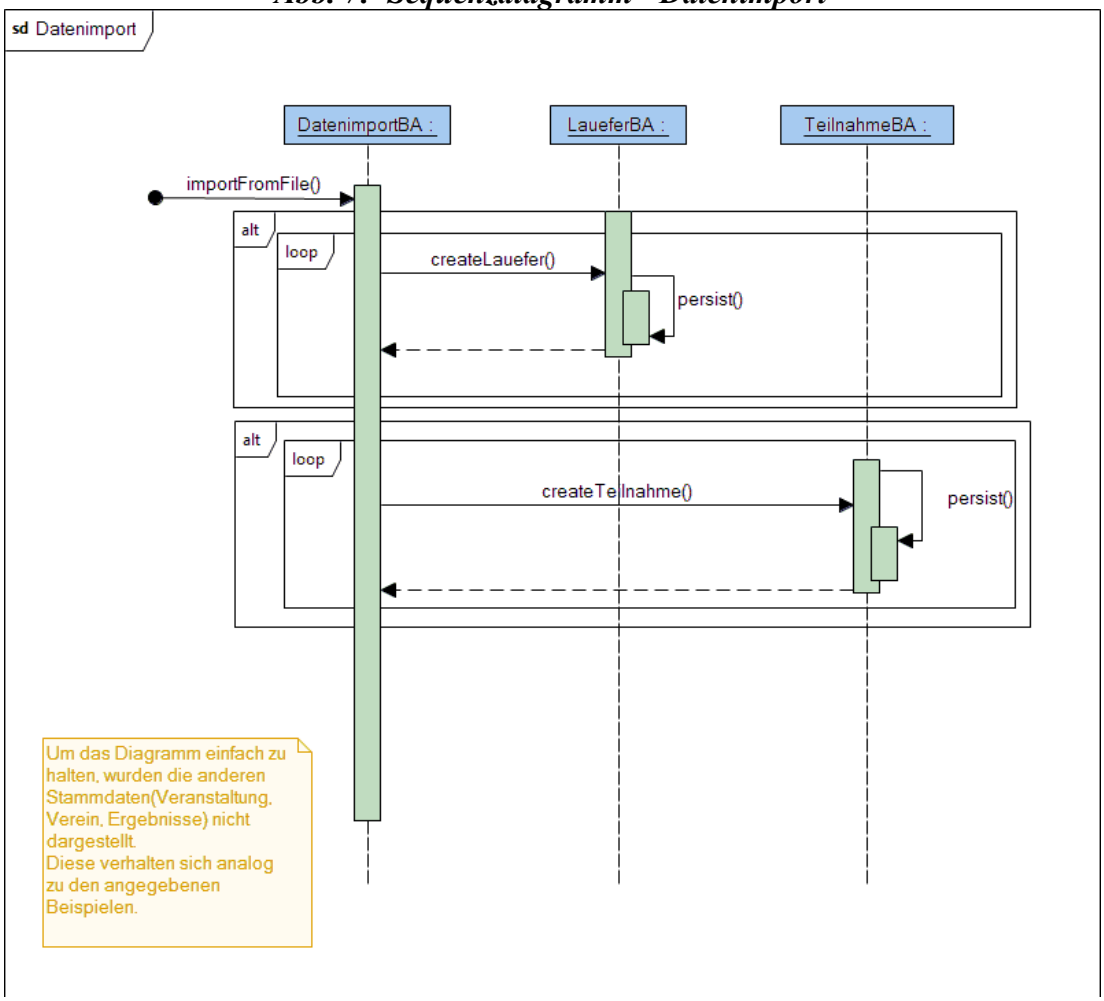
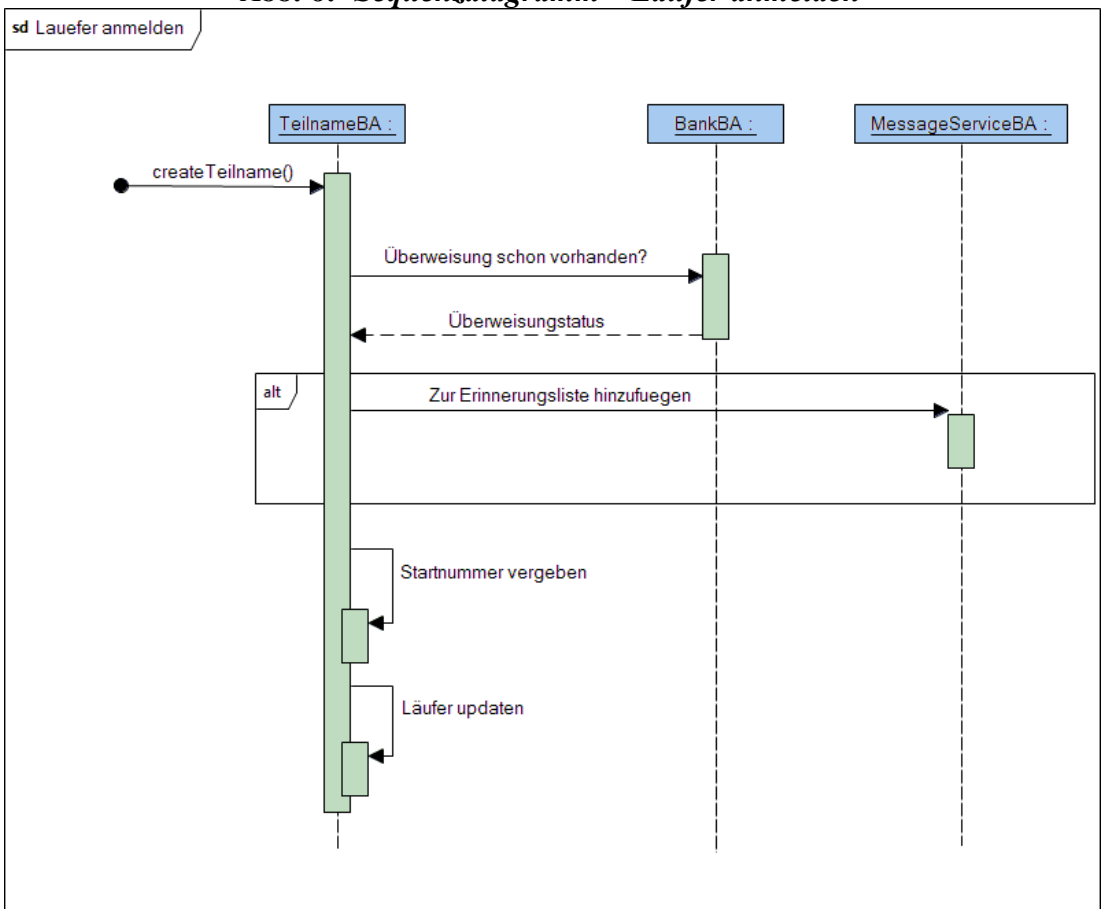


Abb. 8: Sequenzdiagramm – Läufer anmelden



6. Verteilungssicht

Keine nähere Beschreibung.

7. Entwurfsentscheidungen

7.1 Architekturaspekte

- klare Trennung von Logik und Daten (BA-Klassen - Business Logik, BE-Klassen Datenobjekte)
- einfaches, einheitliches und verständliches Design der einzelnen Schichten des Systems
- klare Hierarchien und Abarbeitungsverläufe bei Vermeidung von Zyklen in der Aufrufhierarchie

7.2 Implementierungsaspekte

- guter, eleganter und an "Java Code Conventions" (http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html) angepasster Sourcecode
- oœbersichtlicher und verständlicher Code (gut dokumentiert, aber auch nur dort wo es notwendig ist)
- "Codesprache" Englisch, Fachbegriffe werden jedoch nichtübersetzt
- gute Testbarkeit, Testfunktionen zu den implementierten Füllen

7.3 Managementsicht

- alle relevanten Aufgaben werden mithilfe des Issue-System von Github dokumentiert (mit festem Prozessablauf)
- 4-Augenprinzip bei jeder Issue: Review der Arbeit des Partners
- Selbständiges Arbeiten aber durch regelmäßiges Besprechungen synchronisiert

8. Technische Konzepte

8.1 Programmiersprache

- Es wird ausschließlich in Java programmiert.

8.2 Laufzeitumgebung

- pure Java (kein JavaEE mit EJB etc.) in Java virtual machine JVM
- JPA 2.0 mit EclipseLink als Implementierung

8.3 Testbarkeit

- JUnit-TestCases werden verwendet.

8.4 Tools

- Eclipse Java EE zur Java-Programmierung
- TopCased für UML-Diagramme
- OpenOffice/Libreoffice als Officeprogramm
- Git als Versionsverwaltungstool
- Github als Issuemanagement-Tool/Bugtracker

9. Risiken

Keine nähere Beschreibung.

10. Glossar

Tab. 3: Glossar und Begriffserklärung

Stammdaten	Grundinformationen, meist statische Daten welche nur selten geändert werden, meistens keinen Zeitbezug
BA	Business Activity, „stateless“, Klasse mit Business Logik
BE	Business Entity, keinerlei Logik, persistierbar, serialisierbar Attribute
DTO	Data Transfer Object, Werteklasse enthält keine Logik