Erstellung eines Trainingssimulators für Radargeräte durch Restrukturierung, Modularisierung und Erweiterung einer OSGi-Anwendung

Marco Schäfer

30.1.2020

Inhaltsverzeichnis

1 Einleitung

1.1 Themenumfeld

Softwaresysteme sind heutzutage im ständigen Wandel und werden insbesondere, seitdem es die Möglichkeit gibt Updates und Patches über das Internet bereitzustellen, immer langlebiger. Wenn Quellcode eine lange Lebensdauer haben soll, ist es unabdingbar, dass dieser eine gute Wartbarkeit hat. Ansonsten können der Zeitaufwand und die Entwicklungskosten bis ins Unermessliche steigen. Diese Kosten entstehen, wenn Fehlerbehebungen am Code durchgeführt werden oder dieser um funktionelle Features erweitert wird. Ein wichtiges Konzept, um die Wartbarkeit von Softwaresystemsen zu verbessern ist eine Modularisierung der Softwarekomponenten.

Im Rahmen der Bachelorarbeit wird eine Anwendung mit einer modularen Architektur entworfen und implementiert. Um dies umzusetzen wird das Framework Equinox verwendet, welches die OSGi-Kernspezifikationen implementiert. „OSGi bietet ein Konzept zur Modularisierung und Komponentisierung von Java-Applikationen, und zwar nicht nur zur Entwicklungszeit, sondern auch zur Laufzeit“. Diese Applikation entsteht bei der Firma Thales Deutschland GmbH am Standort in Ditzingen in der Abteilung LAS (Land and Air Systems). In der Abteilung werden Radarsysteme für den zivilen, wie den militärischen Einsatz entwickelt. Die Abteilung ist in das Sensorteam und das Venusteam aufgeteilt. Das Venusteam ist für die Steuerungssoftware der Radarsensoren verantwortlich. Die Steuerungssoftware heißt Venus und bietet ein MMI (Man Machine Interface) zur Steuerung von verschiedenen Radargeräten. Zudem visualisiert die Venus Sensordetektionen auf einer Karte und liefert weitere relevante Sensorinformationen, wie z.B. Hardwarefehlermeldungen in Tabellen. Die Software läuft auf einem Laptop und ist über einen speziellen Anschluss mit dem Sensor verbunden. Zusätzlich entwickeln sie weitere Eclipse Applikationen zum Warten und Testen der Radarsensoren. Das Sensorteam entwickelt die Hardware und Software der verschiedenen Sensormodelle.

1.2 Aufgabenstellung

Das Sensorteam will einen Trainingssimulator erstellen, um neuen Benutzern die Funktionalitäten der Venus näher zu bringen. Bei der Trainingssimulation nehmen ein Trainer und ein bis mehrere Trainees teil. Jeder der beteiligten Personen verfügt über ein Laptop mit der entsprechenden Software. Während der Trainingssimulation, gibt der Trainer Simulationsdaten in die Benutzeroberfläche der Trainingssimulator Anwendung ein. Diese Informationen werden über eine Netzwerkschnittstelle an die Computer der Trainees übermittelt und dort in Echtzeit simuliert. Die Aufgabe des

Trainingssimulators ist es, dass die Trainees lernen, die simulierten Daten mit der Steuerungssoftware korrekt zu erfassen und richtig interpretieren. Ein detaillierterer Ablauf wird in Kapitel X.X beschrieben.

Das Ziel dieser Bachelorarbeit ist es, die beschriebene Trainingssimulator Anwendung zu implementieren. Um Wiederverwendbarkeit der Komponenten und eine einfache Erweiterung um funktionelle Features zu gewährleisten, steht eine modulare Architektur der Software im Vordergrund. Die Grundlage für diese Anwendung ist ein Radarsimulator, welcher einen realen Radarsensor simuliert. Dieser Radarsimulator wurde von früheren Praktikanten und Werkstudenten programmiert und wird derzeitig für interne Tests der Venus verwendet. Jedoch tendiert die Softwarearchitektur des Radarsimulators zu einer monolithischen Architektur, da einzelne Softwaremodule mehrere verschiedene Funktionen haben und wenig klare Schnittstellen zwischen den bereits vorhandenen Modulen definiert sind. Damit darauf sinnvoll aufgebaut werden kann, muss diese Architektur im Vorhinein überarbeitet werden. Zudem soll die Anwendung so angepasst werden, dass der Großteil der Softwarekomponenten in der Testanwendung und in der Trainingssimulator Anwendung verwendet werden können. Dazu muss eine entsprechende Architektur erstellt werden, die durch möglichst wenige Änderungen zu den verschiedenen Ausprägungen abgeleitet werden kann.

Eine weitere Aufgabe ist es, die Trainingsanwendung, sowie die Testanwendung und um drei Features ergänzt werden. Diese Features sind Single Target Tracking, die Bereitstellung des Dopplertons während STT und die Einbindung der Detektionswahrscheinlichkeit von Radarobjekten. Single Target Tracking ist die Funktion ein Radarziel über einen bestimmten Zeitraum zu verfolgen. Dabei wählt man ein bereits existierendes Ziel in der Venus aus und startet den STT Modus. Daraufhin schwenkt der Sensor mit einem kleinen Winkel über das Ziel und richtet sich bei Bewegung des Zieles neu aus. Diese Funktion wird vom Sensor bereitgestellt und nicht von der Venus. Deshalb muss der Simulator diese Funktion bereitstellen. Wenn das Ziel von STT erfasst wird soll nun der Dopplerton des verfolgten Objektes per Audio abgespielt werden. *Detektionswahrscheinlichkeit* Diese Features sollen eine realistischere Trainingsumgebung für neue Nutzer des Trainingssimulators bieten.

1.3 Aufbau der Thesis

Zu Beginn der Thesis werden die allgemeinen Grundlagen der Konzepte und Technologien, die in dieser Arbeit verwendet werden, erläutert. Als nächstes wird die alte Architektur des Radarsimulator analysiert und auf Schwachstellen überprüft. Zudem werden die funktionellen Anforderrungen der Trainingssimulator- und Testsimulatorandwendung aufgelistet und es wird untersucht an welchen Komponenten

die vorgegebenen Erweiterungen anknüpfen müssen. Nach der Analyse wird das Konzept der neuen Architektur vorgestellt. Ein besonderes Augenmerk wird dabei auf die Softwaremodule und deren Schnittstellen gelegt. Des Weiteren werden den verschiedenen Anwendungsausprägungen die Softwaremodule zugeordnet und genauer erklärt. Außerdem wird gezeigt, wie das Datenmodell verändert wurde und von den neuen Softwarekomponenten verwendet wird. Ergänzend wird darauf eingegangen, wie die Buildpipline die Testanwendung, die Traineranwendung und die Traineeanwendung baut. Im folgenden Kapitel wird erklärt, wie die Softwarekomponenten implementiert und zusammengesetzt wurden. Danach wird bewertet, wie der Status der Applikation am Ende des Projektes aussieht, bzw. ob alle wichtigen Funktionen enthalten sind und die Anwendung stabil läuft. Zuletzt gibt es noch eine Schlussbetrachtung der Thesis, in der Erkenntnisse der Arbeit beschrieben werden und wie es in der Zukunft mit den Technologien und der Anwendung weitergehen soll.

2 Grundlagen

3 Analyse

3.1 Softwarearchitektur des Radarsimulators

Im folgenden Abschnitt werden die einzelnen Softwarekomponenten der Radarsimulator Anwendung beschrieben und auf Wiederverwendbarkeit und Modularität überprüft. Dabei wird sich auf die Kernkomponenten der Anwendung beschränkt.

Der Radar Simulator ist eine Eclipse RCP Anwendung. Das Modul, das die Benutzeroberfläche zur Verfügung stellt, ist der RadarSimulator.app. Darin werden die Startinformationen und weitere Fenstereinstellungen für diese Anwendungen angegeben. Der Startpunkt der Anwendung ist die SimulatorMain Klasse. Diese startet das Backend und Frontend. Zudem besitzt der Radarsimulator einen Restart Service. Dieser Service ermöglicht es, die Simulation über ein Fenster Menü item neu zu starten. Dabei wird die Benutzeroberfläche und die Simulator Instanz neu geladen. Diese Funktion ist in der SimulatorMain Klasse implementiert.

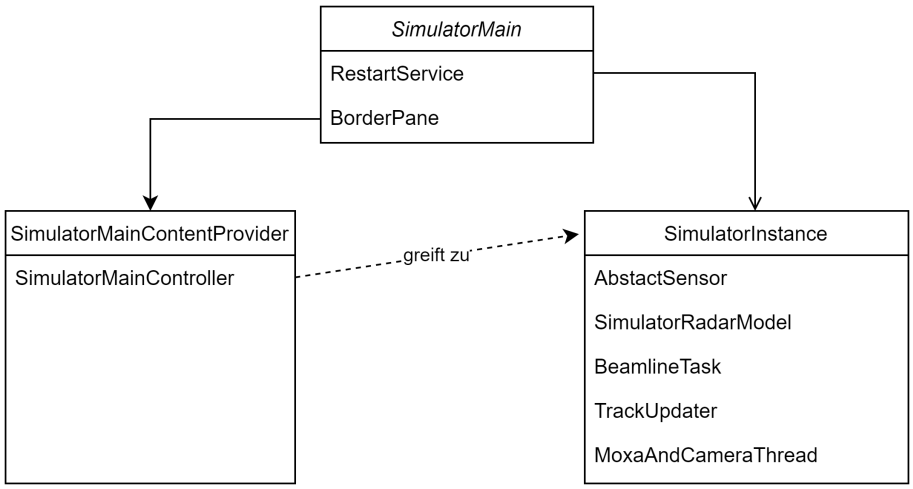

Abbildung 3.1: UML-Diagramm des Radarsimulators

4 Konzept

4.1 Neue Architektur

4.2 Aufbau der Ausprägungen

4.3 Features

4.3.1 Single Target Tracking

Um Single Target Tracking zu ermöglichen, benötigt die Anwendung eine Erweiterung um zwei Funktionen. Zum einen muss überprüft werden, ob sich ein Ziel im Suchbereich befindet, wenn dies zutrifft, soll nach jeder Bewegung des Ziels, das Audiogate auf dessen Position gesetzt werden. Des Weiteren muss sich die Beamline, je nach Zielverfolgung oder Zielsuche korrekt verhalten.

Während der STT Modus aktiviert ist, bewegt sich die Beamline innerhalb eines engen Sektors wiederholt über das Ziel. Wenn kein Ziel gefunden wird, sucht die Beamline in einem größeren Bereich nach einem neuen Ziel. Der Sektor hat das Audiogate als Mittelpunkt und ändert seinen Bereich synchron zum Audiogate. Wenn ein Ziel verfolgt wird beträgt die Breite des Sektors 100 mils. Falls der Sensor ein Ziel zur Verfolgung sucht, beträgt die Bewegungsweite 300 mils. Diese Funktion wird im BeamlineTask implementiert, da dieser Zugriff auf das Audiogate und die Beamlineinformationen hat. Der BeamlineTask hat jedoch keinen Zugriff auf die Daten der Tracks. Deshalb weiß er nicht, ob ein Ziel verfolgt wird und kann kein Audiogate setzen. Damit der Beamlinetask weiß, ob ein Ziel verfolgt wird, wird ein Booleanflag, welches jederzeit abgefragt werden kann, im Model hinzugefügt.

Das Flag, sowie das Audiogate sollen vom Trackupdater gestetzt werden. Dieser hat alle Informationen der verfügbaren Tracks, die in der Simulation existieren und kann berechnen, ob das Ziel derzeitig von der Beamline erfasst wird. Der TrackUpdater hat die Methode schedule(), welche wiederholt nach einem bestimmten Zeitintervall aufgerufen wird. Diese Methode aktualisiert die Tracks und sendet Track Informationen an den Sensor. Diese Methode wird nun erweitert, um das STTFlag und das Audiogate zu setzen. Ein verfolgter Track wird im TrackUpdater gespeichert.

Zum Anfang der Methode wird überprüft, ob bereits ein Track verfolgt wird, falls dies nicht der Fall ist wird die Distanz vom Audiogate zu allen vorhandenen Tracks berechnet. Die Werte werden in einer Liste mit dem jeweiligen Track gespeichert, wenn sie innerhalb des STT-Sektors liegen. Die STT-Funktion ist so definiert, dass sie den nächsten Track zum Audiogate innerhalb des Sektors so lange verfolgt, bis dieser verschwindet oder man das Audiogate manuell ändert. Deshalb wird die Liste so sortiert, damit der Track mit dem geringsten Abstand zum Audiogate an erster Stelle steht. Nachdem sortiert wurde, wird das erste Element gespeichert, das Flag wird auf true gesetzt und die Liste wird geleert. Wenn die Liste leer ist wird nichts getan.

Wird bereits ein Track verfolgt, wird geprüft, ob der zu verfolgende Track aktuell existiert. Das passiert indem man in der Liste, der gegenwärtigen Tracks nach einem

Track mit der identischen ID des gespeicherten Tracks ist sucht. Ist dieser vorhanden, wird dieser Track als neuer Track gespeichert. Als nächstes wird getestet, ob dieser Track vom Sensor detektiert wird. Trifft es zu, dass der Track vorhanden ist und detektiert wird setzt der TrackUpdater das Audiogate auf die aktuelle Position des Tracks. Wenn der Track nicht detektiert wird oder existiert, wird ein Zähler hochgezählt. Es kann passieren, dass ein Track in einem oder mehreren Durchläufen nicht im Bereich der Beamline liegt oder dass der Track vom Sensor nicht erfasst wird, da er z.B. zu weit entfernt ist. Durch den Zähler kann man einen Toleranzwert festlegen, wie oft der Track nicht erkannt werden muss bevor diesem nicht mehr gefolgt wird. Dem Track wird entfolgt, indem das Flag auf false gesetzt und der Track gelöscht wird.

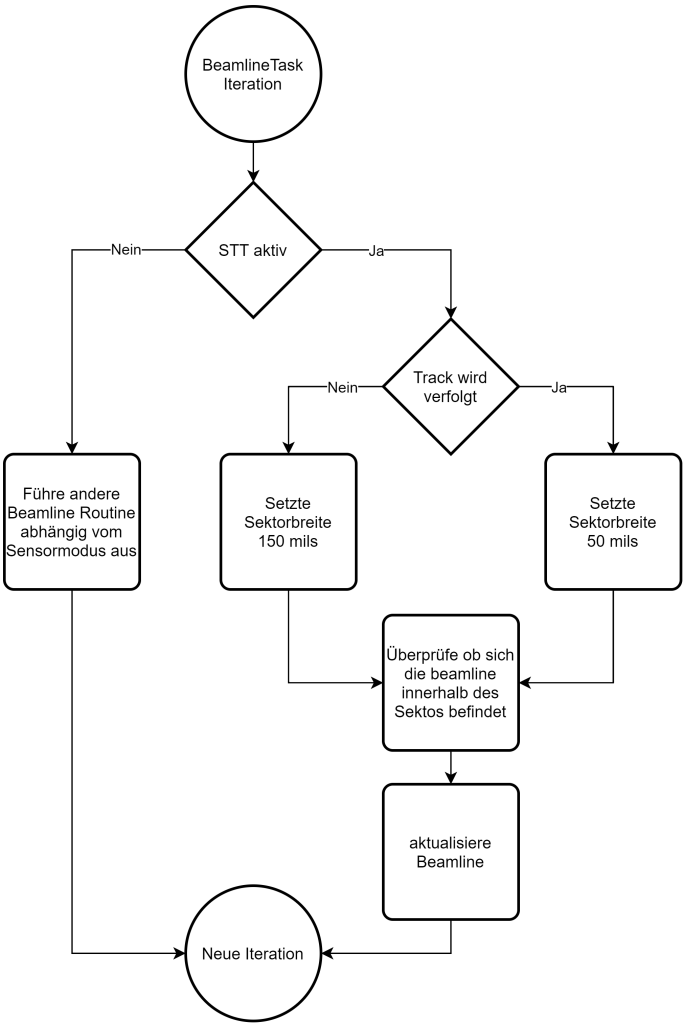

Abbildung 4.1: Flussdiagramm des Algorithmus des BeamlineTask

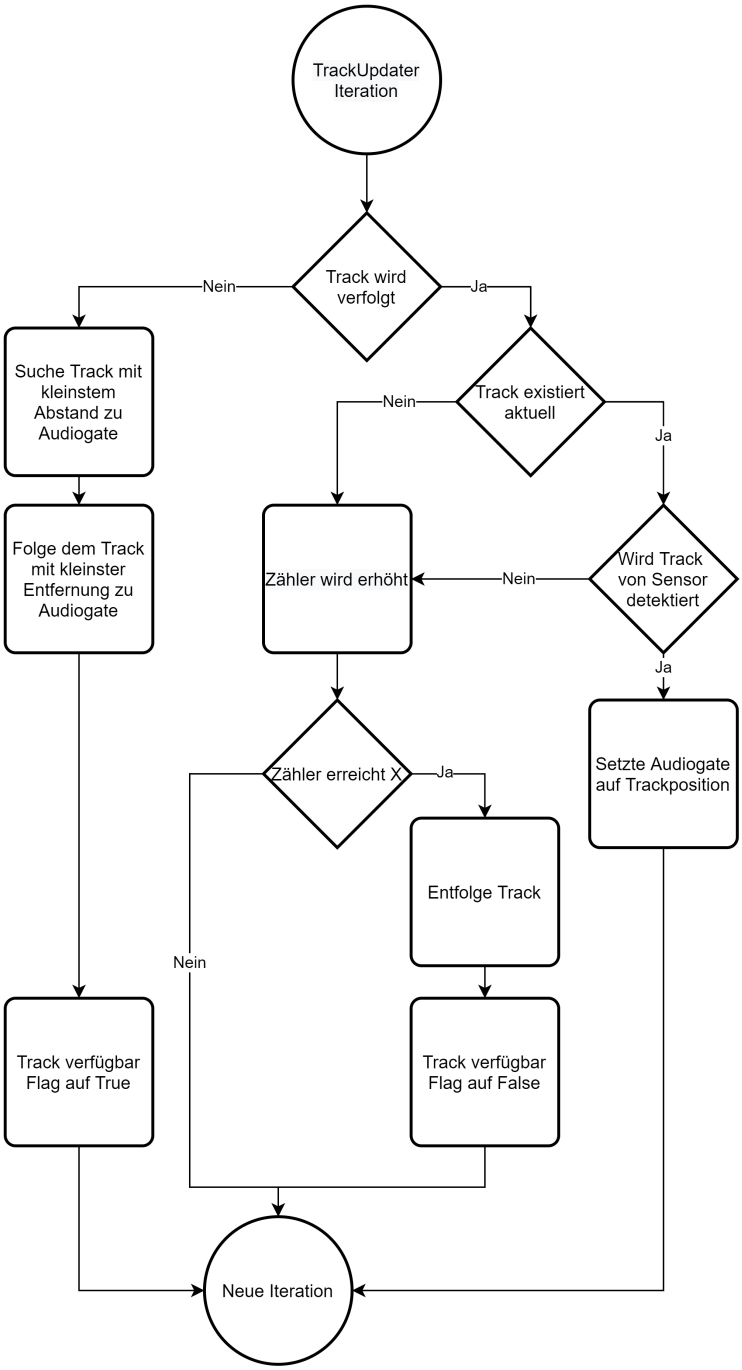

Abbildung 4.2: Flussdiagramm des Algorithmus des Trackupdaters