Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

Spezifikation und Implementierung einer Platooning Funktion auf Basis der CeCar-Plattform

Masterarbeit

im Studiengang
Computer Engineering

am Fachbereich
Ingenieurwissenschaften – Energie und Information

an der
Hochschule für Technik und Wirtschaft Berlin

vorgelegt von
Philipp Fritz Jaß

Berlin, 24.11.2020

Betreuer:
Prof. Dr.-Ing. Carsten Thomas
Prof. Dr.-Ing. habil. Carsten Gremzow

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass

- ich die vorliegende wissenschaftliche Arbeit selbständig und ohne unerlaubte Hilfe angefertigt habe,
- ich andere als die angegebenen Quellen und Hilfsmittel nicht benutzt habe,
- ich die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe,
- die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfbehörde vorgelegen hat.

Berlin, 24.11.2020

Kurzfassung

Diese Masterarbeit beschäftigt sich mit dem Entwurf und der Implementierung eines Platooning-Systems. Dieses System soll auf der Grundlage der CeCar-Plattform entwickelt werden. Das CeCar ist ein Modellauto, welches als Versuchsträger für verschiedenste Funktionen des autonomen Fahrens entwickelt wurde. Ziel ist es mit mehreren dieser Fahrzeuge ein Platoon bilden und im Verbund fahren zu können.

Um der Charakteristik des Platoonings als System-of-Systems gerecht zu werden, wurde eine Multiagenten-Architektur als Basis für das zu entwickelnde System festgelegt. Dabei wird jedes CeCar durch einen BDI-Agenten repräsentiert. Ein solcher Agent verfügt über eine eigene Wissensbasis, in der Informationen zum internen Systemzustand und seiner Umwelt gespeichert sind. Diese wird kontinuierlich aktualisiert und als Grundlage für die Handlungsplanung verwendet. Abhängig von den Zielen, die der Nutzer oder das System selbst bestimmt hat, werden die Handlungen in Form von Strategien geplant. Dabei werden die verfügbaren Strategien auf ihre Ausführbarkeit in der aktuellen Situation hin bewertet und die am besten geeignete ausgewählt. Im implementierten Platooning-System bestehen solche Strategien aus einem oder mehreren Services, die gemeinsam die gewünschte Funktion erfüllen. Die Services können dabei sowohl vom lokalen System als auch von anderen Systemen des Platoons bereitgestellt werden. Im Rahmen dieser Arbeit wurden Strategien für grundlegende Koordinierungsfunktionen eines Platoons implementiert. Dazu gehören Strategien für die Bildung des Platoons, sowie für das Beitreten oder Verlassen eines Platoons. Darüber hinaus wurden zwei einfache Strategien für das kollaborative Fahren realisiert.

Das Platooning-System wurde auf der Basis des ROS2-Frameworks realisiert. Die Software ist in einem ROS2-Package zusammengefasst und kann so auf den verschiedenen Fahrzeugen der CeCar-Plattform eingesetzt werden. Die Funktionen des implementierten Platooning-Systems konnten abschließend in einer Simulationsumgebung anhand einer Beispielfahrt erfolgreich validiert werden.

Abstract

This master's thesis deals with design and implementation of a platooning system. This system will be developed based on the CeCar platform. The CeCar is a model car that has been developed as a test vehicle for various functions of autonomous driving. The aim is to form a platoon with several of these vehicles and to be able to drive collaboratively.

To meet the characteristics of platooning as a system-of-systems, a multi-agent architecture was established as the basis for the system to be developed. Each CeCar is represented by a BDI agent. Such an agent has its own knowledge base that stores information about the internal state of the system and its environment. This knowledge is continuously updated and used as a basis for action planning. Depending on the goals determined by the user or the system itself, the actions are planned in the form of strategies. The available strategies are assessed for their feasibility in the current situation and the most appropriate one is selected. In the implemented platooning system, such strategies consist of one or more services that jointly fulfill the desired function. The services can be provided by both the local system and other systems of the platoon. As part of this thesis, strategies for basic coordination functions of a platoon have been implemented. This includes strategies for the formation of the platoon, as well as for joining or leaving a platoon. In addition, two simple strategies for collaborative driving were implemented.

The platooning system was implemented based on the ROS2 framework. The software is grouped in a ROS2 package and can be used on the various vehicles of the CeCar platform. Finally, the functions of the implemented platooning system were successfully validated in a simulation environment.

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AHS	Automated Highway System
BDD	Block Definition Diagram
BDI	Belief-Desire-Intention
BLDC	Brushless Directed Current
CAS	Complex Adaptive Systems
FoS	Federation-of-Systems
IBD	Internal Block Diagram
MCU	Master Control Unit
OSRF	Open Source Robotics Foundation
RC	Remote Controlled
RCU	Realtime Control Unit
SN	Stakeholder Needs
SOI	System of Interest
SoS	System-of-Systems
SysML	Systems Modeling Language
TC	Test Case
UC	Use Case
UML	Unified Modeling Language
V2I	Vehicle to Infrastructure
V2V	Vehicle to Vehicle

1 Einleitung

Der Straßenverkehr spielt in der heutigen Gesellschaft seit vielen Jahrzenten unverändert eine bedeutende Rolle. Ein Großteil des Personen- aber auch des Güterverkehrs findet auf den Straßen statt. Im Jahr 2018 hatte das Verkehrsaufkommen auf deutschen Straßen etwa das Dreifache des Wertes von 1970 erreicht. Seit 1990 sind die gefahrenen Kilometer um mehr als die Hälfte gestiegen. Einen vorläufigen Höchstwert erreichte das Verkehrsaufkommen in Deutschland im Jahr 2016 mit knapp 770 Milliarden Kilometern, die auf der Straße zurückgelegt wurden [1]. Der etwas langsamere Anstieg in den letzten Jahren ist sicherlich teilweise dadurch zu begründen, dass die Straßeninfrastruktur an ihre Kapazitätsgrenzen gelangt ist. Diese starke Auslastung der Straßen führt gleichzeitig zu einem erhöhten Risiko von Verkehrsunfällen. Im Jahr 2019 wurden mehr als 2,6 Millionen Unfälle auf deutschen Straßen aufgenommen. Davon führten ca. 300.000 Unfälle zu Personenschäden [2]. Die Mehrheit dieser Unfälle ist dabei auf menschliche Fehler zurückzuführen. Außerdem stellt der Straßenverkehr einen erheblichen Anteil der CO2-Emissionen dar. Weltweit lag dieser Anteil im Jahr 2016 bei über 18% der Emissionen aus der Verbrennung fossiler Brennstoffe [3] und spielt damit eine bedeutende Rolle in Bezug auf den Klimawandel. Zur Lösung all dieser Probleme wird verstärkt auf das automatisierte Fahren gesetzt. In diesem Bereich gibt es viele verschiedene Ansätze und technische Lösungen in Form von einfachen Assistenzsystemen für den Fahrer bis hin zu vollständig autonomen Fahrzeugen.

Das sogenannte Platooning ist ein Konzept im Feld des automatisierten Fahrens. Als Platoon wird in der Regel eine Gruppe von gemeinsam agierenden Fahrzeugen bezeichnet, die in einer zusammenhängenden Formation fahren [4]. Es gibt einige große Forschungsprojekte, die sich mit der Umsetzung und dem Potenzial von solchen Platooning-Systemen beschäftigen. Dazu gehören unter anderem PATH, SARTRE oder Energy ITS [5]. In diesen Projekten konnte gezeigt werden, dass unter gleichen Fahrbedingungen das automatisierte Fahren sicherer ist als der achtsamste menschliche Fahrer. Vor allem Systeme mit einer Kommunikation zwischen den einzelnen Fahrzeugen haben sich als besonders vorteilhaft für die Sicherheit erwiesen [6]. Weiterhin ermöglicht die Vernetzung von automatisierten Fahrzeugen eine erhebliche Verbesserung des Verkehrsflusses und die Reduzierung von Staus im Vergleich zum manuell vom Menschen kontrollierten Verkehr, wie er heute auf den Straßen zu finden ist [7]. Das Fahren im Platoon ermöglicht geringe Abständen zwischen den Fahrzeugen bei gleicher Sicherheit [4]. Dadurch fahren, bis auf das führende Fahrzeug, alle anderen Fahrzeuge im Windschatten, was sich positiv auf den Energieverbrauch auswirkt. Vor allem bei LKWs führt die resultierende Reduzierung des Luftwiderstandes zu einem niedrigeren Kraftstoffverbrauch. Auch eine gewisse Verringerung der Emissionen konnte bei den beteiligten Fahrzeugen festgestellt werden [7].

Bei der Realisierung solcher Platooning-Systeme gibt es in der Forschung eine Vielzahl von unterschiedlichen Ansätzen. Diese unterscheiden sich vor allem in den festgelegten Zielen für das Platooning-System, der Kommunikationsstruktur, der notwendigen Infrastruktur, der Zusammensetzung des Platoons und den möglichen Fahrmanövern, die das Platoon ausführen

kann [8]. Viele dieser Ansätze beschäftigen sich hauptsächlich mit der physischen Umsetzung auf echten Straßen oder in realitätsnahen Testumgebungen. Andere Ansätze beschäftigen sich mehr mit dem Platooning-System im Rahmen des Systems-Engineering. Generell kann ein solches System als sogenanntes System-of-Systems betrachtet werden. Es besteht aus mehreren Einzelsystemen, die sowohl in der Lage sind als autonomes System zu agieren, aber auch unter bestimmten Bedingungen mit anderen Systemen zu kooperieren, um so Funktionen untereinander austauschen oder erweitern zu können [9].

In dieser Masterarbeit soll ein solches Platooning-System auf Basis der CeCar-Plattform vorgestellt werden. Das CeCar ist ein Modellauto, welches im Studiengang für Computer Engineering an der HTW Berlin als Testmodell für autonomes Fahren und Fahrassistenzsysteme entwickelt wurde. Bei der Entwicklung des Platooning-Systems werden Methoden des Systems Engineering angewendet, um von Beginn an eine Realisierung als System-of-Systems zu gewährleisten. In dieser Arbeit werden die grundlegend benötigten Funktionen zur Bildung eines Platoons und der Kooperation der Einzelfahrzeuge realisiert, um eine Grundlage zur weiteren Untersuchung von Platooning-Systemen und Systems-of-Systems allgemein mit Hilfe des CeCars zu schaffen. Dafür wird nach einem kurzen Überblick zum aktuellen Stand der Forschung zu Platooning-Systemen und Systems-of-Systems eine Architektur festgelegt, die eine Kommunikation der Systeme definiert und damit die Kooperation der Systeme ermöglichen und fördern soll. Auf dieser Basis werden dann grundlegende Funktionen eines Platoons implementiert. Eine Überprüfung und Auswertung der Funktionen werden anschließend in einer Simulation vorgenommen.

2 Grundlagen

2.1 Definition Platooning

Platooning als Begriff wird in wissenschaftlichen Veröffentlichungen häufig verwendet, wenn es um automatisierte und vernetzte Lösungen für den Straßenverkehr geht. Eine einheitliche Definition dieses Begriffs gibt es allerdings nicht. Oft sind die eingeführten Definitionen recht kurz und nicht sehr ausführlich, was dazu beitragen mag, dass sich bis heute keine einheitliche Verwendung des Begriffes durchgesetzt hat. Zu Beginn dieser Arbeit sollen deshalb einige Definitionen für das Platooning und häufig verwendete Begriffe vorgestellt werden, um ein grundlegendes Verständnis dafür zu schaffen, was Platooning eigentlich ist und welche Aspekte dabei eine Rolle spielen.

Horowitz und Varaiya [10] definieren ein Platoon im Rahmen des PATH Projektes. Im Zusammenhang mit dem PATH Projekt sprechen sie oft von einem Automated Highway System (AHS). Unter diesem Oberbegriff werden alle Technologien zusammengefasst, die einen automatisierten Verkehr auf den Straßen ermöglichen. Dazu gehört beispielsweise auch das Platooning. Sie definieren ein Platoon knapp als Gruppe von bis zu 20 Fahrzeugen, die mit einem geringen Abstand zueinander fahren. In anderen Arbeiten wird meist keine Begrenzung für die Anzahl der beteiligten Fahrzeuge angegeben. Es wird oft der Aspekt der Koordination der Fahrzeuge untereinander oder mit einer zentralen Stelle durch eine stattfindende Kommunikation der Systeme eingebracht [4; 11]. In [5] wird ein Platoon als eine Sammlung von Fahrzeugen beschrieben, die gemeinsam fahren und dabei aktiv in einer Formation koordiniert sind. Eine sehr ausführliche Definition liefert [8]. Dort wird das Platooning auf der Basis von drei grundlegenden Aspekten definiert: dem Grad der Automatisierung, der räumlichen Beziehung und der Sicherheit der Einzelfahrzeuge. Demnach ist Platooning eine Kette von voll- oder teilweise automatisierten Fahrzeugen, die mit geringen Abständen hintereinander herfahren, ohne dabei Sicherheitsbeschränkungen zu verletzten.

Alle diese Definitionen wurden für Platooning-Systeme festgeschrieben, die für den Einsatz im Straßenverkehr entwickelt werden. Da in dieser Arbeit ein Platooning-System mit Modellautos zu Test- und Übungszwecken entwickelt werden soll, welches nicht für den Einsatz zur Personenbeförderung oder im Straßenverkehr vorgesehen ist, wird nachfolgend auf der Basis der Definition von Kalbitz [8] eine für diese Arbeit passende Definition für das Platooning vorgestellt:

Ein Platoon bezeichnet eine Gruppe von voll- oder teilweise automatisierten Fahrzeugen, die mit geringen Abständen hintereinander herfahren und dabei kooperieren, um ein gemeinsames Ziel zu erreichen.

Ein zentraler Bestandteil dieser Definition ist dabei der Zweck des Platoons. Als Grundlage für die Bildung eines solchen Platoons muss es den Fahrzeugen eine Verbesserung mindestens eines Parameters ihrer Fahrt (z.B. Kraftstoffverbrauch, Straßenauslastung, Sicherheit, etc.)

ermöglichen. Um ein Platoon auf dieser Basis zu bilden, ist ein Austausch und eine Koordination zwischen den beteiligten Fahrzeugen notwendig.

In [4] werden zusätzlich zum Platooning einige weitere wichtige Begriffe in diesem Themenbereich erläutert. Danach gibt es für ein Platoon drei verschiede Stufen der Komplexität.
(1) Auf der untersten Stufe findet lediglich eine Steuerung der Geschwindigkeit und des Abstandes der Fahrzeuge statt. Bei dieser sogenannten longitudinalen Steuerung wird das Platoon von einem menschlichen Fahrer im vordersten Fahrzeug gesteuert. Alle folgenden Fahrzeuge halten den vorgegebenen Abstand zum vorherigen Fahrzeug, die Lenkung muss dabei aber in allen Fahrzeugen von einem Menschen gesteuert werden.
(2) Bei Platoons mit semiautonomer Steuerung findet zusätzlich die Steuerung der Lenkung durch Computersysteme statt. Ansonsten werden auch hier die Handlungen des Platoons durch einen menschlichen Fahrer an der Spitze bestimmt.
(3) Erst auf der letzten Stufe, der vollautonomen Steuerung, wird jedes Fahrzeug des Platoons autonom durch einen Computer gesteuert. Es wird folglich in keinem der beteiligten Fahrzeuge ein menschlicher Fahrer benötigt.

Nach dem Zusammenschluss sind neben dem eigentlichen Fahren in einer Formation einige Operationen zur Änderung der Zusammensetzung des Platoons möglich. Diese Operationen werden auch Manöver genannten und im Großteil der wissenschaftlichen Arbeiten zu diesem Thema auf gleiche Weise verwendet:

- *Merge* (engl.: zusammenkommen): Beim Merge-Manöver schließen sich mehrere einzelne Platoons zu einem großen Platoon zusammen. Bei diesem Manöver bleibt also die Zahl der Platoon Mitglieder insgesamt gleich, aber die Anzahl der einzelnen Platoons verringert sich.
- *Form* (engl.: bilden): Das Form-Manöver bezeichnet einen speziellen Fall des Merge-Manövers. In diesem Fall existiert zuvor noch kein Platoon. Mehrere Einzelfahrzeuge bilden dann gemeinsam ein neues Platoon. Gewissermaßen schließen sich also mehrere Platoons mit jeweils genau einem Mitglied zusammen. Streng genommen stellt nach der Platoon Definition ein einzelnes Fahrzeug zwar kein Platoon dar, die Mechanismen für den Zusammenschluss sind aber für Einzelfahrzeuge und ganze Platoons ähnlich. Bei diesem Manöver steigt also sowohl die Anzahl der Platoons als auch der Platoon Mitglieder (da zuvor keine existierten).
- *Join* (engl.: anschließen): Auch das Join-Manöver ist ein Spezialfall des Merging. Dabei existiert bereits ein Platoon und lediglich ein einzelnes Fahrzeug schließt sich diesem an. Bei diesem Manöver erhöht sich lediglich die Anzahl der Platoon Mitglieder. Die Zahl der Platoons bleibt insgesamt gleich.
- *Split* (engl.: aufteilen).: Das Split-Manöver stellt das Gegenteil des Merge-Manövers dar. Ein bestehendes Platoon teilt sich in mindestens zwei neue, kleinere Platoons auf. Die Zahl der Platoons steigt folglich bei diesem Manöver, wobei die Gesamtzahl der Platoon Mitglieder konstant bleibt.
- *Dissolve* (engl.: auflösen): Mit dem Dissolve-Manöver wird die gegenteilige Operation zum Platoon Forming definiert. Es stellt gleichzeitig eine spezielle Form des Split-

Manövers dar. Ein bestehendes Platoon teilt sich dabei nicht in mehrere kleine Platoons auf, sondern löst sich komplett auf, sodass anschließend jedes Mitglied für sich agiert.

- *Leave* (engl.: verlassen): Auch das Leave-Manöver ist eine besondere Form des Splittings. In diesem Fall verlässt lediglich ein Einzelfahrzeug das Platoon, um in der Folge eigenständig zu agieren. Dabei verringert sich nur die Anzahl der Platoon Mitglieder, nicht aber die Zahl der Platoons.

2.2 Definition System-of-Systems

Neben dem Platooning ist auch der Begriff System-of-Systems (SoS) ein zentrales Konzept dieser Arbeit. Er bezeichnet Systeme, die ihrerseits aus einzelnen Komponenten bestehen, die als vollständige Systeme angesehen werden können. Der Ausdruck System-of-Systems stammt aus der Systemtheorie bzw. dem Systems Engineering. Auch zu diesem Begriff finden sich in der Literatur unterschiedliche Definitionen und Verwendungen. Im Folgenden sollen einige dieser Definitionen vorgestellt werden, um ein tiefergehendes Verständnis für dieses Konzept zu schaffen und einige Aspekte dazu im Zusammenhang mit dem Platooning näher zu erläutern.

Ackoff [12] liefert eine der ersten Definitionen für ein System-of-Systems. In seiner Arbeit versucht er die Begrifflichkeiten im Bereich der Systemtheorie zusammenzufassen und in einem einheitlichen System zu beschreiben. Dafür definiert er verschiede Begriffe im Zusammenhang mit Systemen, wie den Systemkontext, verschiedene Arten von Systemen und Verhaltensweisen von Systemen.
Im weiteren Verlauf seiner Ausführungen definiert er Organisationen und Organismen als zwei Arten von Zusammenschlüssen von Systemen. Er verwendet selbst noch nicht die Bezeichnung SoS. Eine Organisation besteht dabei aus mindestens zwei Einheiten, die jeder für sich auf verschiedene Weisen eigene Ziele verfolgen (auch zweckgebundene Einheiten genannt). Um als Organisiation zu gelten, müssen diese Einheiten zudem einen gemeinsamen Zweck erfüllen bzw. ein gemeinsames Ziel verfolgen. Andernfalls würden die einzelnen Einheiten nicht kooperieren und sich folglich auch nicht organisieren. Um diesen gemeinsamen Zweck zu erreichen, gibt es in einer Organisation eine Aufteilung der notwendigen Arbeiten unter den Einzelelementen. Diese Aufteilung kann entweder funktional, räumlich oder zeitlich erfolgen. Ein weiteres wichtiges Merkmal einer Organisation ist nach Ackoff, dass die Systeme in einer Organisation auf das Verhalten der anderen durch Beobachtung oder Kommunikation reagieren können. Organismen stellen ebenfalls Zusammenschlüsse mehrerer Elemente dar, die gemeinsam als Einheit einen Zweck erfüllen. Im Gegensatz zu Organisationen verfolgen die Elemente in einem Organismus aber für sich keine eigenen Ziele und haben keinen eigenen Willen. Für Ackoff besteht darin der große Unterschied beim Zusammenschluss einzelner Systeme.

Eine ähnliche Definition ist in [9] zu finden. Dort wird auch explizit der Begriff System-of-Systems verwendet. Ein SoS ist danach eine Ansammlung von Komponenten, die individuell als Systeme angesehen werden können und zwei zusätzliche Eigenschaften besitzen: eine betriebliche Unabhängigkeit, sowie eine Unabhängigkeit im Management der einzelnen Komponenten. Mit betrieblicher Unabhängigkeit ist dabei gemeint, dass jede Komponente auch nach einer Auflösung des SoS eigenständig sinnvoll operieren kann und einen Zweck erfüllt.

Unabhängigkeit im Management bedeutet, dass die Systeme, auch wenn sie in ein SoS integriert werden, ein gewisses Maß an Selbstbestimmung behalten. Jedes System behält auch als Teil eines SoS eine betriebliche Existenz.

Sage und Cuppan [13] definieren ein System-of-Systems anhand einiger Charakteristiken. Ein SoS liegt demnach vor, wenn alle oder ein großer Teil der folgenden Kriterien erfüllt sind:

- Die Einzelkomponenten erfüllen voneinander abgegrenzte Aufgaben und haben eigene Ziele. Auch als eigenständige Systeme außerhalb eines SoS werden diese Ziele weiterverfolgt.
- Ein SoS wird zu großen Teilen durch dezentrales Management der Einzelsysteme chrakterisiert.
- Teilweise wird ein SoS durch geographische Trennung der Einzelsysteme chrakterisiert.
- Die Funktionalität des SoS entsteht schrittweise und ist nicht von Beginn an vollständig vorhanden, sondern wird stetig erweitert oder verändert.
- Es entstehen weitergehende Funktionen über die Möglichkeiten der Einzelsysteme hinaus.

Als weitergehende Form führen sie sogenannte Federations-of-Systems (FOS) ein. Diese sind noch stärker durch eine Unabhängigkeit der Einzelsysteme gekennzeichnet, das heißt es gibt keine zentral koordinierende Stelle im System. Die Zusammenarbeit in einem solchen System basiert auf dem Bestreben der Systeme das oder die Ziele des gesamten Verbundes zu erfüllen. Die Autoren führen aus, dass für heutzutage immer komplexer werdende Systeme, die sie als sogenannte Complex Adaptive Systems (CAS) bezeichnen, andere Ansätze während des Designs und der Entwicklung notwendig sind als für monolithische Systeme. Dafür nennen sie vor allem Systems Engineering Prinzipien für die zuvor beschriebenen föderalen Systeme, sowie die Berücksichtigung von evolutionären Lebenszyklen bei der Entwicklung von Systemfunktionalitäten und deren Erwerb.

Ein wichtiger Punkt bei der Betrachtung von System-of-Systems ist die Abgrenzung zu herkömmlichen Systemen. Boardman und Sauser [14] arbeiten in ihrem Paper die Hauptmerkmale heraus, die ein SoS charakterisieren und vor allem solche, die ein SoS von einem klassischen monolithischen System unterscheiden.
Die Autoren definieren als grundlegendes Element eines Systems, dass dieses existiert und geschaffen wurde, um einen bestimmten Zweck zu erfüllen. Einzelne Komponenten, die für sich keinen Zweck erfüllen, definieren sie als Teile oder Subsysteme eines Systems. Ein System-of-Systems ist folglich ein Verbund von eigenständigen Systemen, die, um ihren bzw. einen übergeordneten Zweck zu erfüllen, beschlossen haben in gewisser Art und Weise zusammenzuarbeiten. Wichtig ist dabei, dass sie nie vollständig ihre Autonomie verlieren. Ein weiteres wichtiges Merkmal eines SoS ist die Kommunikation der Einzelsysteme. Diese müssen zur Zusammenarbeit Schnittstellen für die anderen Systeme bereitstellen, die normalerweise für den Nutzer nicht sichtbar sind. Diese können interne Services oder Daten sein, die den anderen SoS Mitgliedern zur Verfügung gestellt werden. Außerdem charakterisieren sie ein System-of-Systems als vielfältig in seinen Möglichkeiten im Vergleich zu monolithischen Systemen. Diese Vielfalt an Funktionen und Möglichkeiten entsteht durch den Zusammenschluss von

mehreren einzelnen Systemen und ist laut den Autoren sogar nötig für ein SoS, um auf die vielen Unsicherheiten in der Umwelt aber auch der eigenen Zusammensetzung reagieren zu können. Ein weiterer Unterschied zwischen Systemen und Systems-of-Systems ist die Art und Weise, in der sie ihre Funktionen hervorbringen. Bei Systemen werden die Funktionen während des Designs definiert und anschließend getestet. Systems-of-Systems hingegen werden beim Design lediglich mit Möglichkeiten zur Funktionserbringung ausgestattet. Die tatsächlichen Funktionen entstehen erst zur Laufzeit aufgrund der Zusammensetzung und der Umwelteinflüsse.

Innerhalb des Feldes der SoS gibt es verschiedene Ansätze für die Realisierung. Systems-of-Systems lassen sich dabei grob in drei Klassen einteilen.
(1) Auf der einen Seite gibt es zentrale SoS. Bei solchen Systemen gibt es eine zentrale Instanz, die langfristig den Betrieb kontrolliert, um zu gewährleisten, dass der gesamte Verbund die gesetzten Ziele erfüllt. Die Einzelsysteme behalten dabei zwar einen Teil ihrer Unabhängigkeit, generell wird der eigene Betrieb aber den Vorgaben des zentralen Systems untergeordnet.
(2) Im Gegensatz dazu stehen dezentrale SoS. In solchen SoS kann es auch ein zentrales Management geben, es hat jedoch keinen Vorrang gegenüber den Entscheidungen der Einzelsysteme. Die Teilsysteme müssen also aus eigenem Antrieb zusammenarbeiten. Auch die Ziele des gesamten SoS werden dabei nicht zentral festgelegt, sondern von allen Teilsystemen gemeinsam bestimmt. Aufgrund dieser Eigenschaft werden solche dezentralen SoS oft auch als kollaborative Systeme bezeichnet.
(3) Eine Erweiterung zu den dezentralen SoS stellen sogenannte virtuelle SoS dar. In solchen Systemen gibt es weder eine zentral koordinierende Einheit noch gemeinsam abgestimmte Ziele. Die Entstehung solcher Systeme kann zufällig geschehen, wobei ein übergeordnetes Verhalten der Einzelsysteme entstehen kann und zum Teil auch erwünscht ist. Insgesamt muss sich das SoS aber auf vage Mechanismen verlassen, die zur Erhaltung dieses Verhaltens beitragen. [9]

2.3 Platooning als System-of-Systems

Nachdem in den vorherigen Abschnitten das Platooning und System-of-Systems getrennt voneinander vorgestellt und definiert wurden, können Gemeinsamkeiten der beiden Konzepte ausgemacht werden. Diese Überschneidungen sollen im Folgenden dargestellt werden, um zu zeigen, dass es sinnvoll ist ein Platoon als SoS zu beschreiben.

Ein System-of-Systems ist teilweise durch eine räumliche Trennung der beteiligten Systeme gekennzeichnet. Jedes System ist eindeutig als physische Instanz identifizierbar. Im Fall des Platoonings ist eine solche Instanz ein Fahrzeug. Die einzelnen Fahrzeuge haben jeweils eigene Fahrtziele und Präferenzen hinsichtlich der zu fahrenden Geschwindigkeit und Beschleunigung. Sie verfolgen also ihre eigenen Ziele und haben auch einen eigenen Willen. Das ist sowohl der Fall, wenn ein menschlicher Fahrer die Kontrolle hat, aber auch wenn ein Computersystem das Fahrzeug steuert. Diese Eigenständigkeit der Systeme ist, wie zuvor beschrieben, eine der wichtigsten Eigenschaft bei der Charakterisierung eines System-of-Systems. Darüber hinaus gibt es auch keine zentrale, übergeordnete Instanz, die die Kontrolle über alle Fahrzeuge

hat und beispielsweise das Formen eines Platoons erzwingen kann. Die Platoon Mitglieder müssen aus ihrem eigenen Willen heraus mit den anderen Fahrzeugen kooperieren und dies andauernd fortführen, um das Platoon aufzubauen und zu erhalten. Dieser Aspekt kennzeichnet ein Platooning-System eindeutig als kollaboratives System. Als Motivation zur Kooperation dienen die sich entwickelnden Fähigkeiten des Platoons, die über die der Einzelfahrzeuge hinausgehen. Dazu zählen z.B. die insgesamt steigende Effizienz in der Nutzung der Straßeninfrastruktur oder der sinkende Kraftstoffverbrauch durch das Fahren mit geringen Abständen. Beides sind für die Fahrzeuge erstrebenswerte Ziele, die sie allein jedoch nie in dieser Form erreichen können. Diese individuellen Ziele der Systeme werden damit zu gemeinsamen Zielen des SoS und fördern die Zusammenarbeit. Kann eines der Systeme seine individuellen Ziele innerhalb des Platoons aber nicht mehr erfüllen, besteht jederzeit die Möglichkeit dieses zu verlassen und in der Folge selbstständig zu agieren. Da jedes Fahrzeug eigenständig seine Lenkung und Geschwindigkeit kontrolliert, ist auch nach der Auflösung des System-of-Systems gewährleistet, dass alle Teilsysteme als eigenständige Systeme einen sinnvollen Betrieb fortführen können. Die Systeme erfüllen also das Kriterium der Unabhängigkeit voneinander, aber auch vom SoS insgesamt. Ein Platooning-System lässt sich also sehr gut als System-of-Systems klassifizieren. Darüber hinaus sind auch eindeutige Merkmale eines kollaborativen Systems zu identifizieren.

3 Platooning Ansätze

Für die Realisierung von Platooning-Systemen gibt es sehr verschiedene Ansätze. Die Grundlagen dafür stellen Konzepte aus den Bereichen künstliche Intelligenz, Robotik und AHS dar. Im folgenden Abschnitt sollen einige dieser Konzepte vorgestellt werden.

3.1 AHS Architekturen

Im Rahmen von großen Forschungsprojekten werden Systeme für das automatisierte Fahren auf Autobahnen entwickelt. Solche Systeme werden als *Automated Highway Systems* (AHS) bezeichnet. Dabei soll für beliebige Fahrzeuge ein komplett autonomer Betrieb auf solchen Hochgeschwindigkeitsstrecken ermöglicht werden. Ein wesentlicher Bestandteil dieser Projekte ist der Zusammenschluss von dicht beieinander fahrenden Fahrzeugen zu Platoons. So sollen die Sicherheit und die mögliche Verkehrsdichte auf den Autobahnen erhöht werden. Beispiele für solche Projekte sind *PATH*, *DOLPHIN* oder *Auto ´21*.

Die Architekturen für das Platooning sind dabei in allen Projekten sehr ähnlich. Es gibt in jedem Fahrzeug eine Schicht, die für dessen Aktorik und Sensorik zuständig ist. Dort werden die möglichen Aktionen des Fahrzeuges realisiert. Darüber liegt eine Schicht, die Pläne für die zukünftigen Handlungen des Fahrzeuges entwickelt. Dort findet zusätzlich der Austausch mit den anderen Fahrzeugen im Platoon statt und es werden gemeinsame Pläne und Ziele abgestimmt. Oft ist auch eine Kommunikation nicht nur innerhalb eines Platoons, sondern auch mit anderen Platoons vorgesehen. Über diesem Management Layer gibt es weiterhin eine Schicht für die Steuerung des Verkehrs auf dem gesamten Streckennetz. Dort ist eine Kommunikation der Fahrzeuge mit infrastrukturseitigen Systemen vorgesehen. Damit sind beispielsweise Ampeln oder andere Signaleinrichtungen zur Steuerung des Verkehrs gemeint, aber auch Systeme, die das Verkehrsaufkommen messen und diese Informationen bereitstellen, um eine gleichmäßige Verteilung der Verkehrsströme zu ermöglichen. [4; 10; 15]

3.2 Agentenbasierte Ansätze

Das Konzept eines sogenannten Agenten stammt aus dem Feld der künstlichen Intelligenz. In [16] wird ein Agent als eine Einheit definiert, die reaktiv, proaktiv, autonom und sozial ist. Diese Einheit handelt auf Basis von Informationen, die sie aus der Umwelt zusammengetragen hat. Solche Systeme werden oft auch als intelligente Agenten bezeichnet.

3.2.1 Hybridarchitekturen

Für den Einsatz in dynamischen Umgebungen werden robuste Modelle für Agenten benötigt. Solche Ansätze werden als Hybridarchitekturen bezeichnet. Die Grundstruktur solcher Agenten ist dabei oft recht ähnlich. Sie besitzen sowohl Module, die für die Reaktion auf Veränderungen des internen Status oder in der Umwelt zuständig sind, als auch Module, die die Entwicklung

von Plänen übernehmen. Oft gibt es zusätzliche Module für die Anpassung und Optimierung der Prozesse im System im Hinblick auf die Umgebungsbedingungen. [4]

Ein einfaches Beispiel für eine solche Architektur ist die *3T intelligent architecture* [17] (siehe Abbildung 1). Die unterste Ebene dieser Architektur stellen die *Reactive Skills* (engl. Reaktionsfähigkeiten) dar. In dieser Schicht sind einfache Funktionen definiert, die auf die Umwelt reagieren und dabei entweder einen bestimmten Zustand des Systems erreichen bzw. beibehalten sollen. Darüber liegt die *Sequencing* Schicht. Hier werden kontinuierliche Steuerungsprozesse angestoßen, indem entsprechende Funktionen der *Reactive Skills* Schicht aktiviert oder deaktiviert werden. Die oberste Schicht ist für die Entwicklung von Plänen zur Bewältigung von komplexen Aufgaben zuständig. Die Entscheidungen werden auf der Basis von Zielen und verfügbaren Ressourcen getroffen. Als Ergebnis dieser Planung werden auszuführende Aufgaben an die darunterliegende Schicht übergeben. Ein ähnlicher Ansatz wird auch in [18] beschrieben. Dort laufen die Reaktions- und die Planungsinstanz in unterschiedlichen Prozessen. Beide Komponenten koordinieren sich selbstständig, um die gesetzten Ziele zu erfüllen. Der Planer passt sich dabei der Reaktionskomponente an. So wird gewährleistet, dass die Reaktionskomponente jederzeit eigenständig in Echtzeit agieren bzw. reagieren kann. Parallel prüft der Planer die ausgeführten Aktionen hinsichtlich der Erfüllung der Ziele. Weichen diese voneinander ab, ändert die Planungskomponente die Konfiguration der Reaktionskomponente, um das Erreichen der Ziele zu gewährleisten. In [19] wird ein ähnlicher Ansatz auf der Basis von Touring Maschinen beschrieben.

Lygeros et al. [20] stellen eine etwas andere Hybridarchitektur vor, die speziell für mobile Agenten entwickelt wurde. Diese beschreibt ausschließlich die Steuerung der Bewegung des Systems. Dafür wird eine diskrete und eine kontinuierliche Schicht definiert. In der diskreten Schicht werden Wegpunkte festgelegt, die gemäß des aktuellen Umfelds ausgewählt werden. Die kontinuierliche Schicht besteht aus mehreren Entscheidungseinheiten. Dabei ist jeweils eine Entscheidungseinheit für einen der Wegpunkte zuständig. Die einzelnen Einheiten bilden anhand einer gemeinsamen Kostenfunktion die Kosten für den jeweiligen Wegpunkt. Über

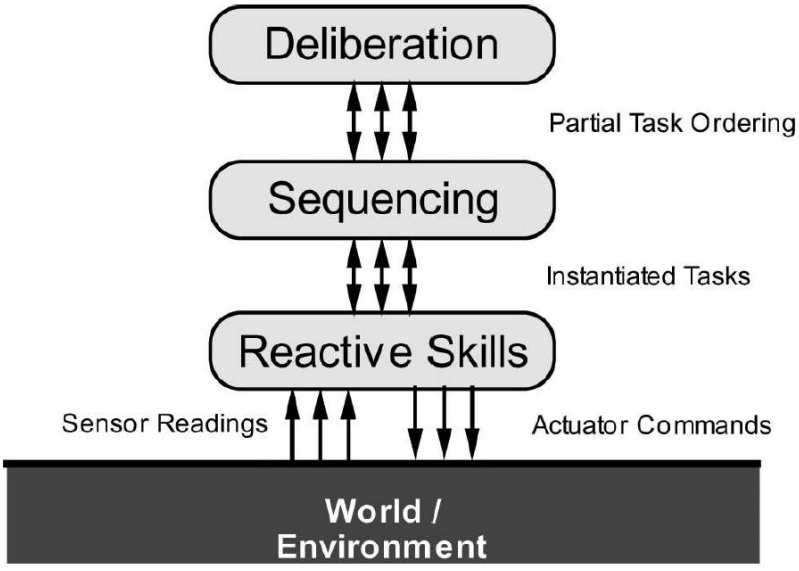


Abbildung 1: 3T intelligent architecture [17]

diese Kosten stehen die Entscheidungseinheiten miteinander im Wettkampf. Die finale Entscheidung der zu fahrenden Trajektorie wird auf Basis dieser Kosten gebildet, unter Berücksichtigung von Sicherheitsbeschränkungen. Auch Sun und Peterson verfolgen mit CLARION-RBF [21] einen etwas anderen Ansatz. In dieser Architektur wird das Verhalten des Agenten durch erlernte Regeln bestimmt. Die Architektur sieht dabei mehrere Schichten vor, in denen auf unterschiedliche Arten Regeln erlernt werden. So kann das System in den unterschiedlichsten Situationen Regeln für sein eigenes Verhalten extrahieren.

3.2.2 Interagierende Agenten

Neben den Hybridarchitekturen gibt es auch Ansätze, die sich vor allem mit der Interaktion mehrerer einzelner Agenten beschäftigen. Solche Systeme werden interagierende Agenten genannt. Ein weit verbreitetes Konzept für solche Agenten ist das Belief-Desire-Intention (BDI) Modell. In [22] wird ein solches Modell auf der Basis des *Robot Operating Systems 2* (ROS2) entwickelt (siehe Abbildung 2). Ein BDI Agent repräsentiert sein Wissen über den eigenen Zustand sowie seine Umwelt als sogenannte Beliefs. Diese dienen als Basis für alle Operationen des Agenten. Weiterhin besitzt ein solcher Agent eine Anzahl von Desires, welche die eigenen Wünsche darstellen. Entsprechend der aktuellen Situation werden einige der Desires aktiviert oder deaktiviert. Ist ein Desire aktiv, so wird es auch als Ziel des Agenten bezeichnet. Er versucht dann mit seinen Handlungen diesen Zielzustand zu erreichen. Dabei verfolgt der Agent bestimmte Pläne oder Strategien, die entweder während des Designs festgelegt oder zur Laufzeit selbstständig entwickelt werden können. Diese sogenannten Intentions stellen das dritte Element eines BDI Agenten dar. Während des Betriebs wählt die Planungsinstanz des Agenten dann basierend auf den gespeicherten Beliefs und aktiven Zielen die passenden und verfügbaren

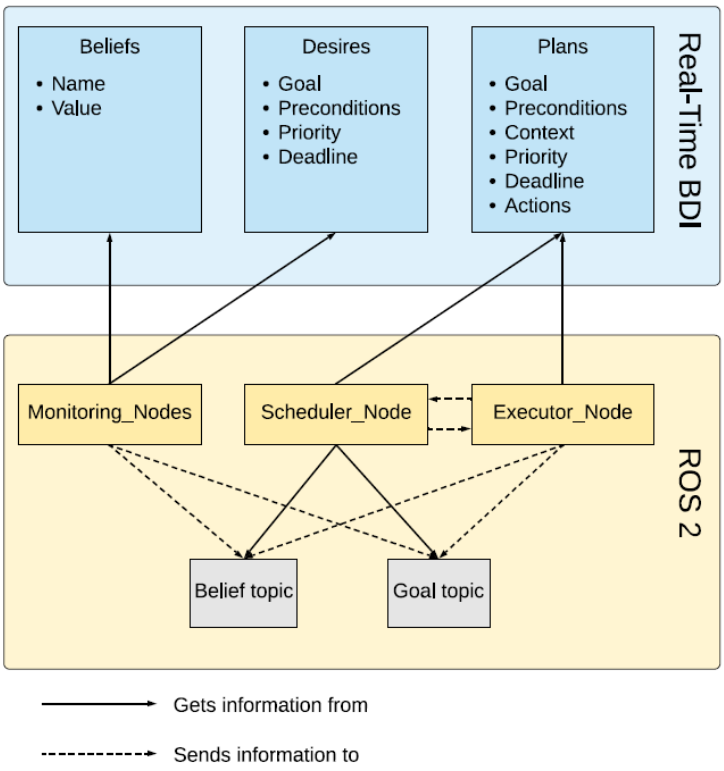


Abbildung 2: ROS 2 BDI Architektur [22]

Strategien aus. Burmeister und Sundermeyer haben mit der COSY Architektur [23] einen sehr ähnlichen Ansatz verfolgt. Bei ihnen werden die einzelnen Elemente (Belief, Desire, Intention) des Agenten in Kategorien eingeteilt, um eine effektivere Planung oder Wissensdarstellung zu gewährleisten. Auch diese Architektur ist stark auf die Interaktion von mehreren Einheiten ausgerichtet.

Noch stärker auf die Zusammenarbeit von Agenten ausgerichtet ist die GRATE Architektur von Jennings [24]. In diesem Modell besteht ein Agent aus zwei Schichten. Eine Schicht ist für die lokale Planung und Ausführung der Handlungen des Agenten zuständig. Die andere Schicht ist für den Austausch mit anderen Agenten verantwortlich. Dadurch ist es den Einzelsystemen möglich gemeinsame Ziele und Handlungen festzulegen. Es können auch Services und Informationen zwischen den Systemen ausgetauscht werden. Die Autoren bezeichnen die enge Kooperation der Agenten auch als Teamwork. Auch das COM-MTDP Framework [25] von Pynadath und Tambe basiert auf der Festlegung von gemeinsamen Intentionen. Die Systeme verfolgen gemeinsam als Team dieselben Ziele, legen ihre Handlungen jedoch eigenständig fest.

3.2.3 Hierarchische Architekturen

In der Literatur werden auch Kombinationen aus hybriden und interagierenden Architekturen vorgestellt. Diese werden als hierarchische Architekturen bezeichnet. Ein Beispiel dafür ist die in [26] vorgestellte Architektur. Diese basiert auf einem Skill-Rule-Knowledgement Modell. Damit sind mehrere Schichten gemeint, die hierarchisch aufeinander aufbauend das Verhalten des Agenten bestimmen. Die Architektur ist in Abbildung 3 zu sehen. Zunächst werden die Informationen aus der Umwelt im *Perception* Modul aufgenommen und in Aktionen oder Ziele klassifiziert. Wird eine Aktion ausgemacht, wird sofort die entsprechende Routine im *Execution* Modul ausgeführt, um auf die eingehende Information zu reagieren. Diese Schicht entspricht

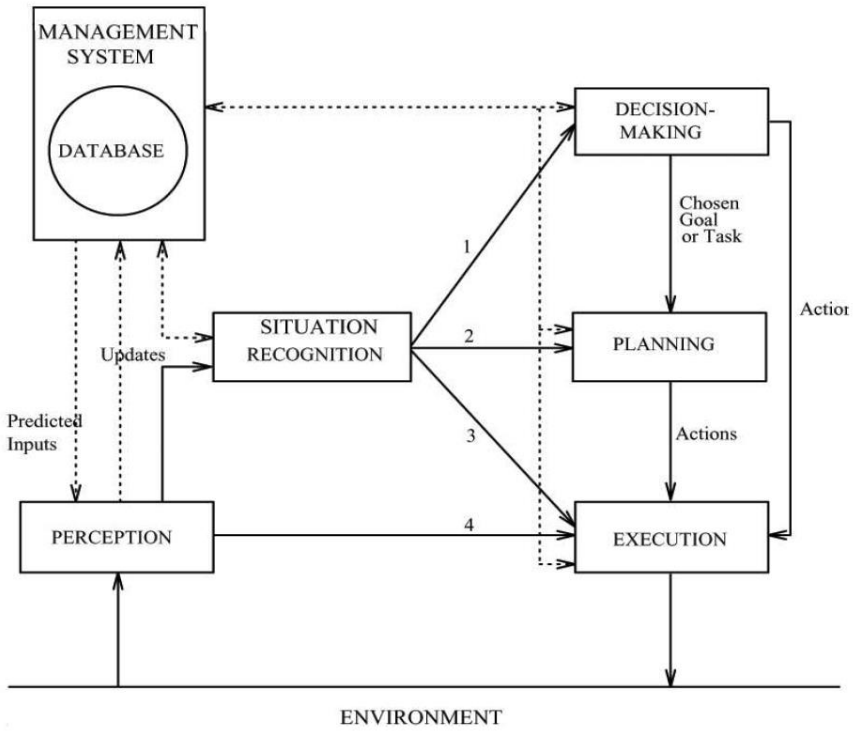


Abbildung 3: SRK basierte Architektur [26]

dem Skill-Layer. Wird ein Ziel erkannt, welches nicht durch eine einfache Routine erreicht werden kann, wird dieses an das *Planning* Modul weitergegeben. Dort wird ein Plan als eine Abfolge von Handlungen erstellt, mit denen das Ziel erreicht werden soll. Gelingt es dem Agenten nicht einen Plan festzulegen oder kann ein Ziel nicht erreicht werden, übernimmt das *Decision Making* Modul die Entscheidungsfindung. Dort wird basierend auf gelernten Erfahrungswerten oder auch dem Austausch mit anderen Agenten die Entscheidung getroffen, welche Handlungen die besten Chancen haben das gesetzte Ziel zu erreichen.

Auch die INTERRAP Architektur von Müller und Pischel [27] ist hierarchisch aufgebaut. Es gibt auch hier eine Schicht für die Ausführung von primitiven Handlungen und direkten Reaktionen auf Veränderungen der Umwelt. Diese wird *Behaviour-Based-Layer* genannt. Darüber liegt mit der *Local-Planning-Layer*, wie in der zuvor beschriebenen Architektur, eine Schicht für die Planung von komplexeren Abläufen. Die oberste Schicht der INTERRAP Architektur wird als *Cooperative-Planning-Layer* bezeichnet. Sie ist ausschließlich für die Kommunikation mit anderen Agenten und die Abstimmung von Zielen und Handlungen zuständig.

3.3 Architekturen aus der Robotik

Im Bereich der Robotik gibt es ebenfalls einige Ansätze, die wichtige Aspekte eines Platooning-Systems realisieren. Dabei stehen vor allem die Handlungsplanung und die Fahrzeugführung im Fokus. Mit der HLSA (Hierarchical Learning reinforcement Stochastic Automaton) Architektur [28] entwickeln Lima und Saridis beispielsweise einen sehr leistungsfähigen Ansatz für die Planung und Ausführung von Aufgaben für ein System (siehe Abbildung 4). Die Architektur besteht aus drei Hierarchieebenen, wobei die auszuführenden Kommandos an die oberste Schicht gegeben werden. Diese ermittelt daraufhin eine Anzahl von verschiedenen Tasks, die jeweils eine Ausführung des eingegangenen Kommandos ermöglichen. Die mittlere Schicht erhält einen solchen Task als Eingang und entwickelt auf dieser Basis eine detaillierte Abfolge von primitiven Handlungen. Als letzter Schritt werden diese Handlungen an die für die jeweilige Ausführung zuständigen Einheiten im Roboter gesendet, welche als Ergebnis die Aktuatoren ansteuern. Zwischen zwei benachbarten Schichten der Architektur gibt es jeweils ein Schnittstellen Modul. Diese Module sind für die Auswahl und Weiterleitung einer der von der höheren Schicht ermittelten Ausgaben an die nächste Schicht zuständig. Zudem wird in den Schnittstellen Modulen eine Performance Funktion von der unteren zur oberen Schicht weitergegeben, die angibt, wie gut die ausgewählten Handlungen eine Aufgabe erfüllt haben. Anhand dieser Funktion kann das System über die Schnittstellen lernen, welche Handlungen ausgewählt werden müssen, um das beste Ergebnis zu erzielen. Andere Ansätze wie [29] verwenden für die Aufgabenplanung eine zentrale Einheit, die eine Kommunikation zwischen verschiedenen roboterspezifischen Modulen steuert und überwacht, um so die Ziele des Roboters zu erreichen. Ähnlich aufgebaut ist auch die DAMN Architektur (Distributed Architecture for Mobile Navigation) [30]. Auch dort gibt es eine zentrale Einheit. In diesem Fall kommuniziert diese Einheit jedoch mit Controllern für bestimmte vordefinierte Verhaltensweisen des Roboters. Die Controller geben jeweils entsprechend ihrer Priorität ein Gebot ab. Diese Gebote werden in der

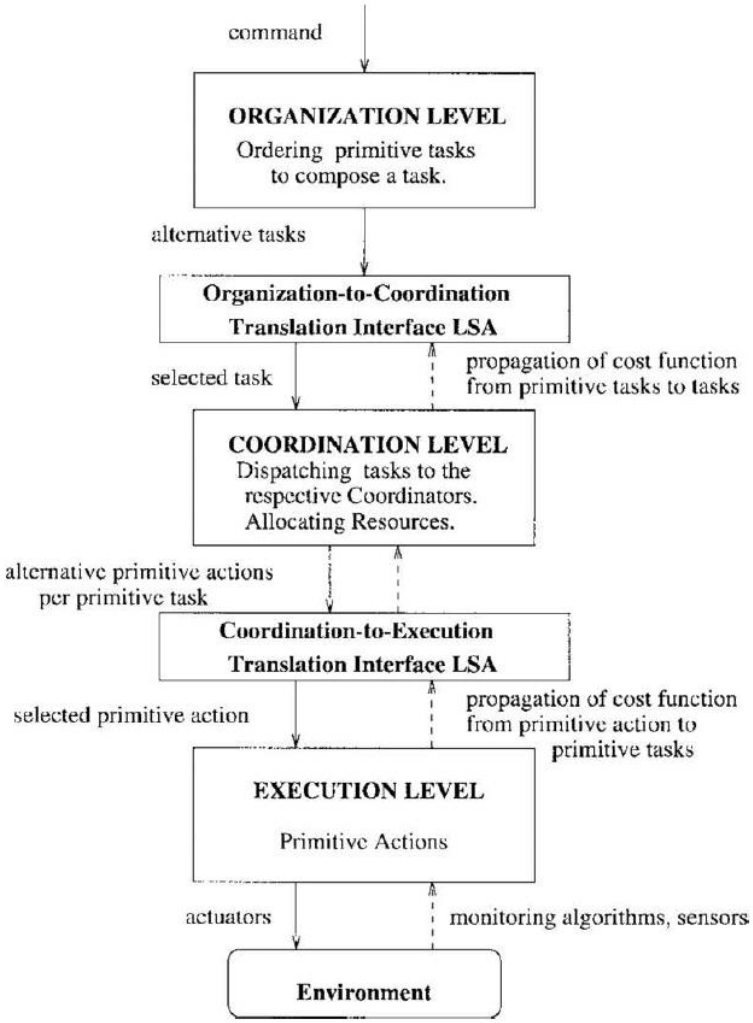


Abbildung 4: Hierarchical Learning reinforcement Stochastic Automaton Architektur [28]

zentralen Einheit miteinander kombiniert und in entsprechende Steuerkommandos für die Aktoren umgewandelt. Dadurch ist das Verfolgen mehrerer Ziele parallel möglich.

Für das tatsächliche Fahren im Platoon wird hauptsächlich auf den Einsatz von Reglern zurückgegriffen. In [31] wird ein Regler vorgestellt, der ein Fahrzeug, welches einem Platoon beitreten möchte, auf einer möglichst sanften Trajektorie zum Platoon führt. Es wird dabei ausschließlich dieser Aspekt der Annäherungsphase eines Fahrzeuges zu einem Platoon betrachtet. Während der Fahrt in einem Platoon ist vor allem das Beibehalten des Abstandes zum vorausfahrenden Fahrzeug wichtig. Dafür hat Santini [11] eine Algorithmus vorgestellt, der den Abstand zum vorderen Fahrzeug und die Geschwindigkeit regelt. Dabei werden Informationen der anderen Fahrzeuge und des eigenen verwendet, um Fahrkommandos zu berechnen. Ziel dieses Ansatzes ist vor allem die Stabilität des Platoons zu gewährleisten, auch bei heterogener Zusammensetzung und instabiler Kommunikation. Mit der SAPIENT Architektur (Situation Awareness Planner Implementing Effective Navigation in Traffic) [32] haben Sukthankar et al. einen etwas ungewöhnlicheren Ansatz für die Fahrzeugführung entwickelt. Sie verwenden für verschiedene physikalische Aspekte des Fahrzeuges, wie die Umgebung an einer Fahrzeugecke oder die Geschwindigkeit, jeweils einen Agenten in ihrer Architektur. Jeder dieser Agenten bewertet die Auswirkungen, die sein Bereich auf zukünftige Aktionen des gesamten Fahrzeuges haben wird.

Anhand dieser Bewertung wählt jeder Agent die günstigste Route aus. In einer zentralen Steuereinheit werden diese Routen anschließend entsprechend ihrer Gewichtungen zu einer resultierenden Route kombiniert. In diesem Ansatz werden zwar Agenten verwendet, allerdings nur für die Fahrzeugführung und nicht zur Repräsentation eines gesamten Fahrzeuges innerhalb eines Platoons. Aus diesem Grund wurde dieser auch nicht zu den agentenbasierten Ansätzen eingeordnet. Die Architektur soll lediglich ein intelligentes Fahren ermöglichen.

4 Entwurf und Design für das CeCar Platooning-System

Im folgenden Teil der Arbeit soll ein Platooning-System für die CeCar-Plattform spezifiziert und entworfen werden. Dieser Entwurf wird auf Basis der vorgestellten Platooning Konzepte entwickelt. Um das Systemdesign nachvollziehbar und möglichst vollständig zu entwerfen, werden Methoden des Systems Engineering angewandt. Die Darstellung des Systemdesigns erfolgt in einem SysML-Modell.

4.1 Vorstellung der CeCar-Plattform

Zunächst sollen in diesem Abschnitt Aufbau und Besonderheiten der CeCar-Plattform näher beschrieben werden. Da das Platooning-System auf Basis dieser Plattform entwickelt werden soll, ist dieser Schritt notwendig, um die Ausgangssituation und daraus abgeleitete Vorgaben für das Platooning-System zu verstehen.

4.1.1 Technische Rahmenbedingungen des CeCar

Das CeCar ist ein Modellauto, welches im Studiengang Computer Engineering am Fachbereich 1 der HTW Berlin entwickelt wurde (siehe Anhang A). Mithilfe dieses Modellautos können die Studierenden Funktionen und Algorithmen für ein autonomes Fahren unter Laborbedingungen entwickeln und testen. Es gibt aktuell vier solcher Modellautos am Fachbereich. Da jedes dieser Fahrzeuge mit unterschiedlicher Sensorik und Software ausgestattet werden kann, bilden sie gemeinsam eine Plattform zum Testen autonomer Fahrfunktionen.

Ein CeCar ist auf Basis eines herkömmlichen ferngesteuerten (RC, engl.: Remote Controlled) Modellautos[1] im Maßstab 1:8 aufgebaut. Das Auto besitzt einen Bürstenlosen Gleichstrommotor (BLDC, engl.: Brushless Directed Current) und Allradantrieb. Die Systemarchitektur und Auswahl der einzelnen Komponenten wurde im Rahmen einer Masterarbeit [33] erstellt. Einen Überblick über die Architektur des CeCars gibt Abbildung 5. Neben den verschiedenen Hardware Komponenten sind vor allem die Realtime Control Unit (RCU) und Master Control Unit (MCU) für die Entwicklung des Platooning-Systems von Bedeutung.

Die RCU ist für die Echtzeitsteuerung der Aktorik verantwortlich. Dort werden hauptsächlich Steuerbefehle für das Motorsteuergerät und den Lenkservomotor generiert und gesendet. Darüber hinaus werden die Drehzahlmesser der einzelnen Räder ausgewertet, um Informationen über Beschleunigung und Geschwindigkeit des Fahrzeuges bereitzustellen. Auf der MCU ist die Applikationssoftware des Systems zu finden. Diese Einheit fungiert als zentrale Steuereinheit des CeCars. Dort wird der Systemzustand überwacht und gesteuert, sowie die autonomen Fahrfunktionen ausgeführt. Das physische Verhalten des Modellautos wird ebenfalls von

[1] Modell: 1/8 8IGHT-E 4WD Buggy Brushless RTR, siehe https://www.horizonhobby.de/de_DE/product/1-8-8ight-e-4wd-buggy-brushless-rtr-blue-green/LOS04014.html

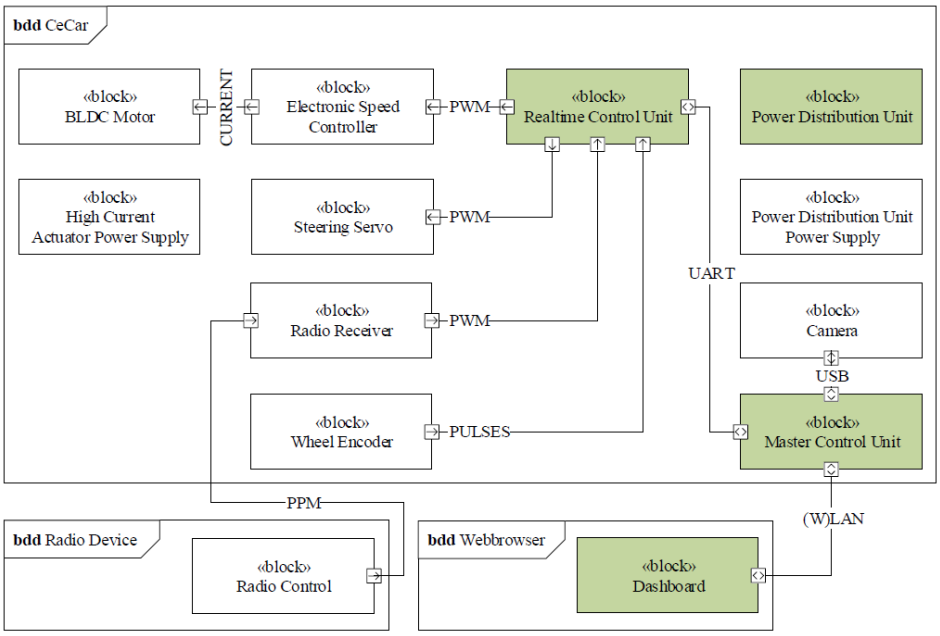


Abbildung 5: Systemarchitektur des CeCars [33]

diesem Steuergerät mittels der RCU gesteuert. Es werden außerdem Daten der verschiedenen Sensoren, mit denen das CeCar ausgerüstet werden kann (LIDAR, Ultraschall, Stereo-Kamera, etc.), ausgewertet und entsprechend verarbeitet.

Die MCU wird mit dem Linux Betriebssystem Ubuntu (Version 18.04 LTS) betrieben. Darauf aufgesetzt arbeitet die MCU mit dem Robot Operating System 2 (ROS2) als Middleware. Das ROS2 ist ein Softwarepaket für die Entwicklung von roboterspezifischer Software. Es stellt viele Software Bibliotheken und Werkzeuge zur Erstellung von Anwendungen für Roboter bereit. Sämtliche Applikationen für das CeCar werden auf dieser Grundlage entwickelt.

Aus den Gegebenheiten des CeCars können einige Bedingungen für das zu entwickelnde Platooning-System abgeleitet werden:

- Verwendung von ROS2,
- Implementierung in C++,
- Nutzung der vorhandenen Schnittstelle zwischen MCU und RCU für Fahrsteuerung,
- Begrenzungen von Geschwindigkeit, Beschleunigung und Lenkwinkel entsprechend der Parameter des Modellautos.

Diese Grundbedingungen sind für alle zu entwickelnden Funktionen und Softwarebestandteile einzuhalten.

4.1.2 Systemkontext für das CeCar Platooning-System

Neben den technischen Gegebenheiten des CeCars ist auch die Analyse des Systemkontextes für das CeCar Platooning-System ein wichtiger Teil in der Beschreibung der Ausgangssituation. Der Systemkontext wird in einem SysML Block-Definition-Diagram (BDD) dargestellt. Dabei repräsentiert jeder Block des Diagramms ein System, mit dem das Platooning-System interagiert. Den vollständigen Systemkontext zeigt Abbildung 6.

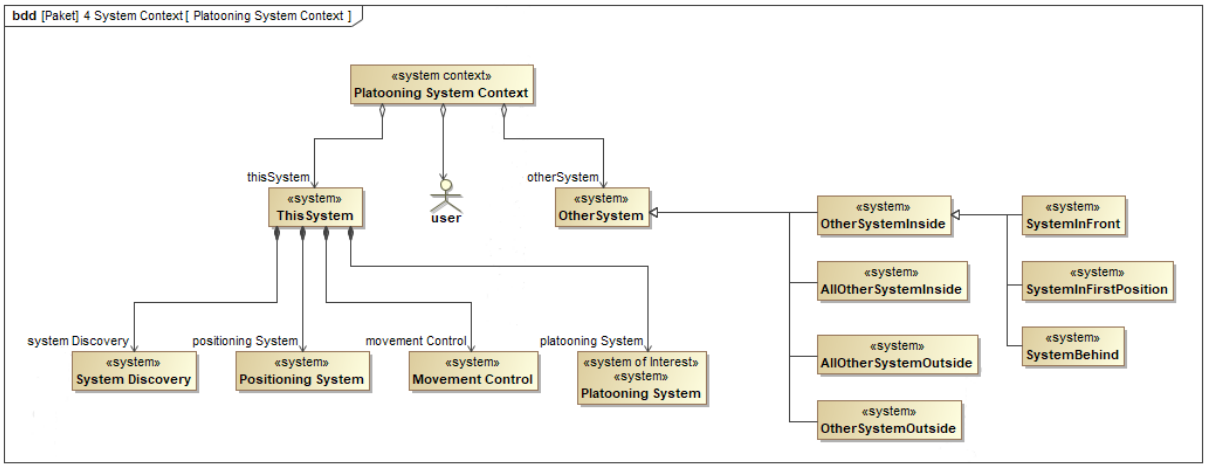


Abbildung 6: Systemkontext für das CeCar Platooning System

In diesem Diagramm ist das zu entwickelnde Platooning-System als System of Interest (SOI) gekennzeichnet. Die anderen Systeme stellen sowohl physische als auch Software Systeme dar, mit denen das Platooning-System interagiert. Das *Platooning-System*, sowie *Movement Control*, *Positioning System* und *System Discovery* sind Software Systeme. Diese sind einem physischen System (*ThisSystem*) zugeordnet, auf dem sie ausgeführt werden. Das System *Movement Control* ist für die Ansteuerung der Aktorik eines CeCars zuständig. Dieses übernimmt die entsprechende Kommunikation mit der RCU und wird für die Umsetzung von Fahrbefehlen verwendet. Das *Positioning System* stellt Informationen zur Position und Lage eines CeCars zur Verfügung. Die Komponente *System Discovery* sammelt Informationen sowohl zum eigenen als auch zu anderen verfügbaren Systemen (CeCars). Zu diesen Informationen gehören die eindeutigen Bezeichnungen der Systeme, sowie deren bereitgestellte Services. Als Ergebnis wird eine Art Lexikon von verfügbaren Services und deren Schnittstellen mit einer Zuordnung zum jeweiligen physischen System bereitgestellt.

Das physische System ist im Fall der CeCar-Plattform ein einzelnes CeCar. Auch die Systeme *user* und *OtherSystem* sind physische Systeme. Unter dem Block *OtherSystem* sind alle Varianten von weiteren CeCars zusammengefasst, mit denen ein einzelnes CeCar interagieren kann. Die Namenszusätze *Inside* und *Outside* beziehen sich dabei auf die Zuordnung zu einem Platoon. Alle Systeme, die mit *Inside* benannt sind, befinden sich im selben Platoon, wie das betrachtete CeCar (*ThisSystem*). Analog befinden sich alle mit *Outside* benannten Systeme entweder in einem anderen oder in keinem Platoon.

Im Rahmen dieser Masterarbeit wird ausschließlich die Software für das SOI entwickelt und implementiert. Alle weiteren Software Komponenten werden als gegeben vorausgesetzt und wurden bereits in anderen Projekten [33; 34] entwickelt. Die vorhandenen Schnittstellen dieser Systeme müssen folglich beim Entwurf des Platooning-Systems berücksichtigt werden.

4.2 Anforderungen für das CeCar Platooning-System

Der erste Schritt für den Entwurf des Platooning-Systems ist die Definition der Anforderungen. Damit soll ein Rahmen für das zu entwickelnde System und seine Funktionen geschaffen

werden, der die Grundannahmen und Einschränkungen für die Aufgabenstellung festlegt. Da es sich bei dem Platooning-System für das CeCar um eine Neuentwicklung für diese Plattform handelt, können im Rahmen dieser Masterarbeit noch nicht alle Möglichkeiten, die ein solches Platooning-System bietet, realisiert werden. Wie bereits in der Einleitung beschrieben, soll ein System entwickelt werden, welches grundlegende Platooning Funktionen implementiert und die Basis für zukünftige Experimente und Projekte mit dem Platooning ermöglicht. In den folgenden Anforderungen (Stakeholder Needs, kurz SN) werden sowohl die zu implementierenden Grundfunktionen als auch einige einschränkende Annahmen für das System beschrieben.

In Tabelle 1 werden die Grundfunktionen, die das Platooning-System realisieren soll, beschrieben. Die wohl wichtigste Funktion ist dabei in Anforderung SN-1 zu finden. Um überhaupt als Platoon existieren zu können, müssen die Mitglieder zum einen über die Zusammensetzung des Platoons informiert sein. Zum anderen muss diese Zusammensetzung durch die Mitglieder verändert werden können. Einige Beispiele dafür sind das Formen des Platoons sowie das Verlassen oder Beitreten von einzelnen Systemen. Die hauptsächliche Aufgabe eines existierenden Platoons stellt das kollaborative Fahren in einer Gruppe von Fahrzeugen dar. Diese Funktionalität leitet sich aus Anforderung SN-4 ab. Dabei wird zunächst nicht festgelegt, wie genau die Formation auszusehen hat, in der die Fahrzeuge gemeinsam fahren. Die Anforderungen SN-2 und SN-3 spezifizieren bereits etwas näher, wie diese Platooning Grundfunktionen umgesetzt werden sollen. Als Grundelement für das Verhalten der einzelnen Systeme sollen diese mit verschiedensten Strategien ausgerüstet sein, die abhängig von den Systemzielen ausgeführt werden. Damit werden wichtige Charakteristiken für ein System-of-Systems, wie in Abschnitt 2.2 beschrieben, in den Anforderungen festgehalten. Jedes System soll unabhängig von seiner Mitgliedschaft in einem Platoon einen eigenen operativen Zweck erfüllen und seine Handlungen auf die Erfüllung dessen ausrichten. Weiterhin sollen im Platoon als SoS weitergehende Funktionalitäten über die Fähigkeiten eines einzelnen Systems hinaus entstehen. Dazu ist es notwendig, dass ein Austausch von Informationen und Funktionen zwischen den Platoon Mitgliedern besteht. Das soll auf Basis von Services geschehen, die jedes einzelne System für die Nutzung durch andere Systeme des Platoons bereitstellt.

ID	Name	Text
SN-1	Manage platoon composition	The platoon shall manage its composition according to the available systems and the platoon goals.
SN-2	Use distributed services	Platoon members shall be able to use services provided by other platoon members.
SN-3	Satisfy goals	The platoon shall act according to goals and use strategies to satisfy them.
SN-4	Driving in Formation	The platoon shall drive as a group in a given formation.

Tabelle 1: Anforderungen für Grundfunktionen des Platooning-Systems

ID	Name	Text
SN-5	Act as decentralized system	The platoon shall act as a decentralized systems, which means there is no master system in the platoon that executes all the planning and reasoning.
SN-6	Build homogeneous platoon	The platoon exclusively consists of CeCars with the same implemented services and available strategies.
SN-7	Communication infrastructure	The platoon shall only use a vehicle to vehicle communication and no vehicle to infrastructure communication.
SN-8	Multiple platoon membership	A System is only member of one Platoon at the same time.

Tabelle 2: Anforderungen zur Charakterisierung des Platooning-Systems

Die zusätzlichen Anforderungen in Tabelle 2 legen einige wichtige Eigenschaften des Platooning-Systems fest. Um ein Platoon zu bilden, welches robust gegenüber Systemausfällen ist, soll dieses als ein dezentrales System aufgebaut sein (Anforderung SN-5). Dadurch übernimmt nicht ein einzelnes System eine zentral koordinierende Rolle und muss folglich nicht besonders gegen einen Ausfall abgesichert werden. Mit der Anforderung SN-6 wird eine Einschränkung hinsichtlich der Platoon Zusammensetzung definiert. Ein Platoon soll zunächst einfachheitshalber nur aus gleichen CeCars bestehen. Damit wird der Aufwand für die Implementierung reduziert, um nicht den Rahmen dieser Arbeit zu übersteigen. Da die CeCar-Plattform zum aktuellen Zeitpunkt nur aus Fahrzeugen besteht und keine zusätzlichen Infrastrukturelemente umfasst, soll dementsprechend die Kommunikation des Platooning-Systems ausschließlich zwischen den Fahrzeugen stattfinden. Diese Fahrzeug-zu-Fahrzeug (V2V, engl.: vehicle to vehicle) Kommunikation wird durch die Anforderung SN-7 gefordert. Eine Fahrzeug-zu-Infrastruktur (V2I, engl.: vehicle to infrastructure) Kommunikation wird darüber hinaus explizit ausgeschlossen. Als letztes wird mit Anforderung SN-8 eine weitere Vereinfachung getroffen. Zur Reduzierung der Komplexität soll ein System zunächst nur Mitglied in einem Platoon gleichzeitig sein können. Dadurch wird der Aufwand zur Koordinierung der Platoon Mitgliedschaften auf ein Mindestmaß begrenzt.

4.3 Anwendungsfälle für das CeCar Platooning-System

Mit den zuvor vorgestellten Anforderungen wurde ein Rahmen für das zu entwickelnde Platooning-System abgesteckt. Innerhalb dieses Rahmens können nun die notwendigen Funktionen zur Erfüllung dieser Anforderungen analysiert werden. Als ein wichtiger Schritt im Systems Engineering Prozess werden dazu Anwendungsfälle (UC, engl.: Use Case) aus den Anforderungen abgeleitet. Diese Anwendungsfälle stellen jeweils abgegrenzte Funktionalitäten dar, die zur Realisierung einer oder mehrerer Anforderungen beitragen. Für das CeCar Platooning-System wurden während der funktionalen Analyse 16 solcher Anwendungsfälle ausgemacht. Das entstandene SysML-Use Case Diagramm ist in Abbildung 7 zu sehen. Im Folgenden

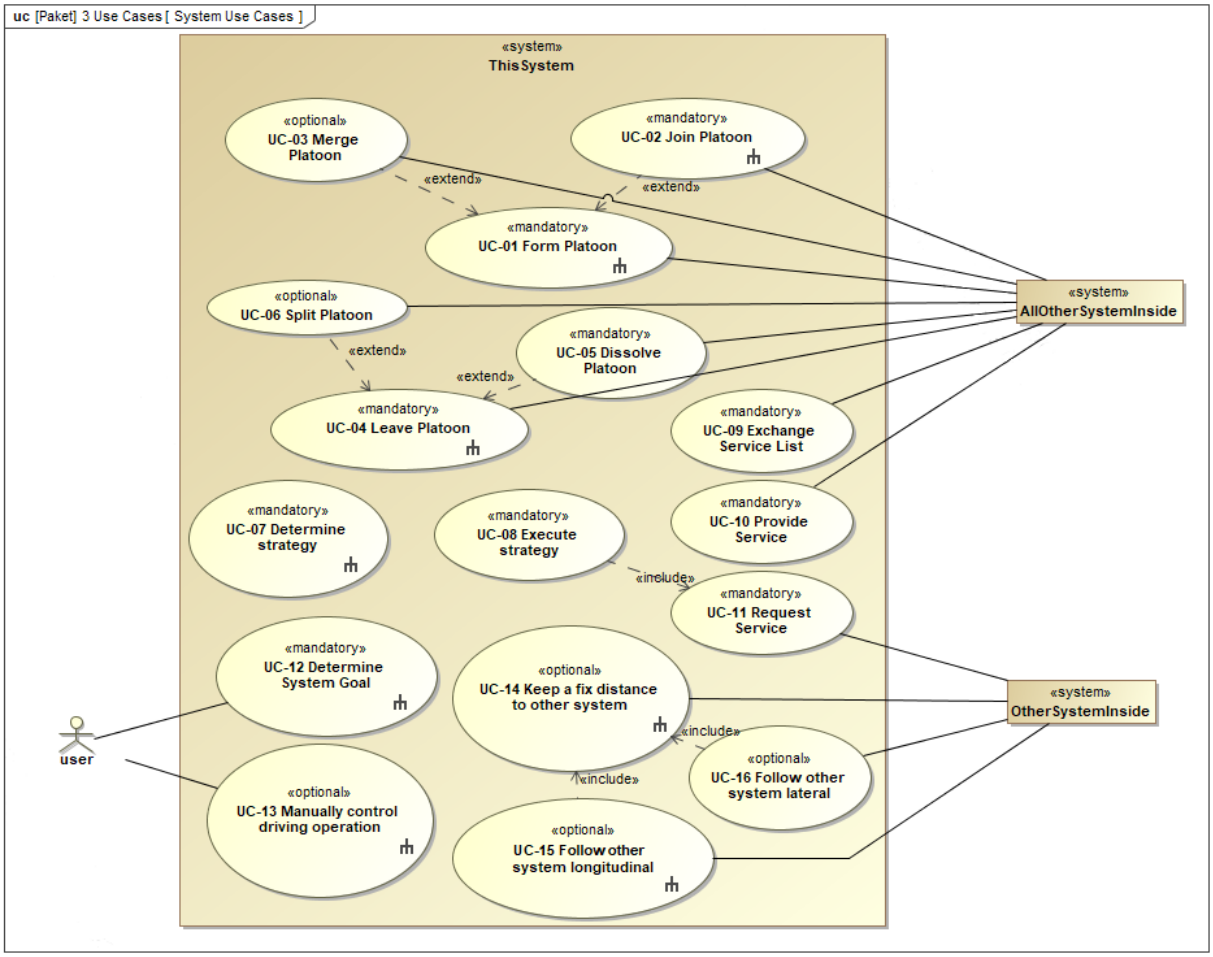


Abbildung 7: Anwendungsfälle für das CeCar Platooning System

sollen die Anwendungsfälle näher betrachtet werden. Dabei ist die Einstufung der einzelnen Use Cases zu beachten. Für die Klassifizierung der Anwendungsfälle wurden die Bezeichner *mandatory* und *optional* eingeführt. Ein mit *mandatory* gekennzeichneter UC beschreibt eine grundlegend notwendige Funktion für das Platooning-System. Die Implementierung eines solchen UC ist im Rahmen dieser Arbeit erforderlich. Anwendungsfälle, die mit *optional* gekennzeichnet sind, stellen dagegen Zusatzfunktionen oder Sonderfälle dar. Sie sind nicht notwendig für ein funktionsfähiges Platooning-System und werden bei der Implementierung daher nicht explizit betrachtet.

Die wohl wichtigste Funktion des Platooning-Systems ist die Steuerung der Platoon Zugehörigkeit. Um die Vorteile eine Platoons nutzen zu können, muss ein System in der Lage sein entweder ein neues Platoon zu formen oder einem bestehenden beizutreten. Die Anwendungsfälle UC-01 und UC-02 repräsentieren genau diese Funktionen. Für eine Implementierung der Platooning Funktionen spielen diese beiden Use Cases daher eine zentrale Rolle. UC-03 ist eine Kombination aus der Bildung eines neuen Platoons und dem Beitritt eines einzelnen Systems. Beim sogenannten merging schließen sich zwei vorhandene Platoons zu einem einzigen größeren Platoon zusammen. Dieser Anwendungsfall stellt einen Sonderfall für die Platoon Bildung dar und ist keine grundlegend notwendige Funktion für das Platooning-System. Aus diesem Grund ist er als *optional* gekennzeichnet. Analog zum Platoon Beitritt muss den Platoon Mitgliedern auch das Verlassen eines Platoons möglich sein. Haben sich beispielsweise die Ziele

eines Systems dahingehend geändert, dass kein gemeinsames Fahren im Platoon mehr sinnvoll erscheint, muss es in der Lage sein aus dem Platoon auszutreten und in der Folge als eigenständiges System zu agieren. Diese Funktionalitäten werden durch UC-04 bis UC-06 repräsentiert. Die Anwendungsfälle *Leave Platoon* und *Dissolve Platoon* entsprechen dabei dem Verlassen des Platoons durch ein einzelnes Fahrzeug bzw. der kompletten Auflösung des Platoons. UC-06 stellt wieder einen Sonderfall dar. Mit diesem Use Case wird die Aufspaltung eines Platoons in mehrere kleinere beschrieben. Auch dieser Anwendungsfall wird zunächst nicht bei der Implementierung berücksichtigt.

In Anforderung SN-2 wird gefordert, dass Services zwischen verschiedenen Systemen im Platoon ausgetauscht werden. Um diese Anforderung zu erfüllen, wurden die Anwendungsfälle UC-09 bis UC-11 definiert. Durch sie wird gewährleistet, dass jedes System Informationen darüber erhält, welche Services im Platoon verfügbar sind und dass diese Services individuell genutzt werden können. Die Nutzung von verteilten Services bildet ein wichtiges Grundkonzept zum Erreichen von Platoon Funktionen, die über die Fähigkeiten der Einzelsysteme hinausgehen. Der Austausch von Informationen über die verfügbaren Services (UC-09) wird bereits durch das System *System Discovery* (siehe Abbildung 6) implementiert und ist deshalb nicht Teil der zu realisierenden Funktionen des Platooning-Systems. Ebenso werden die Funktionalitäten der Anwendungsfälle UC-10 und UC-11 durch das verwendete ROS2 Framework bereitgestellt. In Rahmen dieser Arbeit müssen dafür lediglich die Schnittstellen der Services definiert werden.

Mit UC-07 und UC-08 wird die grundlegende Handlungsplanung des Platooning-Systems beschrieben. Diese Planung soll nach Anforderung SN-3 auf Basis von verschiedenen Strategien erfolgen, wobei je nach den Zielen des Systems eine passende Strategie ausgewählt werden soll. Das System muss also sowohl eine Funktion zur Bestimmung der besten Strategie (UC-07) als auch für die Ausführung einer solchen Strategie (UC-08) bereitstellen. Um die jeweiligen Ziele des Systems festzulegen, soll der Nutzer eine Möglichkeit zum Eingriff in das System haben. Die entsprechende Funktion wird durch UC-12 repräsentiert. Alle weiteren Anwendungsfälle (UC-13 bis UC-16) sind keine Grundfunktionen für das Platooning-System. Diese sind für das kollaborative Fahren im Platoon vorgesehen und sollen in dieser Arbeit zunächst eine untergeordnete Rolle bei der Implementierung spielen. Der Fokus liegt dabei lediglich auf der Schaffung einer Möglichkeit zum Testen des Auswahlprozesses zwischen alternativen Strategien für das Fahren und nicht auf einer fehlerfreien und sicheren Implementierung für das kollaborative Fahren.

Wie zu Beginn dieses Abschnitts erwähnt, sind alle vorgestellten Anwendungsfälle für das Platooning-System aus den Anforderungen abgeleitet. Einen Überblick über die Zuordnung der Use Cases zu den Anforderungen gibt die Matrix in Anhang C.

4.4 Architektur für das CeCar Platooning-System

In diesem Abschnitt soll die Systemarchitektur vorgestellt werden, die ausgewählt wurde, um die Systemfunktionen zu realisieren. Bei dieser Auswahl wurden die in Abschnitt 3 vorgestellten Konzepte für Platooning-Systeme in Betracht gezogen und auf ihre Anwendbarkeit für die CeCar-Plattform hin ausgewertet.

Um eine System-of-Systems Charakteristik des Platoons zu gewährleisten, wurde als Grundlage für die Architektur eine agentenbasierte Struktur ausgewählt. Dabei wird das Platooning-System jedes CeCars durch einen Agenten repräsentiert. Jedes Einzelsystem agiert auf der Basis von eigenem Wissen und eigenen Zielen. Die operative Unabhängigkeit der Systeme als wichtiges SoS Merkmal wird durch dieses Design sichergestellt. Da der Zusammenschluss zu einem Platoon ein hohes Maß an Kooperation zwischen den Einzelsystemen erfordert, bietet sich eine Architektur für interagierende Agenten für das Platooning-System an. Aufgrund der Gegebenheiten der CeCar-Plattform wurde unter den Konzepten aus Abschnitt 3.2.2 die BDI Architektur aus [22] als Grundlage für die Architektur des Platooning-Systems ausgewählt. Zum einen wurde diese Architektur für ein ROS2 System entwickelt. Auch für das CeCar wird das ROS2 Framework verwendet. Diese Gemeinsamkeit ermöglicht eine einfache Adaption des Konzeptes auf das CeCar Platooning-System. Zum anderen gewährleistet die BDI Architektur eine modulare Erweiterbarkeit des Platooning-Systems. Diese ist wichtig, da im Rahmen der Masterarbeit zunächst nur die Grundfunktionen für das Platooning implementiert werden sollen. Die Systemarchitektur muss eine Implementierung von zusätzlichen Funktionen im Rahmen anderer Projekte ermöglichen. Im Fall eines BDI Agenten können einzelne Aspekte des Systems sehr gut getrennt voneinander erweitert werden. So können beispielsweise neue Strategien für den Agenten implementiert werden, ohne dass vorhandene Strategien, Ziele oder Strukturen verändert werden müssen.

Die modifizierte Systemarchitektur für das CeCar Platooning-System ist in Abbildung 8 zu sehen. Dort sind sowohl die gewählten Subsysteme als auch deren Schnittstellen dargestellt. Das Platooning-System wird aus zwei Subsystemen gebildet. Diese werden in dem SysML Internal-Block-Diagram (IBD) durch die Parts Strategy Planning und Strategy Execution repräsentiert. Jedes dieser beiden Subsysteme übernimmt dabei eine grundlegende Aufgabe eines Agenten. Das Subsystem Strategy Planning ist für die Planung und Auswertung der Handlungen des Agenten zuständig. Dort werden alle Informationen, die das System über sich und seine Umwelt erhält, gespeichert. Um der Struktur als BDI Agent gerecht zu werden, geschieht diese Informationsverarbeitung auf der Basis von sogenannten Beliefs. Ein solches Belief repräsentiert ein Wissensobjekt, welches eine Information anhand eines Namens und dazugehörigem Wert speichert. Diese Informationen erhält das Subsystem vor allem über die Schnittstellen zu den anderen Systemen. So werden beispielsweise Daten zu Position und Orientierung des CeCars ermittelt. Darüber hinaus stellt das Strategy Planning System dem Nutzer des Platooning-Systems eine Schnittstelle zur Beeinflussung der Systemziele bereit. Diese Ziele werden im Kontext eines BDI Agenten auch als Desire bezeichnet. Die Hauptaufgabe dieses Subsystems stellt die Handlungsplanung dar. Dadurch werden die sogenannten Intentions des Agenten festgelegt. Alle Handlungen, die das System ausführen kann, sind in Form von Strategien

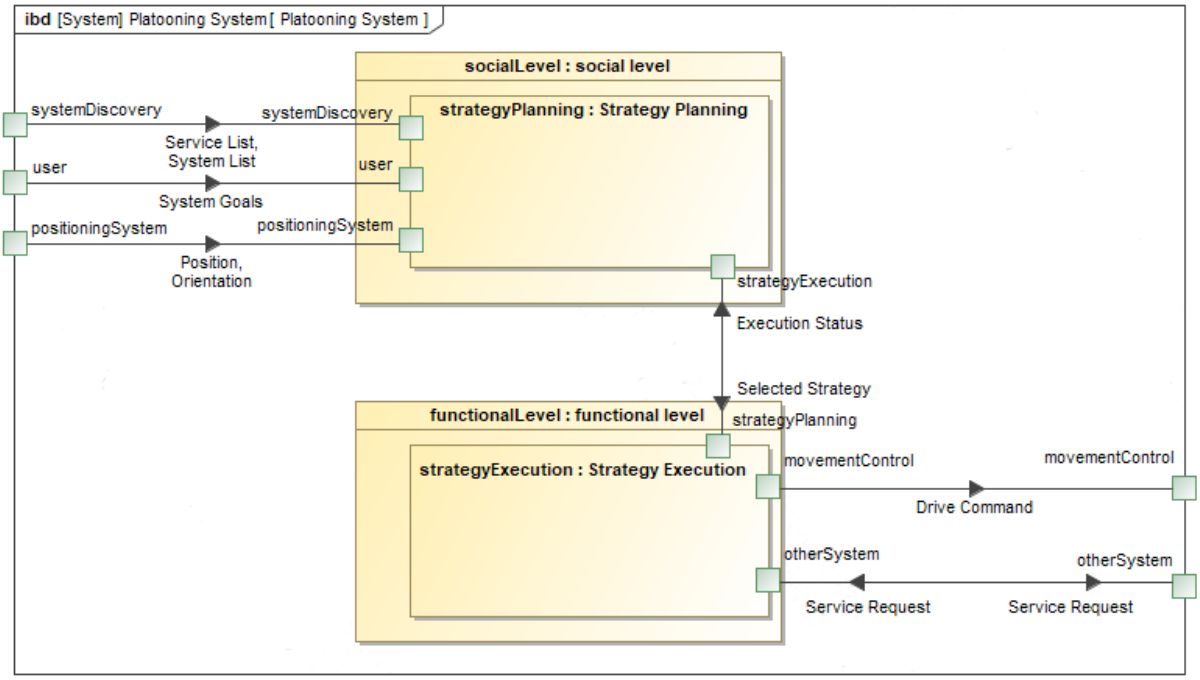


Abbildung 8: Systemarchitektur für das CeCar Platooning System

abgespeichert. Eine Strategie besteht dabei aus einem oder mehreren verschiedenen Services, die die Aktorik ansteuern oder Berechnungen durchführen. Die Strategien werden während des Planungsprozesses daraufhin überprüft, ob sie auf der Basis der aktuellen Beliefs ausführbar sind und zum Erreichen der Desires des Systems führen. Weiterhin wird anhand der Daten, die das *System Discovery* System liefert, die Verfügbarkeit aller Services und Systeme überprüft, die für die Strategie notwendig sind. Wurde erfolgreich eine Strategie evaluiert, wird diese über die Schnittstelle zum zweiten Subsystem *Strategy Execution* übergeben.

Das Subsystem *Strategy Execution* ist für die Koordinierung und Ausführung einer Strategie zuständig. Dazu gehört zum einen die Bereitstellung aller Services, die das lokale CeCar ausführen kann. Die entsprechenden Funktionen der Services werden in diesem Subsystem implementiert. Diese sind sowohl durch das eigene System als auch über eine Schnittstelle nach außen durch andere Systeme nutzbar. Zum anderen übernimmt das *Strategy Execution* System die Steuerung der Services abhängig von der ausgewählten Strategie. Das bedeutet, dass nach dem Kommunizieren einer neuen Strategie vom *Strategy Planning* System noch aktive Services der alten Strategie deaktiviert und die benötigten Services der neuen Strategie aktiviert werden. Zusätzlich überwacht dieses Subsystem den Status der einzelnen aktiven Services. So wird unter anderem zurückgemeldet, dass die Ausführung eines Service abgeschlossen wurde oder ein Fehler im Service aufgetreten ist. Als Ergebnis der Strategieausführung wird von diesem Subsystem falls notwendig auch die Aktorik des CeCars angesteuert. Dazu wird die vorhandene Schnittstelle zum *Movement Control* System verwendet.

In dem IBD ist zu sehen, dass beide Subsysteme jeweils einem Hierarchielevel zugeordnet sind. Die Einteilung der Systemarchitektur in einen *social* und einen *functional level* stammt aus einer Forschungsarbeit [35], in deren Kontext diese Masterarbeit mit dem Platooning-System einen Demonstrator für die dort entwickelten Konzepte implementiert. Dort wird für adaptive SoS, wie z.B. das Platooning-System, eine solche Hierarchie vorgeschlagen, um den Aspekt

der Kommunikation und Koordination mit anderen Systemen im SoS klar von der Ausführung der Systemfunktionen zu trennen. Im Platooning-System ist das *Strategy Planning* System für die Koordinationsaufgaben mit den anderen SoS Mitgliedern verantwortlich und kann damit eindeutig dem *social level* zugeordnet werden. Das *Strategy Execution* System ist ausschließlich für die Steuerung der Services, also der Systemfunktionen, zuständig. Damit wird es in dieser Struktur in den *functional level* eingeordnet.

Um die Einteilung der Systemfunktionen auf die Architekturelemente nachvollziehbar zu machen, wurde eine SysML Allocation Matrix für die Zuordnung der Use Cases erstellt (siehe Anhang D). Dort ist zu sehen, welche der zuvor vorgestellten Anwendungsfälle durch welches Subsystem realisiert werden. Über die Zuordnung der Use Cases ist zudem eine Verfolgbarkeit der Architekturelemente zu den Anforderungen aus Abschnitt 4.2 gegeben. Mit der Festlegung der Systemarchitektur ist der Entwurf des CeCar Platooning-Systems abgeschlossen. Im folgenden Teil der Arbeit wird die Implementierung entsprechend dieses Systementwurfs beschrieben.

5 Implementierung des CeCar Platooning-Systems

Nachdem Architektur und Funktionen des CeCar Platooning-Systems entworfen wurden, kann auf dieser Grundlage die Implementierung begonnen werden. Diese soll im folgenden Kapitel näher beschrieben werden. Dafür wird zunächst die Strukturierung der Software erläutert. Im Anschluss soll das Konzept der Strategien und deren Ausführung durch das Platooning-System vorgestellt werden.

5.1 Struktur der Software

Als Grundlage für das Verständnis des Platooning-Systems soll in diesem Abschnitt zunächst die Struktur der implementieren Software vorgestellt werden. Wie in Abschnitt 4.1.1 festgelegt, wurde das Platooning-System mit der Programmiersprache C++ realisiert. Um die wesentlichen Elemente dieses Quellcodes zu veranschaulichen, wurde daher ein UML-Klassendiagramm erstellt (siehe Abbildung 9). Darin sind alle verwendeten Klassen und ihre Beziehungen enthalten.

Die zentralen Elemente des Platooning-Systems bilden die Klassen *StrategyPlanner*, *StrategyExecutor* und *ActionServer*. Diese sind von der Klasse *Node* aus der ROS2 Client Library[2],

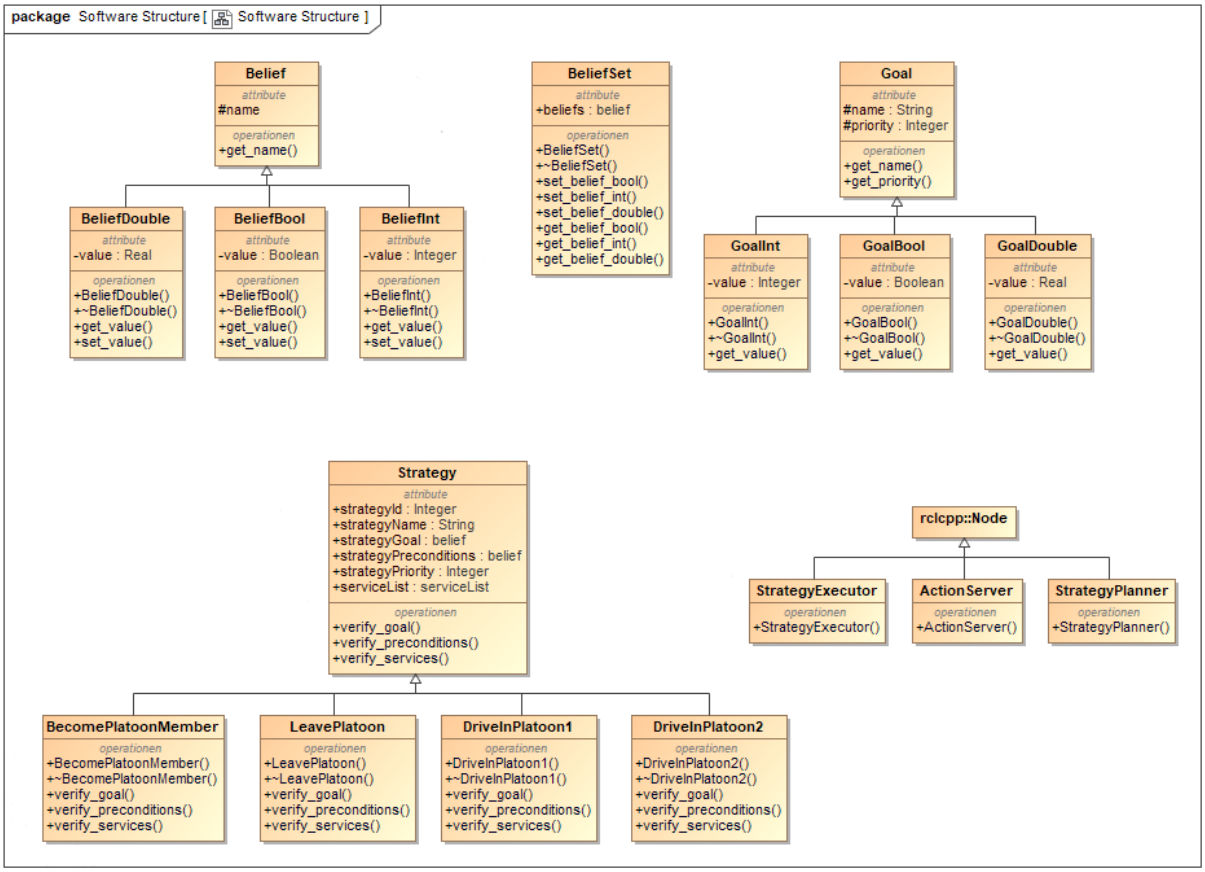


Abbildung 9: Klassendiagramm für das CeCar Platooning System

[2] Siehe https://index.ros.org/p/rclcpp/

der zentralen Software Bibliothek für das ROS2 Framework, abgeleitet. Eine solche Node wird zur Laufzeit des Programms in einem eigenen Prozess ausgeführt. Diese drei Nodes realisieren zusammen sämtliche Funktionalitäten des Platooning-Systems. Auf den Aufbau des Systems zur Laufzeit wird im nächsten Abschnitt näher eingegangen. Zum besseren Verständnis des Diagramms wurden bei diesen Klassen lediglich die als *public* deklarierten Funktionen dargestellt. Bei allen drei Klassen handelt es sich jeweils um den Konstruktor zur Erzeugung eines Objektes der Klasse. Darüber hinaus besitzen alle Nodes zahlreiche private Attribute und Funktionen zur Realisierung der Systemfunktionen. Für ein Verständnis der Softwarestruktur sind diese hier jedoch nicht relevant.

Alle weiteren Klassen repräsentieren die Objekte, mit denen das Platooning-System als BDI Agent arbeitet. Diese grundlegende Struktur wurde aus der in Abschnitt 3.2.2 vorgestellten ROS2 Implementierung[3] von Alzetta und Giorgini [22] übernommen und an die Gegebenheiten der CeCar-Plattform angepasst. Dabei wird jedes Element des BDI Agenten in einer eigenen Klasse realisiert: Beliefs durch die Klasse *Belief*, Desire durch die Klasse *Goal* und Intentions durch die Klasse *Strategy*. Im Diagramm ist zu sehen, dass sich die Klassen *Belief* und *Goal* stark ähneln. Beide speichern eine Information mit einem Namen und einem dazugehörigen Wert. Da im Platooning-System mit unterschiedlichsten Datentypen gearbeitet wird, wurden für beide Elemente abgeleitete Klassen für drei elementare Datentypen definiert. Zusätzlich erhält ein *Goal* Objekt eine Priorität, um bei der späteren Planung aus unterschiedlichen Systemzielen auswählen zu können. Im Unterschied zur Klasse *Belief* besitzt die Klasse *Goal* darüber hinaus keine *set_value* Funktion. Zur Laufzeit können dadurch die Beliefs des Systems jederzeit der aktuellen Situation entsprechend angepasst werden, ein erstelltes Systemziel bleibt jedoch unveränderbar. Zur Speicherung des gesamten Wissens des Systems wird die Klasse *BeliefSet* eingeführt. In einem Objekt dieser Klasse können beliebig viele unterschiedliche Beliefs gespeichert werden. Über die entsprechenden Member Funktionen wird der Zugriff auf vorhandene Beliefs, sowie deren Änderung zur Laufzeit vereinfacht. Mit der Klasse *Strategy* werden die unterschiedlichen Strategien, die das System ausführen kann, definiert. Dort werden wichtige Parameter, wie Name, Vorbedingungen und das Ziel einer Strategie festgelegt. Außerdem werden die Grundfunktionen zur Validierung einer Strategie hinsichtlich ihrer Anwendbarkeit in einer bestimmten Situation in den Member Funktionen der Klasse implementiert. Dadurch kann der Quellcode zur Strategieplanung kompakter und übersichtlicher gehalten werden. Für jede neue Strategie des Systems wird eine eigene Klasse von der *Strategy* Klasse abgeleitet. Das ist notwendig, um eine unterschiedliche Verifizierung von Zielen, Vorbedingungen oder Services in jeder einzelnen Strategie zu ermöglichen. In Abbildung 9 sind daher die vier bereits implementierten Strategien des Systems zu sehen. Auf die Implementierung der einzelnen Strategien wird in Abschnitt 5.3 genauer eingegangen.

Der gesamte Quellcode für das Platooning-System ist in einem ROS2 *Package* mit dem Namen *platooning* zusammengefasst. Darüber hinaus wurden alle neu eingeführten ROS2 *Message*, *Service* und *Action* Definitionen für das Platooning im *platooning_interfaces Package*

[3] Siehe https://github.com/ElDivinCodino/ROS2BDI

gesammelt. Diese Strukturierung in *Packages* gewährleistet ein hohes Maß an Wiederverwendbarkeit der Software in anderen ROS2 Systemen und Projekten.

5.2 ROS2 Struktur

In diesem Abschnitt soll die Struktur des CeCar Platooning-Systems zur Laufzeit beschrieben werden. Da die Implementierung dieses Systems auf der Basis des ROS2 Frameworks erfolgt ist, wird das Platooning-System durch die Grundkonzepte von ROS2 charakterisiert. Dazu zählen vor allem *Nodes*, *Topics* und *Services*. Eine *Node* ist in ROS2 ein zentraler Programmknoten, in dem der Quellcode in einem eigenen Prozess ausgeführt wird. Zwischen solchen *Nodes* können Informationen auf zwei Wegen ausgetauscht werden. Zum einen nutzt ROS2 *Topics*, über die Nachrichten nach dem Publish-Subscribe Prinzip ausgetauscht werden können. Zum anderen können Funktionen der *Nodes* nach dem Server-Client Prinzip in Form von *Services* ausgetauscht werden. Neben diesen Grundmechanismen werden für die Implementierung des CeCar Platooning-Systems außerdem sogenannte *Actions* verwendet, die das ROS2 Framework bereitstellt. Eine solche *Action* ist eine komplexere Form der ROS2 *Services*. Diese sind aus einer Kombination von *Topics* und *Services* aufgebaut (siehe Anhang B). Sie sind für länger andauernde Aufgaben vorgesehen und im Gegensatz zu den *Services* auch während der Ausführung unterbrechbar. Im Platooning-System werden diese *Actions* für die Realisierung der Services eines CeCars verwendet. Deshalb ist im Folgenden zwischen CeCar Services, aus denen die Strategien aufgebaut sind, und ROS2 *Services* zu unterscheiden. Die Unterscheidung wird anhand der Schriftart vorgenommen. Alle ROS2 Elemente sind in diesem Abschnitt *kursiv* geschrieben.

Für die Implementierung des Platooning-Systems muss aus dessen Systemarchitektur (siehe Abschnitt 4.4) die Struktur in der ROS2 Umgebung des CeCars abgeleitet werden. Ein zentraler Aspekt ist dabei die Aufteilung der vorgestellten Subsysteme des Platooning-Systems auf einzelne *Nodes*. Diese Aufteilung ist in Abbildung 10 dargestellt. Dort ist zu sehen, dass das Platooning-System durch drei *Nodes* gebildet wird. Das Subsystem *Strategy Planning* wird dabei durch die *Node strategy_planner* realisiert. Die Funktionalitäten des *Strategy Execution* Subsystems sind in der ROS2 Implementierung auf die zwei *Nodes strategy_executor* und *action_server* aufgeteilt. Zusammen bilden diese drei *Nodes* einen Platooning Agenten (die Repräsentation des Platooning-Systems in der ROS2 Umgebung). Diese Aufteilung wurde gewählt, um die grundlegenden Aufgaben des Platooning-Systems unabhängig voneinander ausführen zu können. In der *strategy_planner Node* findet die Strategieplanung für das System statt. Dort werden alle Informationen zum Zustand des Systems und dessen Umwelt in einem *BeliefSet* gesammelt und gespeichert. Auf dieser Basis werden bei Veränderungen die verfügbaren Strategien des Systems hinsichtlich der Systemziele und des aktuellen Wissensstandes neu bewertet und die geeignetste Strategie ausgewählt. Diese Planung soll parallel zu der Ausführung einer Strategie geschehen, um auch während der Strategieausführung flexibel auf Änderungen in der Umwelt reagieren zu können. Deshalb wird die Aufgabe der Steuerung der für eine Strategie notwendigen Services in der separaten *strategy_executor Node* realisiert. Dort werden entsprechend der ausgewählten Strategie die benötigten Services aktiviert und andere noch aktive Services beendet. Neben der Steuerung der notwendigen Services ist auch die

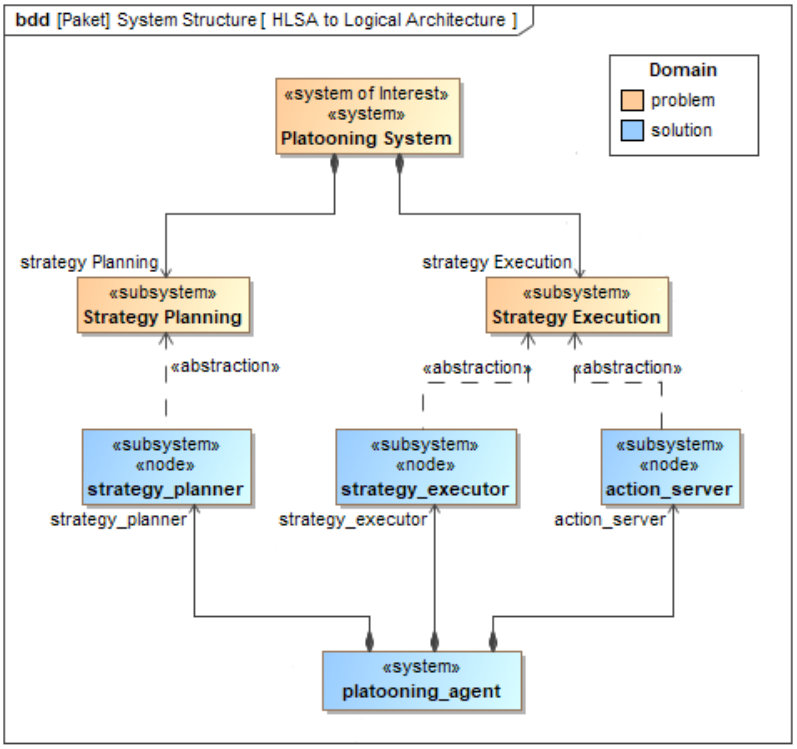


Abbildung 10: Ableitung der ROS2 Nodes aus der Systemarchitektur

Bereitstellung der Services eines CeCars Aufgabe des *Strategy Execution* Subsystems. Um die Steuerung und Ausführung der Services zu entkoppeln, werden alle Services eines Systems von der separaten *Node action_server* bereitgestellt. Diese registriert alle Services bei der *System Discovery* und führt einen Service nach einer entsprechenden Anforderung aus.

Neben der Aufteilung der *Nodes* ist auch deren Verknüpfung ein wichtiger Aspekt in der implementierten ROS2 Struktur. Zur Erfüllung der Systemfunktionen müssen die einzelnen *Nodes* miteinander kommunizieren. Da es das Hauptziel des Platooning-Systems eines CeCars ist, sich mit anderen CeCar Systemen zu einem Platoon zusammenzuschließen, ist darüber hinaus auch eine Kommunikation mit weiteren Systemen aus dem Systemkontext des Platooning-Systems notwendig. Zur Darstellung dieser Schnittstellen in der ROS2 Implementierung wurde ein SysML-IBD für einen Platooning Agenten erstellt (siehe Abbildung 11). Diese ROS2 Architektur ist aus der Systemarchitektur abgeleitet und weist daher eine ähnliche Struktur auf. Auch die *Nodes* können als Bestandteile der Subsysteme dem *social* und *functional level* zugeordnet werden. Der *action_server* ist die einzige Einheit des Systems, welche die Services eines CeCars für sich und andere Fahrzeuge bereitstellt. Damit ist auch nur diese *Node* im *functional level* verortet. Die anderen beiden *Nodes* stellen keine Funktionen in Form von Services bereit, sondern koordinieren und steuern die Nutzung dieser Services in Abstimmung mit anderen Fahrzeugen. Deshalb sind der *strategy_planner* und *strategy_executor* in den *social level* einzuordnen. In Abbildung 11 sind außerdem die Schnittstellen der *Nodes* dargestellt. Dort ist zu sehen, dass sowohl eine Kommunikation über *Topics* als auch ROS2 *Services* stattfindet. Die Schnittstellen *strategyExecution* und *serviceRegistration* sind dabei *Service* Aufrufe. Alle Verbindungen, die über einen *Topic* Block führen, werden durch ein *Topic* mit dem entsprechenden

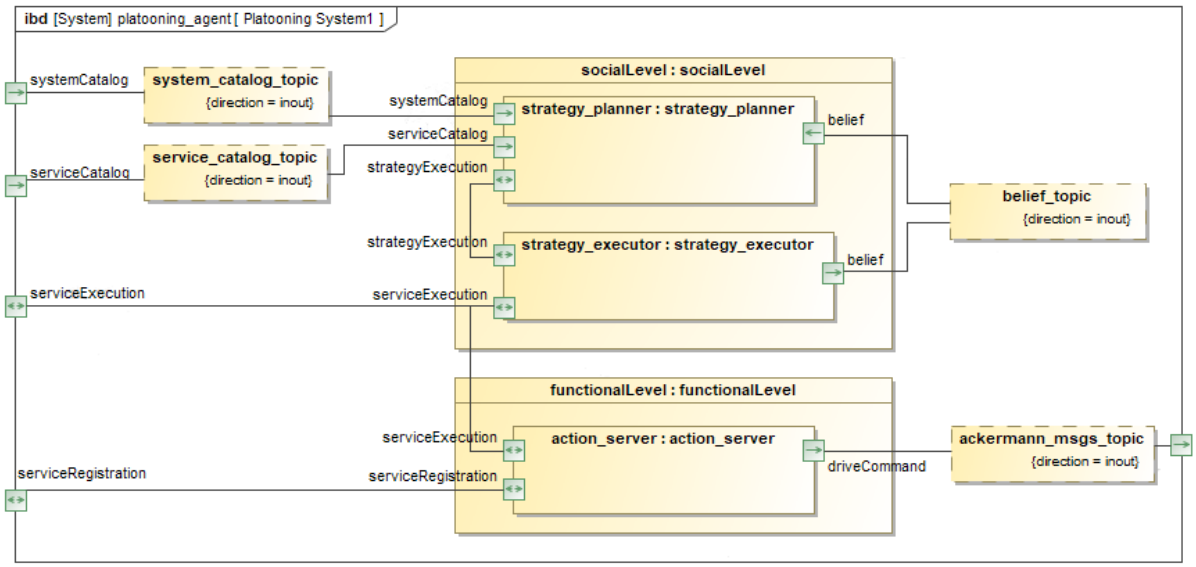


Abbildung 11: ROS2 Architektur des CeCar Platooning Systems

Namen realisiert. Über die *serviceExecution* Schnittstelle steuert der *strategy_executor* die CeCar Services. Diese Schnittstelle stellt also eine Kommunikation mittels *Actions* dar. Zusätzlich zu den hier dargestellten Schnittstellen kommunizieren die *Nodes* über weitere systeminterne *Services* für einige Hilfsfunktionen und Datenaustausch. Für die grundlegende Struktur der ROS2 Implementierung sind diese allerdings nicht relevant und wurden zur Verbesserung der Übersichtlichkeit nicht dargestellt.

5.3 Realisierung der Strategien

Nachdem die statische Struktur des CeCar Platooning-Systems beschrieben wurde, soll nun das dynamische Verhalten erläutert werden. Das Verhalten des Platooning-Systems wird im Wesentlichen durch die Ausführung von Strategien bestimmt. Im folgenden Abschnitt wird deshalb auf den Aufbau der implementierten Strategien und deren Auswahl näher eingegangen.

5.3.1 Aufbau der Strategien

Für das Verständnis der Strategien ist zunächst deren Aufbau von Bedeutung. Die Struktur der Strategien in dieser Arbeit orientiert sich an dem Konzept der Strategy Blueprints aus [35]. Dieses Konzept wurde im Rahmen des SIRESS Forschungsprojektes [36] erstellt. Das in dieser Masterarbeit beschriebene Platooning-System wurde in diesem Zusammenhang als zukünftiger Demonstrator für Systems-of-Systems implementiert. Als Grundlage für die Strategien dienen einzelne Services. Diese implementieren mögliche Handlungen des Platooning-Systems. Dabei erfüllt jeder Service eine klar abgegrenzte Funktion. Das kann beispielsweise das Auslesen von Sensordaten, die Ansteuerung der Aktorik oder eine übergeordnete Berechnung sein. Ziel der Strategy Blueprints ist es nun, eine Strategie zum Erreichen eines Systemziels aus einer Kombination dieser Services zusammenzustellen. Ein solcher Blueprint legt dabei lediglich den Typ und die geforderten Schnittstellen der zu verwendenden Services fest. Die tatsächliche Auswahl

der Services und damit auch der Systeme, die diese Services bereitstellen, erfolgt erst während der Strategieplanung.

Im Rahmen dieser Masterarbeit wird eine etwas vereinfachte Form dieser Strategy Blueprints implementiert, um den Aufwand für die Strategieplanung zu verringern. Die Services, aus denen eine Strategie aufgebaut ist, werden per Design definiert. Dadurch ist eine Strategie zur Laufzeit immer statisch aus denselben Services aufgebaut. Da laut der Anforderung SN-6 nur homogene Platoons im Rahmen dieser Arbeit gebildet werden, ist allerdings bei der Ausführung einer Strategie die Auswahl des Systems, welches einen benötigten Service zur Verfügung stellt, möglich. Stellen mehrere Systeme denselben Service zur Verfügung, kann für die Strategie ein beliebiger dieser Services aktiviert werden, sofern in der Strategie nicht explizit ein System vorgeschrieben ist. Eine Übersicht der in dieser Masterarbeit implementierten Strategien ist in Abbildung 12 zu sehen.

Die Strategien *Become_Platoon_Member* zum Formen eines Platoons und *Leave_Platoon* zum Verlassen eines Platoons bestehen als elementare Platooning Funktionen jeweils aus nur einem Service. Dieser muss auch immer durch das lokale CeCar bereitgestellt werden. Für das kollaborative Fahren im Platoon wurden zwei alternative Strategien vorgesehen (*Drive_In_Platoon1* und *Drive_In_Platoon2*). Diese implementieren nur sehr rudimentäre Fahrfunktionen für ein Platoon. Mit Hilfe dieser Strategien soll vor allem die Bewertung und Auswahl von mehreren sehr ähnlichen Strategien getestet werden. Ein solches Szenario ähnelt dem Auswahlprozess zwischen mehreren Instanzen desselben Strategy Blueprints. Außerdem wird durch die beiden Strategien die Verwendung von Services anderer CeCar Systeme erprobt. In der Strategie *Drive_In_Platoon1* wird der Service *provide_drive_control* des Fahrzeuges an der Spitze des Platoons angefordert. Durch diesen Service werden die Fahrkommandos, die ein Nutzer an das führende Fahrzeug sendet, auf einem entsprechenden *Topic* für die anderen Systeme im Platoon

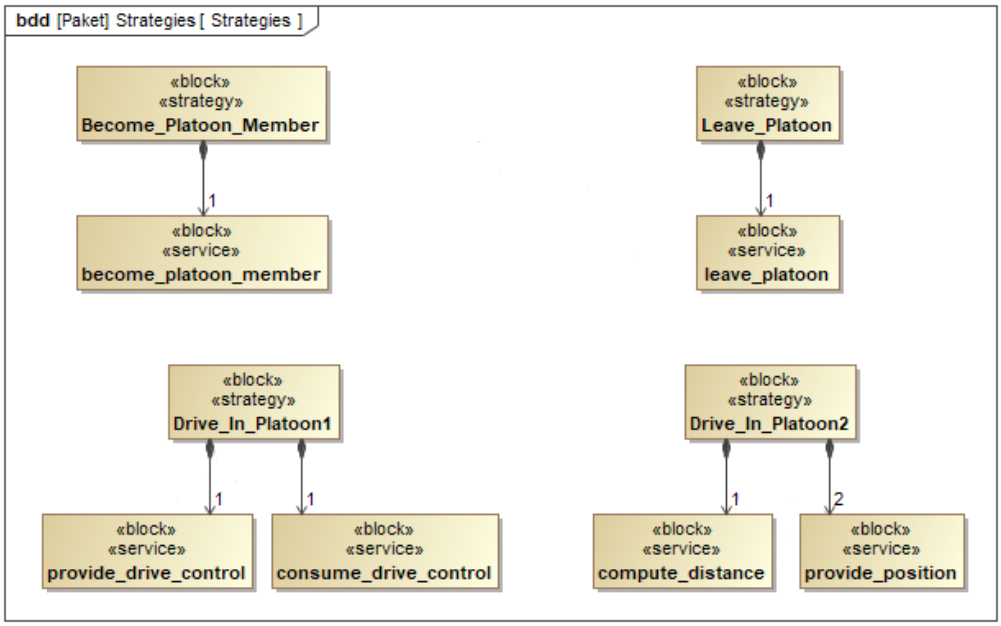


Abbildung 12: Strategien des CeCar Platooning Systems

bereitgestellt. Das CeCar, welches die Strategie ausführt, verwendet zusätzlich den lokalen Service *consume_drive_control*, welcher die Fahrkommandos anschließend an die eigene Aktorik weiterleitet, um das Verhalten des führenden Fahrzeuges zu imitieren. Die zweite Strategie für das kollaborative Fahren basiert auf einer Regelung des Abstandes zwischen zwei Fahrzeugen. Dafür verwendet ein CeCar sowohl den lokalen *provide_position* Service als auch den des vor ihm fahrenden Fahrzeuges, welche jeweils die aktuelle Position des CeCars zur Verfügung stellen. Im Service *compute_distance* wird anhand der beiden bereitgestellten Positionen der Abstand der Fahrzeuge ermittelt. Dieser bildet den Input für einen Regler, der den Abstand auf einen festgelegten Sollwert einstellt. Als Stellgröße dient dazu die Geschwindigkeit des hinteren CeCars. In beiden Strategien ist nur eine Fahrt ohne Lenkbewegungen vorgesehen.

5.3.2 Auswahl der geeignetsten Strategie

Das CeCar Platooning-System muss während der gesamten Laufzeit des Programms seine eigenen Handlungen ständig bewerten und gegebenenfalls neu planen. Nur so kann gewährleistet werden, dass das Platooning-System den Zielen und Umweltbedingungen entsprechend handelt. Eine neue Strategieplanung wird deshalb immer durchgeführt, wenn sich Systemziele ändern oder entscheidende Veränderungen im Systemzustand oder den Umweltbedingungen detektiert werden. Für diesen Planungsprozess stehen dem System als Ergebnis dieser Masterarbeit die zuvor beschriebenen vier Strategien zur Verfügung.

Eine entscheidende Rolle für die Strategieplanung spielen die Systemziele. Diese werden über ROS2 *Parameter* der *strategy_planner Node* definiert. Durch dieses Design können die Ziele auch zur Laufzeit des Programms vom Nutzer verändert werden. Für die Realisierung der Grundfunktionen des Platoonings wurden im *strategy_planner* zwei Zielparameter definiert, über die das jeweilige Systemziel aktiviert oder deaktiviert werden kann:

- *Platooning* (Ziel für Platoon Bildung),
- *Drive_In_Platoon* (Ziel für kollaboratives Fahren).

Darüber hinaus wurden drei Parameter zum Einstellen der Zielprioritäten eingeführt:

- *Leave_Platoon_Priority* (Priorität für das Verlassen eines Platoons),
- *Platooning_Priority* (Priorität für die Bildung eines Platoons),
- *Drive_In_Platoon_Priority* (Priorität für das kollaborative Fahren).

Als erster Schritt der Strategieplanung wird das Systemziel mit der höchsten Priorität bestimmt. Je kleiner der Wert des zugehörigen Parameters für die Zielpriorität dabei ist, desto höher ist die Priorität des Ziels. Die Ziele werden im Platooning-System durch *Goal* Objekte (siehe Abschnitt 5.1) repräsentiert. Diese werden abhängig von den Werten der Zielparameter erzeugt, erhalten den jeweiligen Prioritätswert und werden in einem *GoalSet* gesammelt. Dieses *GoalSet* wird nach dem *Goal* mit der höchsten Priorität durchsucht. Das entsprechende Objekt wird im weiteren Planungsprozess als das zu verfolgende Systemziel festgelegt.

Im nächsten Schritt werden die Strategien des Systems hinsichtlich ihrer Anwendbarkeit in der aktuellen Situation bewertet. Ein SysML Aktivitätsdiagramm für diesen Prozess ist in Anhang F zu sehen. Zunächst wird das Strategieziel mit dem Systemziel verglichen. Nur wenn eine

Strategie zum Erreichen des Systemziels führt, wird sie als Kandidat in Betracht gezogen. Darüber hinaus wird geprüft, ob alle Rahmenbedingungen für die Aktivierung der Strategie erfüllt sind. Diese Bedingungen sind fest für eine Strategie definiert und im entsprechenden *Strategy* Objekt gespeichert. Dazu gehören die Vorbedingungen der Strategie sowie die benötigten Services. Die *strategy_planner Node* prüft erst anhand des aktuellen *BeliefSets*, ob dort für alle Vorbedingungen, die ebenfalls als *Belief* formuliert sind, ein entsprechender Eintrag mit dem korrekten Wert vorhanden ist. Konnten auf diese Weise alle Vorbedingungen verifiziert werden, wird anschließend der *Service Catalog* vom *System Discovery* System hinsichtlich der benötigten Services für die Strategie ausgewertet. Als Ergebnis dieser Überprüfungen entsteht eine Sammlung von allen Strategien, die für eine Ausführung in Frage kommen.

Im letzten Schritt der Strategieplanung wird aus diesen Strategiekandidaten der geeignetste ausgewählt. In [35] wird für diese Auswahl der Einsatz von Evaluierungsfunktionen vorgeschlagen, mit denen bewertet wird, wie gut eine Strategie ein gegebenes Ziel erfüllt. Um die Komplexität dieses Auswahlprozesses zu reduzieren werden im Rahmen dieser Arbeit die Strategien zunächst lediglich anhand ihrer Priorität bewertet. Ein Aktivitätsdiagramm für diesen Auswahlprozess ist in Abbildung 13 dargestellt. Für das Platooning-System wird in dieser Arbeit festgelegt, dass eine Strategie umso geeigneter für die Ausführung ist, je niedriger deren Priorität ist. In der Implementierung wird eine niedrige Priorität durch einen hohen Prioritätswert ausgedrückt. Die Priorität der Strategie muss jedoch trotzdem höher als die des aktuellen

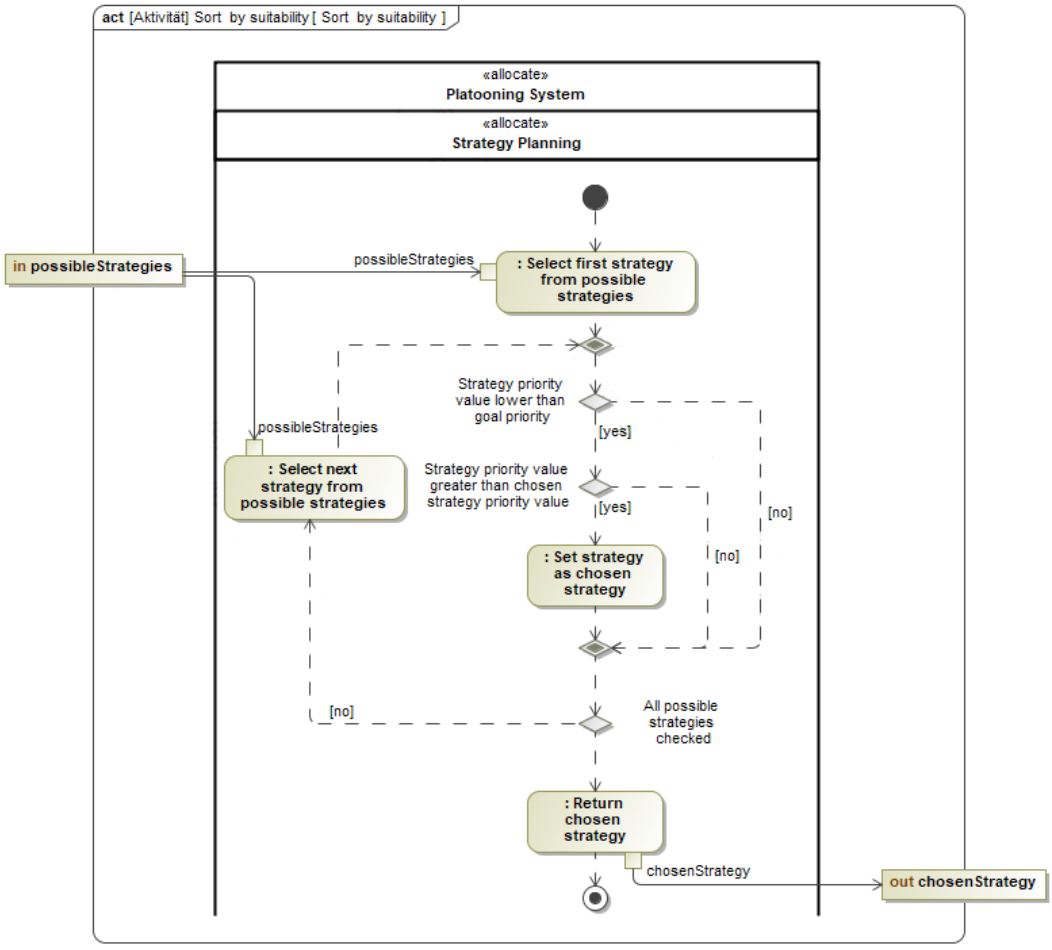


Abbildung 13: Aktivitätsdiagramm für die Auswahl der geeignetsten Strategie

Systemziels sein (also einen niedrigeren Prioritätswert aufweisen). Auf diese Weise wird die Grundlage für eine parallele Strategieausführung für mehrere Systemziele geschaffen. Demnach ist gewährleistet, dass eine gewählte Strategie mindestens mit der Priorität des Systemziels ausgeführt wird, gleichzeitig aber nicht die Ausführung einer Strategie für ein höher priorisiertes Ziel blockiert. Im Rahmen dieser Arbeit wird zwar zunächst nur der Fall eines aktiven Ziels zu einer Zeit betrachtet, mit diesem Auswahlprozess soll aber eine spätere Erweiterung leichter ermöglicht werden. Genauso ist eine Erweiterung der Strategieauswahl hinsichtlich der Bewertung der einzelnen Strategien gut möglich. Der Vergleich der Strategien kann statt auf Basis der Prioritäten mithilfe eines beliebigen anderen Wertes zur Charakterisierung der Strategien durchgeführt werden. So ist für die Evaluierung beispielsweise eine Bewertung anhand der benötigten Zeit oder der Genauigkeit der Zielerfüllung denkbar. Konnte als Ergebnis des vorgestellten Planungsprozesses keine gültige Strategie ermittelt werden, wird bis zum nächsten Trigger Event für die Strategieplanung die bisher aktive Strategie weiter ausgeführt. Nur wenn das System ein Platoon Mitglied ist, wird in diesem Fall die Strategie zum Verlassen des Platoons aktiviert.

5.3.3 Strategieausführung am Beispiel Platoon Forming

Nachdem eine Strategie in der *strategy_planner Node* ausgewählt wurde, muss diese vom Platooning-System ausgeführt werden. Das Ausführen einer Strategie bedeutet, dass alle Services, die zu dieser Strategie gehören, aktiviert werden. Wenn ein CeCar mit anderen zusammen ein Platoon gebildet hat, muss dabei koordiniert werden, welche Services von welchem System gestartet werden. Diese Koordination übernimmt die *strategy_executor Node*. Abhängig von der ausgewählten Strategie sendet diese *Node* Anfragen zur Aktivierung der notwendigen Services an die jeweiligen *action_server Nodes*, die diese Services zur Verfügung stellen. Bevor eine Strategie ausgeführt wird, muss jedoch die vorherige beendet sein. Aus diesem Grund sendet die *strategy_executor Node* zunächst eine Anfrage zum Abbrechen der *Action* an alle noch aktiven Services der vorherigen Strategie. Das beschriebene Verfahren zur Strategieausführung ist zur Veranschaulichung im SysML Sequenzdiagramm in Anhang G dargestellt.

Zum besseren Verständnis der Strategieausführung soll diese am Beispiel der Strategie *Become_Platoon_Member* näher erläutert werden. Gleichzeitig soll damit der Prozess zum Formen eines Platoons beschrieben werden. Auch dieser Ablauf kann in Form eines Sequenzdiagramms dargestellt werden (siehe Abbildung 14). Die in diesem Diagramm auf der linken Seite abgebildeten *Nodes* entsprechen dem Platooning-System eines CeCars. Auf der rechten Seite sind zwei weitere CeCar Systeme zu sehen, welche jedoch nur als gesamtes System und nicht mit den jeweiligen *Nodes* dargestellt sind.

Im Szenario des Platoon Formings hat der *strategy_planner* eines CeCars die Strategie *Become_Platoon_Member* ausgewählt und eine entsprechende Anforderung an den *strategy_executor* desselben Systems gesendet. Dieser aktiviert in der Folge den Service *become_platoon_member*, den der *action_server* des lokalen Fahrzeuges bereitstellt. Diese *Node* akzeptiert die Anforderung und startet in der Folge den Prozess zum Formen eines Platoons. Als erster Schritt in diesem Prozess wird der *strategy_planner Node* des lokalen Systems

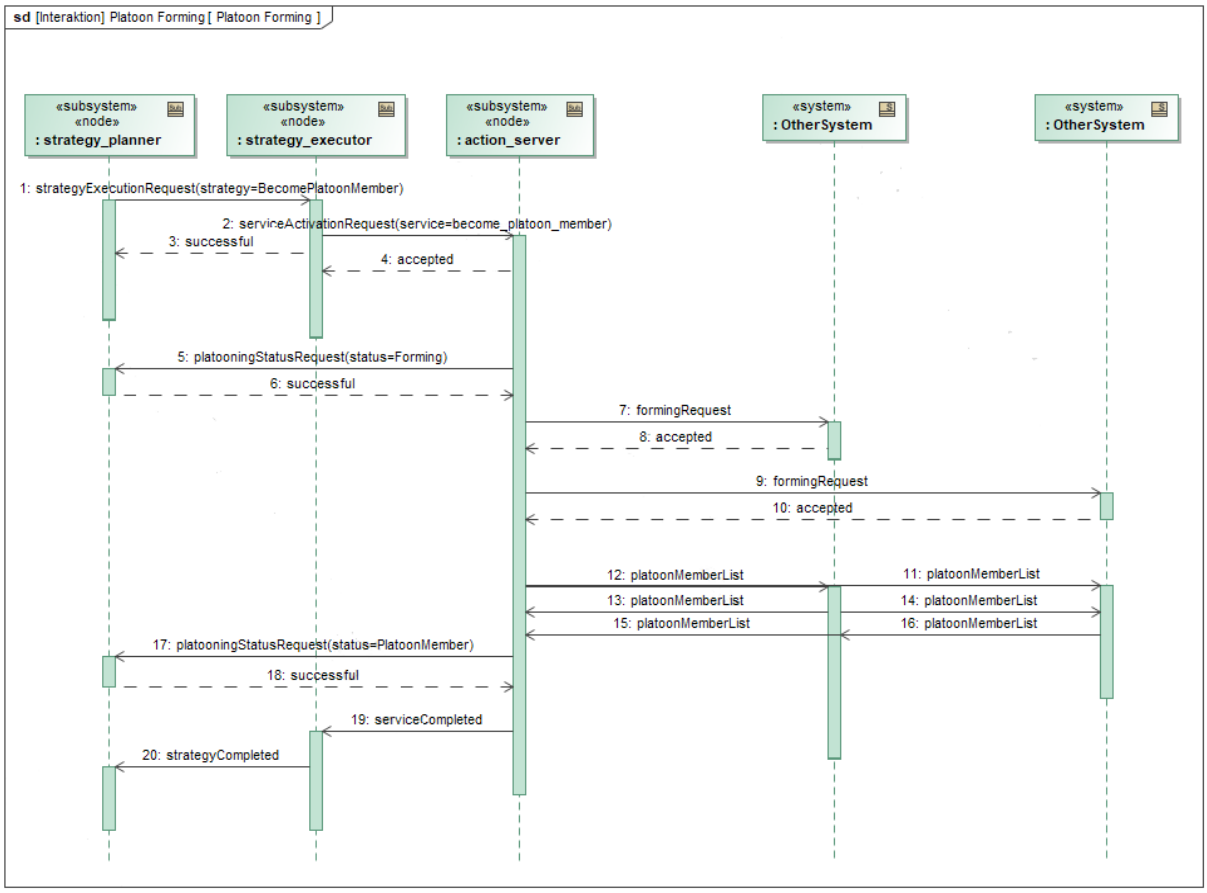


Abbildung 14: Sequenzdiagramm für das Platoon Forming

mitgeteilt, dass sich das System im Status des Formings befindet. Der *strategy_planner* verwaltet als zentrale Instanz diesen Platooning Status. Eine Änderung des Platooning Status soll deshalb nur von dieser Node vorgenommen werden. Eine Übersicht der möglichen Zustände des Platooning Status gibt das SysML Zustandsdiagramm in Anhang E. Nachdem der ROS2 *Service* des *strategy_planners* zum Setzen des Platooning Status beendet ist, initiiert der *action_server* die Platoon Bildung mit den anderen verfügbaren CeCar Systemen. Dazu wird der *System Catalog*, den das *System Discovery* System bereitstellt, durchsucht. An alle verfügbaren Systeme wird mit dem *formingRequest* die Anforderung eines entsprechenden ROS2 *Services* zur Evaluierung des Platoon Formings gesendet. Dabei bewertet jedes System abhängig vom internen Platooning Status, ob eine neue Platoon Mitgliedschaft möglich ist. Als Ergebnis wird an das initiierende System zurückgemeldet, ob das Platoon Forming akzeptiert oder abgelehnt wurde und zusätzlich an welcher Position sich das System aktuell befindet. Antwortet ein System, dass es bereits Mitglied in einem existierenden Platoon ist, wird der Forming Prozess abgebrochen und ein Platoon Joining gestartet. Das Sequenzdiagramm zu dieser Alternative zum Platoon Forming ist in Anhang H zu sehen.

Solange kein Betritt zu einem existierenden Platoon möglich ist, werden die Antworten der anderen Systeme gesammelt und es wird eine Liste mit allen Systemen zusammengestellt, die eine Platoon Bildung akzeptiert haben. Diese Liste wird als *platoonMemberList* bezeichnet und dient der Kommunikation der Zusammensetzung des Platoons. Zunächst wird die Liste anhand der mitgesendeten Positionen der Systeme sortiert, um die Reihenfolge, in der die Systeme im

Platoon angeordnet sind, zu bestimmen. Anschließend wird die sortierte *platoonMemberList* als Bestätigung für eine erfolgreiche Platoon Bildung über ein entsprechendes Topic an die anderen CeCars gesendet. Damit sind alle Mitglieder des neuen Platoons über dessen Bildung und Zusammensetzung informiert. Jedes System bestätigt in der Folge die Platoon Zusammensetzung durch ein erneutes Senden der erhaltenen Liste über das *Topic*. Alle Mitglieder des Platoons empfangen diese Bestätigungen und prüfen, ob das Verständnis der Platoon Zusammensetzung für alle Mitglieder übereinstimmt. Stellt ein System Abweichungen fest, aktualisiert es seine lokale Liste und stößt den Prozess durch Senden dieser Liste erneut an. Dieser Ablauf wird so lange wiederholt, bis alle Platoon Mitglieder dasselbe Verständnis der Platoon Zusammensetzung bestätigt haben.

Nachdem die Platoon Bildung koordiniert wurde, ändert jedes System seinen Platooning Status zu *PlatoonMember*. In der Folge agieren diese Systeme als ein Platoon. Damit ist der Service *become_platoon_member* im *action_server* abgeschlossen. Das wird an den *strategy_executor* zurückgemeldet, welcher seinerseits den erfolgreichen Abschluss der Strategie *Become_Platoon_Member* an den *strategy_planner* meldet, da die Strategie nur aus diesem einen Service besteht. Anschließend ist es Aufgabe der *strategy_planner Node* eine neue Strategie zu finden, die zu der veränderten Situation passt.

6 Test des CeCar Platooning-Systems

Nachdem das Platooning-System für die CeCar-Plattform implementiert wurde, sollen im Rahmen dieser Masterarbeit abschließend einige Tests zum Nachweis der Funktionen des Systems durchgeführt werden. Der entsprechende Testaufbau und die Durchführung werden im Folgenden beschrieben.

6.1 Testumgebung

Zunächst soll die Testumgebung für die Systemtests vorgestellt werden. Für den Testaufbau werden mindestens zwei fahrbereite CeCars benötigt. Die Fahrzeuge müssen zudem mit den für das Platooning-System notwendigen ROS2-Paketen ausgestattet sein. Während der Tests muss die Umgebung ausreichend abgesichert sein, damit die Fahrzeuge bei eventuellen Fehlfunktionen während der Fahrt nicht beschädigt werden, da die CeCar-Plattform zum aktuellen Zeitpunkt keine funktionsfähige Kollisionsvermeidung bereitstellt. Außerdem müssen Navigations- bzw. Orientierungsfunktionen für die Fahrzeuge implementiert werden, da die Systeme beispielsweise für die Platoon Bildung oder das kollaborative Fahren ihre Position und Orientierung im Raum kennen müssen. Auch diese Funktionen stellt die CeCar-Plattform aktuell nicht zur Verfügung. Um den Aufwand für die Durchführung der Tests zu verringern, werden diese daher nicht mit realen CeCars durchgeführt. Dazu wurde im Rahmen dieser Arbeit eine Simulationsumgebung für die Testdurchführung geschaffen. Diese Simulation bietet umfassende Möglichkeiten zur Gestaltung der Testumgebung und ermöglicht eine isolierte Betrachtung der implementierten Platooning Funktionen, unabhängig von den Gegebenheiten realer CeCars. Auch logistische Probleme bei der Testdurchführung werden vermieden. Die erstellte Simulationsumgebung soll nachfolgend kurz vorgestellt werden.

Als Basis für die Simulation wurde die Software Gazebo[4] (Version 9) gewählt. Diese Software wird genau wie ROS von der Open Source Robotics Foundation (OSRF) entwickelt und ist daher sehr gut auf die Nutzung in Verbindung mit ROS und seinem Nachfolger ROS2 ausgerichtet. Zudem ist sie frei erhältlich und ermöglicht eine sehr detailgenaue Simulation von verschiedensten Robotern. Für die Simulation wurde als erstes ein stark vereinfachtes Modell des CeCars in Gazebo erstellt. Dabei wurde vor allem auf eine möglichst genaue Nachbildung der Antriebsart und Lenkung des CeCars Wert gelegt. Das Modell besteht aus einem quaderförmigen Chassis, das mit vier Rädern verbunden ist. Wie auch das CeCar besitzt dieses einfache Modell einen Allradantrieb und Vorderachslenkung. Auf Basis dieses Modells können dann beliebig viele CeCars in der Simulationsumgebung instanziiert werden (siehe Abbildung 15). Für die Tests wird darüber hinaus eine leere Welt ohne Gegenstände oder Hindernisse, wie sie in der Abbildung zu sehen ist, verwendet.

[4] Siehe http://gazebosim.org/

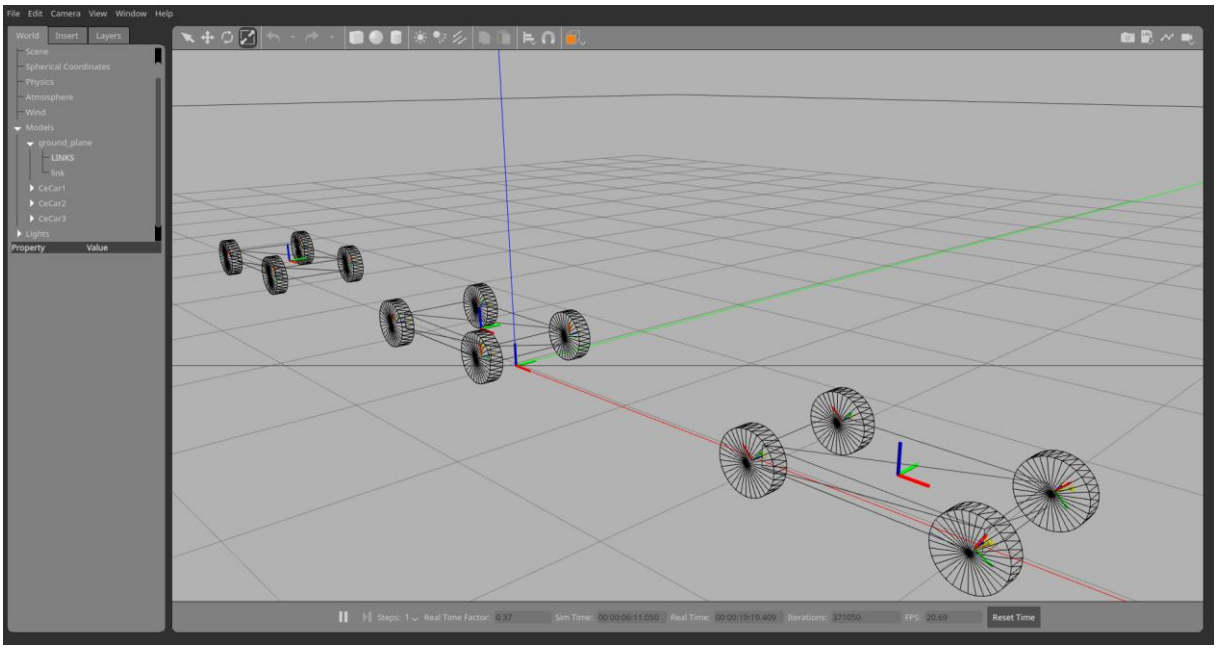


Abbildung 15: Simulationsumgebung für die CeCar-Plattform

Um die Bewegungen des CeCars zu simulieren, müssen die Gelenke, mit denen Räder und Chassis verbunden sind, entsprechend in ihrer Bewegung gesteuert werden. Diese Steuerung ist direkt aus einem ROS2 System mithilfe eines Gazebo Plugins möglich. Um die Nutzung der CeCar Schnittstellen, sowie eine Nachbildung als autonomes Fahrzeug zu gewährleisten, wurde das Plugin aus [37] modifiziert. Dieses Plugin wurde für die Steuerung von manuell gelenkten Fahrzeugen entwickelt und benötigt ein Lenkrad in dem zu steuernden Modell. Im modifizierten Plugin ist kein Lenkrad mehr notwendig und die Schnittstelle zur Steuerung des Modells wurde entsprechend der Schnittstelle in einem realen CeCar auf die Nutzung von *ackermann_msgs* umgestellt. Auf diese Weise können die simulierten Fahrzeuge über Vorgaben von Geschwindigkeit, Beschleunigung, Ruck, Lenkwinkel und Lenkwinkelgeschwindigkeit gesteuert werden. Zur Laufzeit wird das Plugin in ROS2 durch eine *Node* (*ackermann_drive*) repräsentiert, welche empfangene Fahrkommandos in entsprechende Bewegungsvorgaben für die Gelenke des Gazebo Modells umwandelt. Die Kommunikation von Fahrkommandos erfolgt über das *Topic cmd_vel*. Zur Realisierung des Allradantriebs werden alle vier Räder des Modells gleichermaßen um ihre horizontale Achse rotiert. Für die Lenkung werden die beiden Vorderräder entsprechend des Modells der Ackermann Lenkung [38] um ihre vertikale Achse rotiert. Das Einstellen der jeweils vorgegebenen Werte für die Fahrparameter wird mit Hilfe von PID-Reglern für jedes einzelne Gelenk gewährleistet. Das Gazebo Plugin übernimmt weiterhin die Übermittlung von Odometrie Daten aus der Simulation in das ROS2 System. Die Plugin *Node* sendet über entsprechende Topics sowohl aktuelle Fahrparameter (Geschwindigkeit, Lenkwinkel, Beschleunigung, etc.) als auch die zurückgelegte Distanz des simulierten Fahrzeugs.

Um das Testen von Fahrfunktionen im Rahmen des Platoonings zu vereinfachen, wurde außerdem eine *Node* (*teleop_car*) programmiert, die eine Steuerung eines CeCar Modells über Tastatureingaben ermöglicht. Ohne diese *Node* müsste zur Steuerung eines Fahrzeugs für jede Änderung der Fahrparameter ein neues Fahrkommando über die Konsole an das Fahrzeug gesendet

werden. Eine Steuerung über die Tastatur vereinfacht den Vorgang erheblich und ist für den Nutzer deutlich intuitiver. Die resultierende Struktur der Testumgebung zur Laufzeit ist in Anhang K zu sehen.

6.2 Beispielfahrt

Das CeCar Platooning-System soll anhand einer Beispielfahrt getestet werden. Dabei sollen verschiedene Szenarien des Platoonings mit dem implementierten System durchlaufen werden, um die korrekte Funktionsweise der Implementierung nachzuweisen. Zu diesen Szenarien gehören die Grundfunktionen des Platoonings, wie die Platoon Bildung, das kollaborative Fahren und das Verlassen eines Platoons. Außerdem soll auch die Funktionsweise des Systems als Agent überprüft werden, der auf Basis von verschiedenen Strategien der Situation angepasste Handlungen ausführt. Um eine Reproduzierbarkeit dieses Tests zu gewährleisten, wurde die Beispielfahrt anhand verschiedener Testfälle (TC, engl.: Test Case) definiert. Jeder Testfall deckt dabei einen oder mehrere der in Abschnitt 4.3 definierten Anwendungsfälle für das Platooning-System ab, sodass in der Gesamtheit die Erfüllung jedes Use Case durch mindestens einen Testfall überprüft wird. Die einzelnen Testfälle sind in der Tabelle in Anhang J aufgeführt. Alle Testfälle wurden nacheinander entsprechend des beschriebenen Vorgehens ausgeführt und anschließend anhand des Verhaltens in der Simulation und den Log-Daten überprüft.

Die Testfälle TC-1 bis TC-3 und TC-6 bis TC-8 werden vor allem über Konsolenausgaben der einzelnen *Nodes* des Platooning-Systems ausgewertet. Diese Log-Daten ermöglichen eine Nachvollziehbarkeit von internen Statusänderungen und der Kommunikation zwischen den *Nodes*. So ist nicht nur die äußere Erfüllung der Anwendungsfälle bewertbar, sondern auch eine Analyse der im SysML-Modell definierten Vorgaben für die Implementierung möglich. Für die Grundfunktionen des Platooning (Forming, Joining und Leaving) kann eine solche Analyse auf Basis der erstellten Sequenzdiagramme durchgeführt werden. Damit werden die Kommunikationsmechanismen überprüft, die für die Koordinierung eines Platoons festgelegt wurden. Ein Beispiel für eine solche Analyse ist in Abbildung 16 zu sehen. In den dort aufgeführten Konsolenausgaben wurden zum Nachweis die entsprechenden Schritte aus dem Sequenzdiagramm für das Forming (siehe Abbildung 14 in Abschnitt 5.3.3) farblich markiert und mit den entsprechenden Nummern versehen. Dabei sind nahezu alle gesendeten Nachrichten über die Konsolenausgabe sichtbar. Da alle *Nodes* des Systems parallel ausgeführt werden, kann die Reihenfolge in der Konsolenausgabe von der Darstellung im Sequenzdiagramm etwas abweichen. Grund dafür sind die unterschiedlichen Signallaufzeiten und der Umstand, dass alle drei *Nodes* nur abwechselnd Zugriff auf die Konsolenausgabe haben können. Ein solcher Abgleich wurde auch für das Joining (siehe Anhang L) und Leaving (siehe Anhang M) anhand der jeweiligen Sequenzdiagramme durchgeführt (siehe Anhang H und Anhang I). Für TC-1 und TC-8 ist kein Vergleich mit einem Sequenzdiagramm möglich. Deshalb wurden in den entsprechenden Konsolenausgaben (siehe Anhang N und Anhang O) die Schritte aus der Beschreibung des zu erwartenden Ergebnisses des Testfalls markiert. Die Testfälle TC-3 und TC-4 überprüfen die Implementierung des kollaborativen Fahrens. Dabei wird jeweils eine der beiden verfügbaren

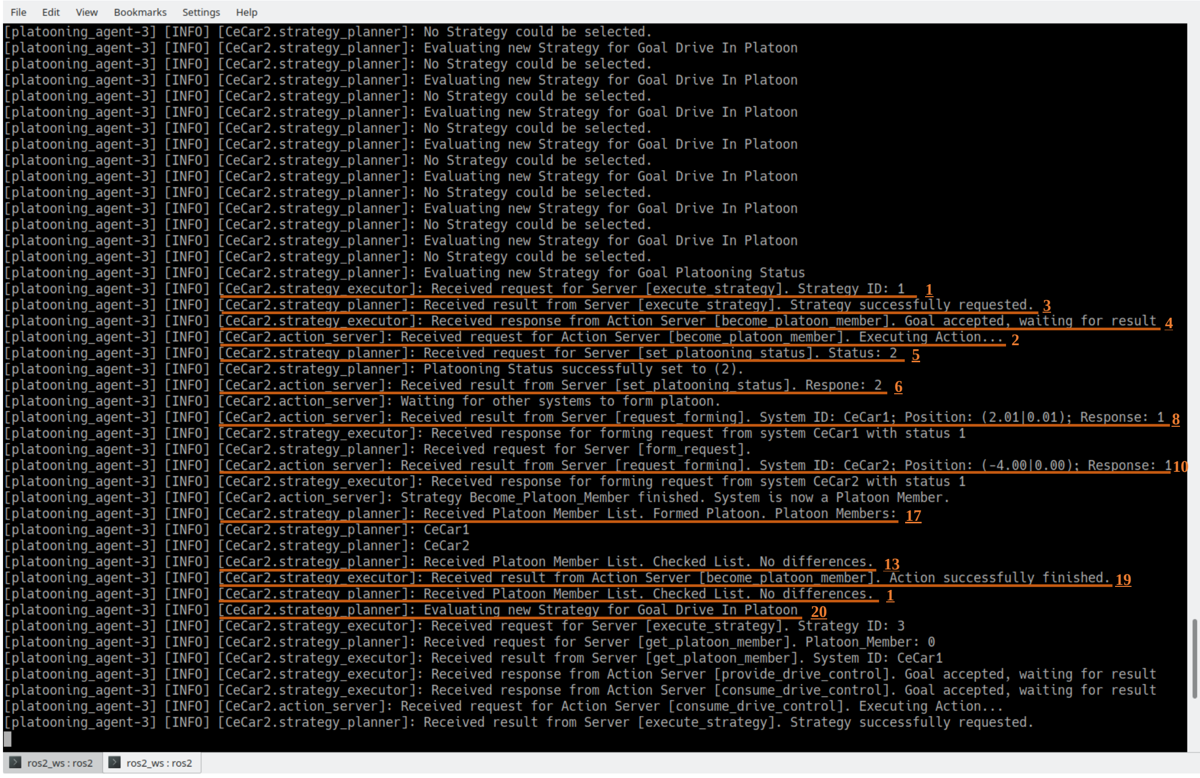


Abbildung 16: Konsolenausgabe für das Platoon Forming

Strategien getestet. Zur Auswertung der beiden Testfälle wird das Verhalten der simulierten Fahrzeuge beobachtet. In der Simulation lässt sich leicht erkennen, ob alle Fahrzeuge gemeinsam fahren und dabei einen definierten Abstand zueinander einhalten.

Das Ergebnis für jeden der definierten Testfälle ist ebenfalls in der Tabelle in Anhang J aufgeführt. Es ist zu erkennen, dass alle Testfälle erfolgreich vom Platooning-System absolviert wurden. Eine Messung von Leistungsparametern des Systems wurde nicht durchgeführt, da im Rahmen dieser Arbeit lediglich Grundfunktionen eines Platooning-Systems implementiert werden sollten. Bei der Implementierung wurde dabei kein Wert auf eine möglichst effiziente Kommunikation zwischen den Fahrzeugen oder eine optimale Ressourcennutzung gelegt. Insgesamt konnte festgestellt werden, dass das System die geforderten Funktionen korrekt umsetzt und ein Platooning mit mehreren CeCars ermöglicht.

7 Fazit

Ziel dieser Masterarbeit war der Entwurf und die Implementierung einer Platooning Funktion für die CeCar-Plattform. Dazu wurde nach einer Analyse des aktuellen Forschungsstandes eine Architektur für ein solches Platooning-System entwickelt. Für den Systementwurf wurden die Gegebenheiten der CeCar-Plattform analysiert und berücksichtigt. Der gewählte Systems-Engineering Ansatz ermöglicht eine Nachvollziehbarkeit der einzelnen Systemelemente zu den definierten Anforderungen. Entsprechend dieses Systemdesigns wurde anschließend ein Platooning-System für das CeCar implementiert. Als Ergebnis der Implementierung ist ein Softwaremodul entstanden, welches in Form eines ROS2-Packages für alle Fahrzeuge der CeCar-Plattform eingesetzt werden kann. Alle Funktionen des implementierten Systems wurden zum Nachweis der Funktionalität in einer Simulationsumgebung getestet.

Ein Platoon stellt als Gruppe von gemeinsam agierenden Fahrzeugen ein klassisches Beispiel für ein System-of-Systems dar. Beim Entwurf des Platooning-Systems für das CeCar wurde daher besonders auf eine Realisierung von SoS Merkmalen geachtet. Die gewählte Struktur als Multiagenten-System stellt eine gute Basis für die SoS Charakteristik des Platoonings dar. In dieser Struktur wird jedes Fahrzeug des Platoons durch einen BDI-Agenten repräsentiert. Jeder dieser Agenten verfügt über seine eigene Wissensbasis und führt auf dieser Grundlage eine eigenständige Handlungsplanung durch. Damit ist die Unabhängigkeit der einzelnen Fahrzeuge voneinander gewährleistet. Der Zusammenschluss zu einem Platoon erfolgt auf Basis der Ziele der jeweiligen Agenten. Die entsprechenden Schnittstellen zur Kommunikation der Einzelsysteme wurden in der Systemarchitektur festgelegt und gewährleisten eine Kooperation der Fahrzeuge.

Der Systementwurf wurde mithilfe eines SysML-Modells dokumentiert. Das ermöglicht zum einen eine anschauliche Darstellung der Systemelemente und -funktionen. Zum anderen sind auf diese Weise die unterschiedlichen Designschritte und deren Zusammenhänge nachvollziehbar. Das Modell dient daher nicht nur der Dokumentation, sondern soll eine zukünftige Erweiterung des Platooning-Systems und seiner Funktionen erleichtern. Zu dieser Erweiterbarkeit trägt außerdem die gewählte Agentenarchitektur bei. In dieser BDI-Struktur können einzelne Elemente wie Strategien oder Systemziele unkompliziert ergänzt werden.

Bei der Implementierung des CeCar Platooning-Systems wurden alle wichtigen Grundfunktionen für das Platooning realisiert. Sowohl das Bilden und Auflösen eines Platoons als auch das Beitreten und Verlassen des Platoons von einzelnen Systemen wurden in entsprechenden Strategien umgesetzt. Die Strukturierung der Software in drei einzelne ROS2 Nodes gewährleistet eine Trennung von Handlungsplanung und -ausführung. Das ermöglicht eine kontinuierliche Neubewertung der geplanten Handlungen im Hinblick auf die aktuelle Situation parallel zur Ausführung der Strategien. Damit ist eine gute Anpassungsfähigkeit des Platooning-Systems auf seine Umwelt gegeben. Durch die Verwendung von ROS2 Actions für die Realisierung der Services, aus denen die Strategien bestehen, kann zu jeder Zeit eine aktive Strategie unterbrochen werden, um eine neu ausgewählte Strategie auszuführen. Die Beeinflussung der

Systemziele über Parameter der ROS2 Nodes stellt darüber hinaus eine einfache Möglichkeit für den Nutzer dar, die Systeme zu steuern.

Mit der Simulation in Gazebo wurde im Rahmen dieser Arbeit eine Möglichkeit geschaffen die Funktionen des Platooning-Systems ohne großen Aufwand zu testen. Damit wurde sowohl die logistische Herausforderung der Bereitstellung der notwendigen Hardware von mehreren CeCars sowie das Risiko einer Beschädigung dieser Hardware vermieden. Diese Simulationsumgebung könnte auch für zukünftige Tests anderer Funktionen der CeCar-Plattform verwendet werden und somit ein schnelleres und effizienteres Testen ermöglichen. Anhand der durchgeführten Tests konnten in der Simulation alle geforderten Funktionen für das CeCar Platooning-System erfolgreich validiert werden. Mit diesen Tests wurde somit die Funktionsfähigkeit des Systems nachgewiesen. In der Zukunft sollten diese Tests aber auch mit realen CeCars durchgeführt werden, um auch das Verhalten des Systems in einer realen Umgebung beurteilen zu können.

Mit dem Platooning-System wurde in dieser Arbeit eine Grundlage für weitere Projekte und zukünftige Forschung im Bereich der Systems-of-Systems und des Platoonings auf der Basis der CeCar-Plattform geschaffen. Denkbar ist dabei beispielsweise eine Erweiterung der Platooning Funktionen. Die Strategien für das kollaborative Fahren implementieren sehr rudimentäre Fahrfunktionen. Hier können in zukünftigen Untersuchungen zum Beispiel Funktionen für das Fahren mit gelenkten Fahrzeugen oder Maßnahmen zur Gewährleistung der Sicherheit im Platoon entwickelt werden. Auch eine Erweiterung der SoS-Charakteristik hinsichtlich des Austausches und der Abstimmung von Zielen zwischen einzelnen Fahrzeugen des Platoons ist denkbar. Darüber hinaus ermöglicht das Platooning-System auf der Grundlage des Konzepts der Strategien eine Erforschung der Rekonfiguration innerhalb eines Systems-of-Systems. Hier ist eine Erweiterung auf die Nutzung von Blueprints für die Strategieplanung denkbar. Damit ließe sich auch das Verhalten des SoS im Hinblick auf fehlerbedingte Ausfälle von Services oder Systemen untersuchen.

Insgesamt wurde mit der Realisierung des CeCar Platooning-Systems das eingangs der Arbeit formulierte Ziel erreicht. Es konnte ein Systementwurf auf der Basis vorhandener Platooning Konzepte entwickelt werden. Die grundlegenden Funktionen für das Platooning wurden anhand dieses Systemdesigns implementiert und erfolgreich validiert.

Literaturverzeichnis

[1] Bundesanstalt für Straßenwesen (2019) *Fahrleistung aller Kraftfahrzeuge in Deutschland in den Jahren 1970 bis 2018 (in Milliarden Kilometern)* [online]. Statista GmbH. https://de.statista.com/statistik/daten/studie/2988/umfrage/entwicklung-der-fahrleistung-von-kfz/ [Zugriff am: 20. Jul. 2020].

[2] Statistisches Bundesamt (2020) *Anzahl der Unfälle im deutschen Straßenverkehr in den Jahren 2012 bis 2019* [online]. Statista GmbH. https://de.statista.com/statistik/daten/studie/73424/umfrage/unfaelle-im-strassenverkehr/ [Zugriff am: 20. Jul. 2020].

[3] BDL (2020) *Anteil der Verkehrsträger an den weiltweiten CO2-Emissionen aus der Verbrennung fossiler Brennstoffe im Jahr 2016* [online]. Statista GmbH. https://de.statista.com/statistik/daten/studie/317683/umfrage/verkehrsttraeger-anteil-co2-emissionen-fossile-brennstoffe/ [Zugriff am: 20. Jul. 2020].

[4] Hallé, S. (2003) *Architectures for Collaborative Driving Vehicles – From a Review to a Proposal*. Sainte-Foy, Quebec: Dép. d'informatique et de génie logiciel, Université Laval.

[5] Bergenhem, C.; Shladover, S.; Coelingh, E.; Englund, C.; Tsugawa, S. (2012) *Overview of platooning systems*. Chalmers University of Technology.

[6] Carbaugh, J.; Godbole, D. N.; Sengupta, R. (1998) *Safety and capacity analysis of automated and manual highway systems* in: Transportation Research Part C: Emerging Technologies 6, 1-2, S. 69–99. https://doi.org/10.1016/S0968-090X(98)00009-6

[7] Shladover, S. E. (2007) *PATH at 20—History and Major Milestones* in: IEEE Transactions on Intelligent Transportation Systems 8, H. 4, S. 584–592. https://doi.org/10.1109/TITS.2007.903052

[8] Kalbitz, T. (2017) *A comparison of approaches for platooning management* [Masterarbeit]. Universität Mannheim. https://madoc.bib.uni-mannheim.de/42305/1/Tanja%20Kalbitz_A%20Comparison%20of%20Approaches%20for%20Platooning%20Management.pdf [Zugriff am: 16. Apr. 2020].

[9] Maier, M. W. (1998) *Architecting principles for systems-of-systems* in: Systems Engineering 1, H. 4, S. 267–284. https://doi.org/10.1002/(SICI)1520-6858(1998)1:4<267::AID-SYS3>3.0.CO;2-D

[10] Horowitz, R.; Varaiya, P. (2000) *Control design of an automated highway system* in: Proceedings of the IEEE 88, H. 7, S. 913–925. https://doi.org/10.1109/5.871301

[11] Santini, S.; Salvi, A.; Valente, A. S.; Pescape, A.; Segata, M.; Lo Cigno, R. (2015) *A consensus-based approach for platooning with inter-vehicular communications* in:

IEEE INFOCOM 2015 - IEEE Conference on Computer Communications. Kowloon, Hong Kong. IEEE, S. 1158–1166.

[12] Ackoff, R. L. (1971) *Towards a System of Systems Concepts* in: Management Science 17, H. 11, S. 661–671. https://doi.org/10.1287/mnsc.17.11.661

[13] Sage, A. P.; Cuppan, C. D. (2001) *On the systems engineering and management of systems of systems and federations of systems* in: Information knowledge systems management 2, H. 4, S. 325–345.

[14] Boardman, J.; Sauser, B. (2006) *System of Systems - the meaning of of* in: *2006 IEEE/SMC International Conference on System of Systems Engineering*. Los Angeles, California, USA. IEEE, S. 118–123.

[15] Tsugawa, S.; Kato, S.; Matsui, T.; Naganawa, H.; Fujii, H. (1-3 Oct. 2000) *An architecture for cooperative driving of automated vehicles* in: *2000 IEEE Intelligent Transportation Systems. Proceedings*. Dearborn, MI, USA. IEEE, S. 422–427.

[16] Georgeff, M. P.; Rao, A. S. (1992) *An abstract architecture for rational agents* in: Nebel, B.; Rich, C.; Swartout, W. [Hrsg.] *Principles of knowledge representation and reasoning: Proceedings of the third international conference (KR '92)*. San Mateo, Calif.: Morgan Kaufmann, S. 439–449.

[17] Bonasso, P. R.; Firby, J. R.; Gat, E.; Kortenkamp, D.; Miller, D. P.; Slack, M. G. (1997) *Experiences with an architecture for intelligent, reactive agents* in: Journal of Experimental & Theoretical Artificial Intelligence 9, 2-3, S. 237–256. https://doi.org/10.1080/095281397147103

[18] Lyons, D. M.; Hendriks, A. J. (1992) *A Practical Approach to Integrating Reaction and Deliberation* in: *Artificial Intelligence Planning Systems*. Elsevier, S. 153–162.

[19] Ferguson, I. A. (1992) *TouringMachines: an architecture for dynamic, rational, mobile agents*. UCAM-CL-TR-273. https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-273.pdf.

[20] Lygeros, J.; Godbole, D. N.; Sastry, S. (1996) *Multiagent hybrid system design using game theory and optimal control* in: *35th IEEE Conference on Decision and Control*. Kobe, Japan. IEEE, S. 1190–1195.

[21] Sun, R.; Peterson, T. (1997) *A hybrid model for learning sequential navigation* in: *1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*. Monterey, CA, USA. IEEE Comput. Soc. Press, S. 234–239.

[22] Alzetta, F.; Giorgini, P. (2019) *Towards a Real-Time BDI Model for ROS 2* in: *20th Workshop "From Objects to Agents" (WOA 2019)*. Parma, Italy. CEUR, S. 1–7.

[23] Burmeister, B.; Sundermeyer, K. (1992) *Cooperative problem-solving guided by intentions and perception (abstract)* in: ACM SIGOIS Bulletin 13, H. 3, S. 10. https://doi.org/10.1145/152683.152690

[24] Jennings, N. R. (1992) *Towards a Cooperation Knowledge Level for Collaborative Problem Solving* in: Neumann, B. [Hrsg.] *Proceedings / ECAI 92, 10th European Conference on Artificial Intelligence: August 3 - 7, 1992, Vienna, Austria*. Chichester: Wiley, S. 224–228.

[25] Pynadath, D. V.; Tambe, M. (2002) *The Communicative Multiagent Team Decision Problem: Analyzing Teamwork Theories and Models* in: Journal of Artificial Intelligence Research 16, S. 389–423. https://doi.org/10.1613/jair.1024

[26] Chaib-Draa, B.; Levesque, P. (1996) *Hierarchical model and communication by signs, signals, and symbols in multi-agent environments* in: Perram, J. W.; Müller, J.-P. [Hrsg.] *Distributed software agents and applications: 6th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW '94, Odense, Denmark, August 3 - 5, 1994 ; proceedings*. Berlin: Springer, S. 150–165.

[27] Müller, J. P.; Pischel, M. (1994) *An Architecture for Dynamically Interacting Agents* in: International Journal of Cooperative Information Systems 03, H. 01, S. 25–45. https://doi.org/10.1142/S021821579400003X

[28] Lima, P. U.; Saridis, G. N. (1999) *Intelligent controllers as hierarchical stochastic automata* in: IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society 29, Nr. 2, pp. 151–163. https://doi.org/10.1109/3477.752790

[29] Simmons, R. G. (1994) *Structured control for autonomous robots* in: IEEE Transactions on Robotics and Automation 10, H. 1, S. 34–43.

[30] Rosenblatt, J. K. (1997) *DAMN: a distributed architecture for mobile navigation* in: Journal of Experimental & Theoretical Artificial Intelligence 9, 2-3, S. 339–360. https://doi.org/10.1080/095281397147167

[31] Hochstadter, A.; Cremer, M. (1996) *Development and comparison of two different strategies for building a platoon of vehicles* in: *Vehicle Navigation and Information Systems Conference, 1996*. Orlando, FL, USA. IEEE, S. 232–240.

[32] Sukthankar, R.; Baluja, S.; Hancock, J. (1998) *Multiple Adaptive Agents for Tactical Driving* in: Applied Intelligence 9, H. 1, S. 7–23. https://doi.org/10.1023/A:1008243013521

[33] Klemm, F. (2018) *Aufbau und Evaluierung einer autonomen Fahrzeugplattform* [Masterarbeit]. Hochschule für Technik und Wirtschaft Berlin.

[34] Siefke, L.; Sommer, V.; Wudka, B.; Thomas, C. (2020) *Robotic Systems of Systems Based on a Decentralized Service-Oriented Architecture* in: Robotics 9, H. 4, S. 78. https://doi.org/10.3390/robotics9040078

[35] Wudka, B.; Thomas, C.; Siefke, L.; Sommer, V. (2020) *A Reconfiguration Approach for Open Adaptive Systems-of-Systems* in: Proceedings of the 2nd International Workshop on Governing Adaptive and Unplanned Systems of Systems.

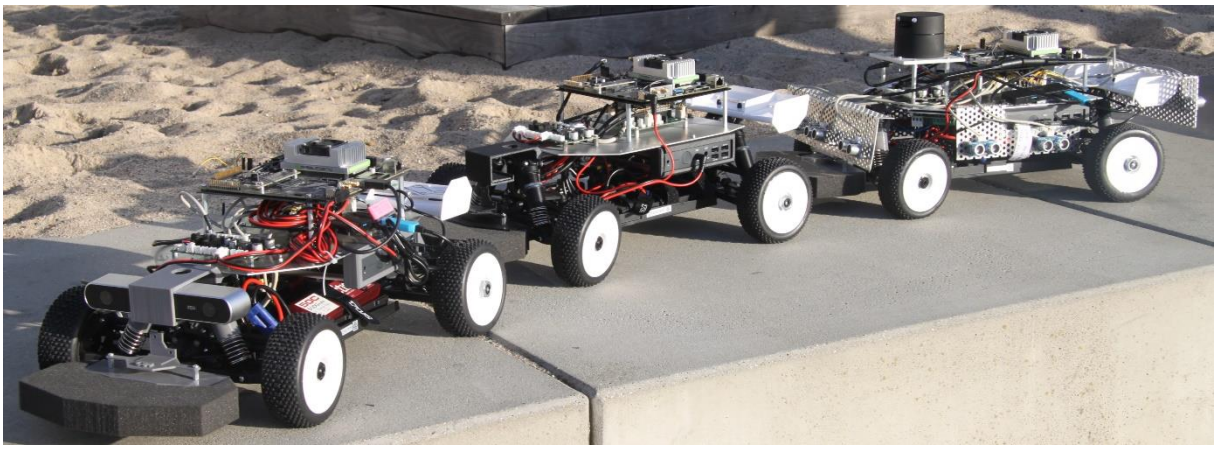
[36] Thomas, C.; Sommer, V. (2019) *Sicherheitsrelevante Rekonfigurierende Systems of Systems* [online]. Institut für angewandte Forschung Berlin. https://www.ifaf-berlin.de/uploads/media/ifaf_siress_expose.pdf [Zugriff am: 6. Nov. 2020].

[37] Shivesh Khaitan (2019) *Gazebo ROS Ackermann Drive Plugin* [online]. Open Source Robotics Foundation. https://github.com/shiveshkhaitan/gazebo_ros_pkgs/blob/b3426432d64322f57c6b233a5dd6741c1ccb71f0/gazebo_plugins/src/gazebo_ros_ackermann_drive.cpp [Zugriff am: 10. Nov. 2019].

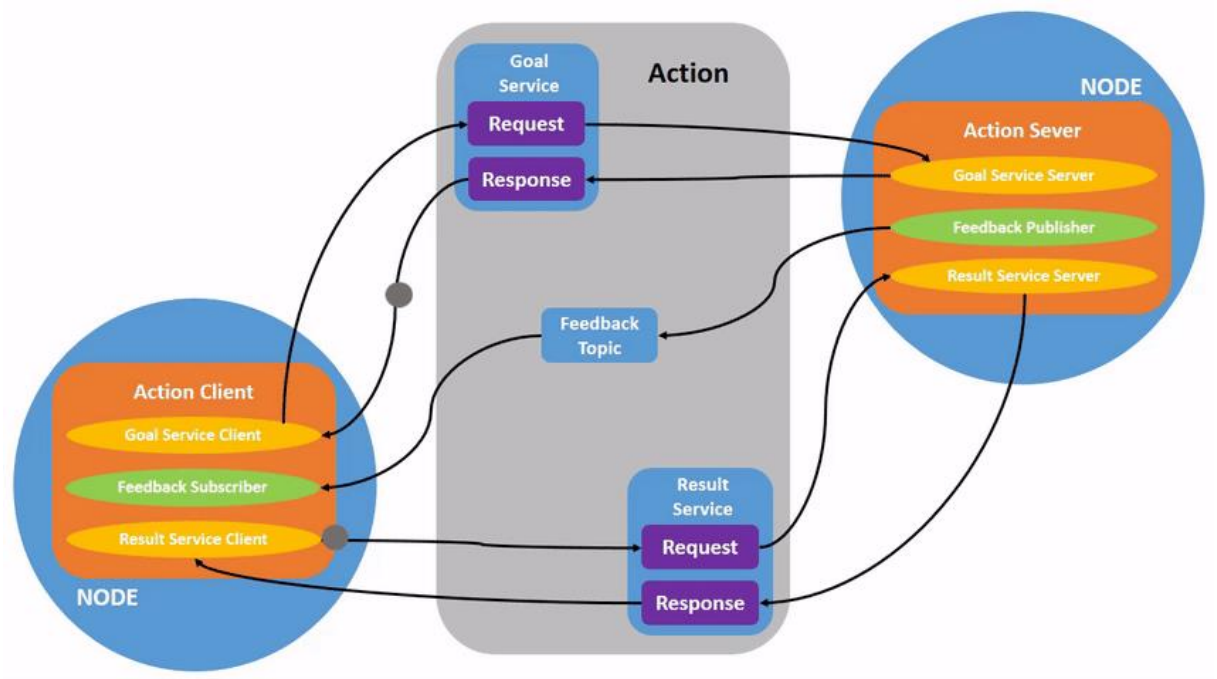
[38] Mitchell, W. C.; Staniforth, A.; Scott, I. (2006) *Analysis of Ackermann Steering Geometry* in: *Motorsports Engineering Conference & Exposition*. SAE International400 Commonwealth Drive, Warrendale, PA, United States.

[39] (2020) *Understanding ROS 2 actions* [online]. Open Robotics. https://index.ros.org/doc/ros2/Tutorials/Understanding-ROS2-Actions/ [Zugriff am: 5. Nov. 2020].

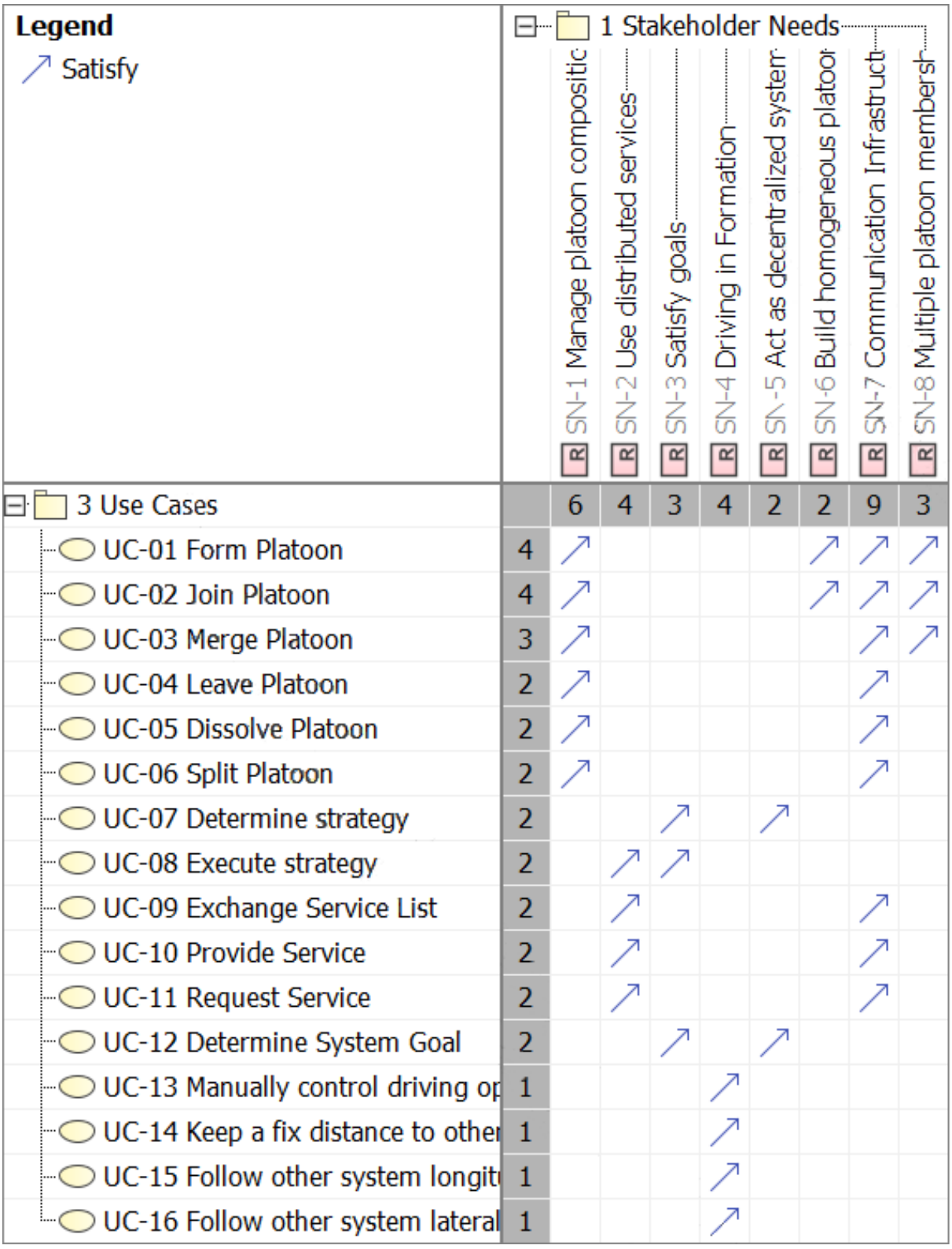
Anhang



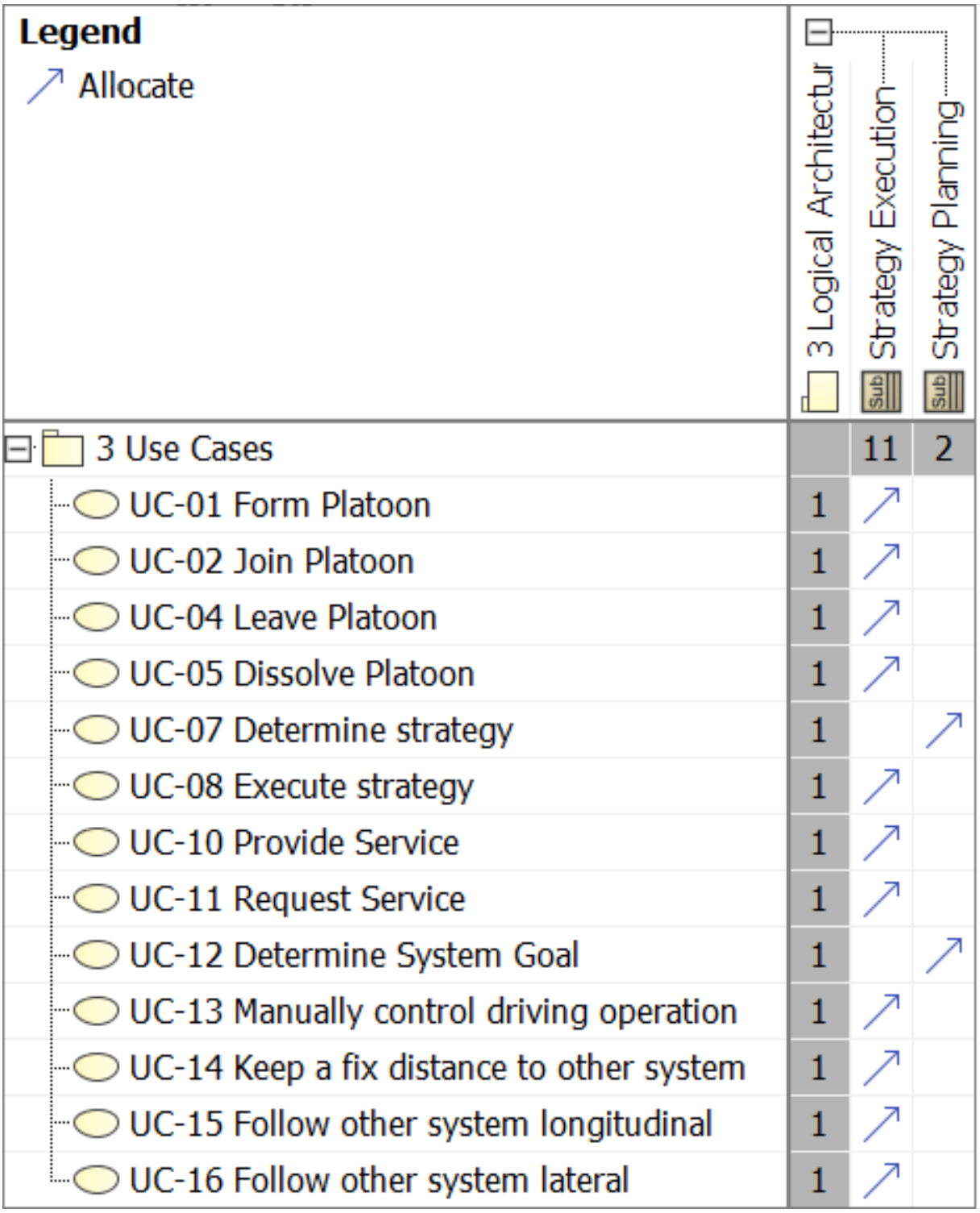
Anhang A: CeCar-Plattform, Foto: Björn Wudka



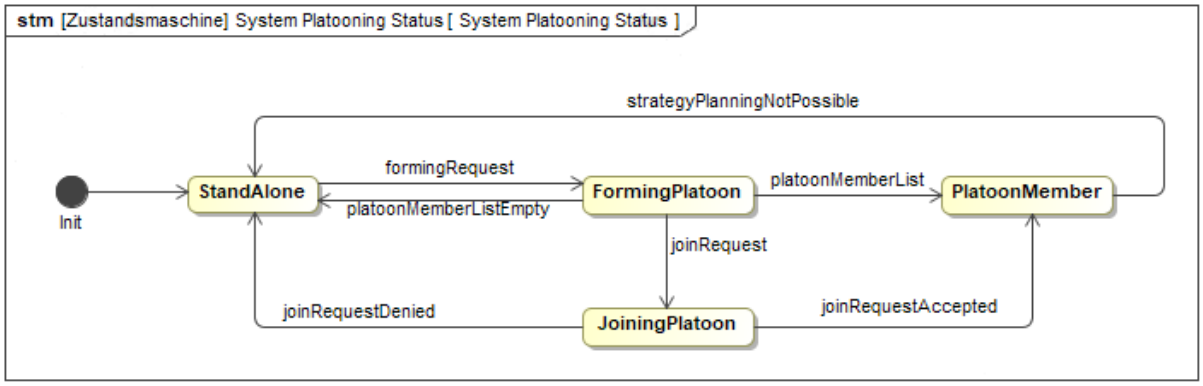
Anhang B: Aufbau von ROS2 Actions [39]



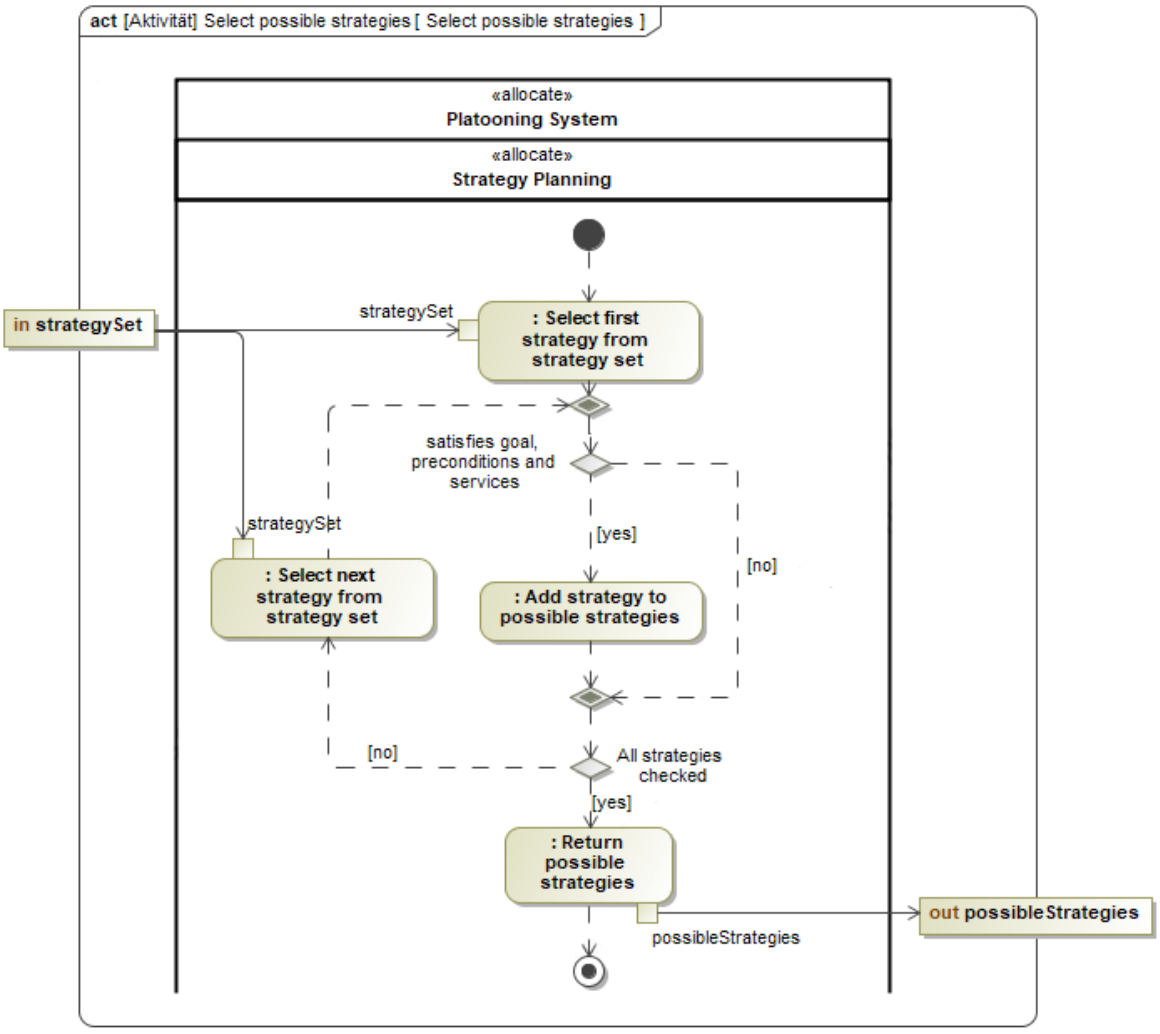
Anhang C: Matrix zur Erfüllung der Anforderungen durch die Use Cases



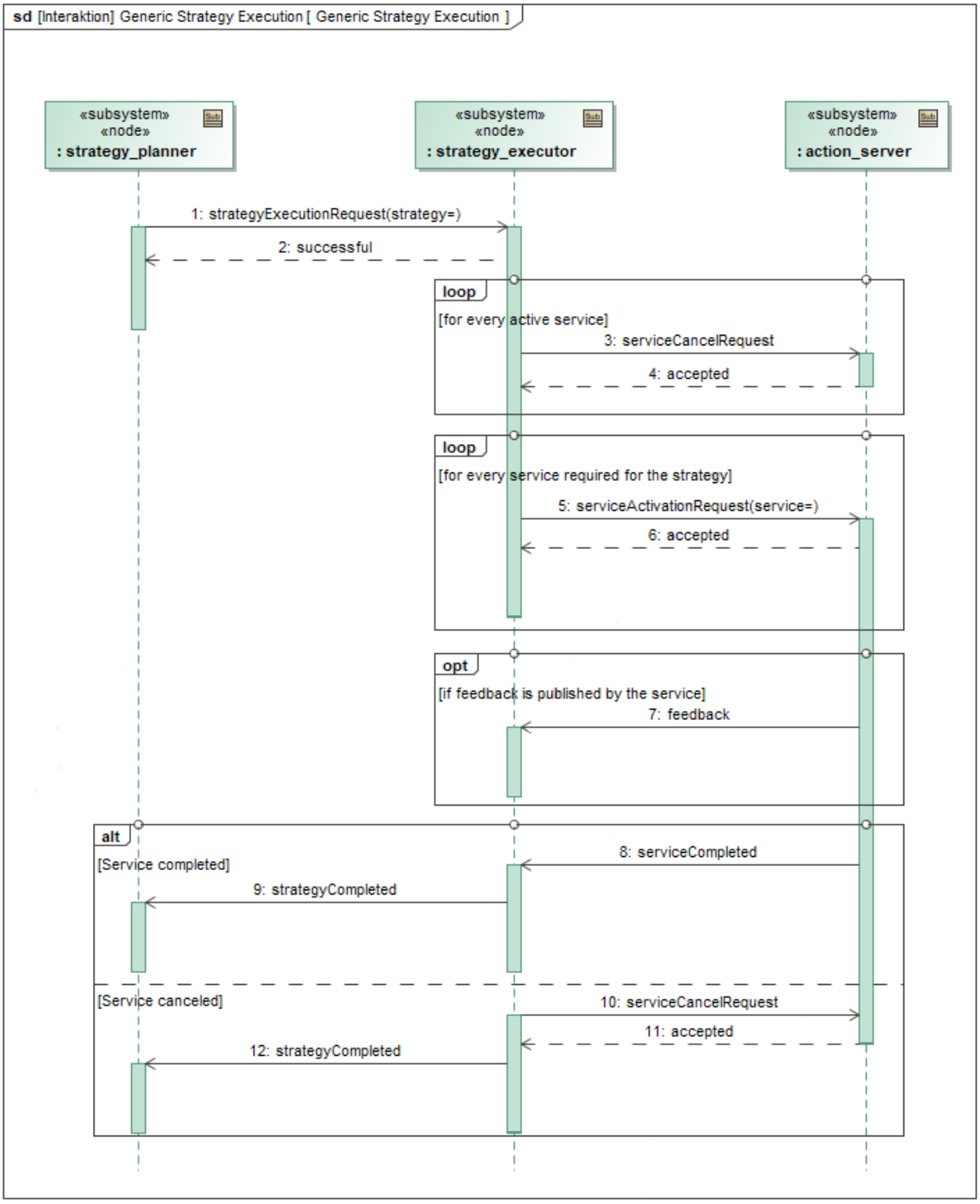
Anhang D: Matrix zur Zuordnung der Use Cases zu den Architekturelementen



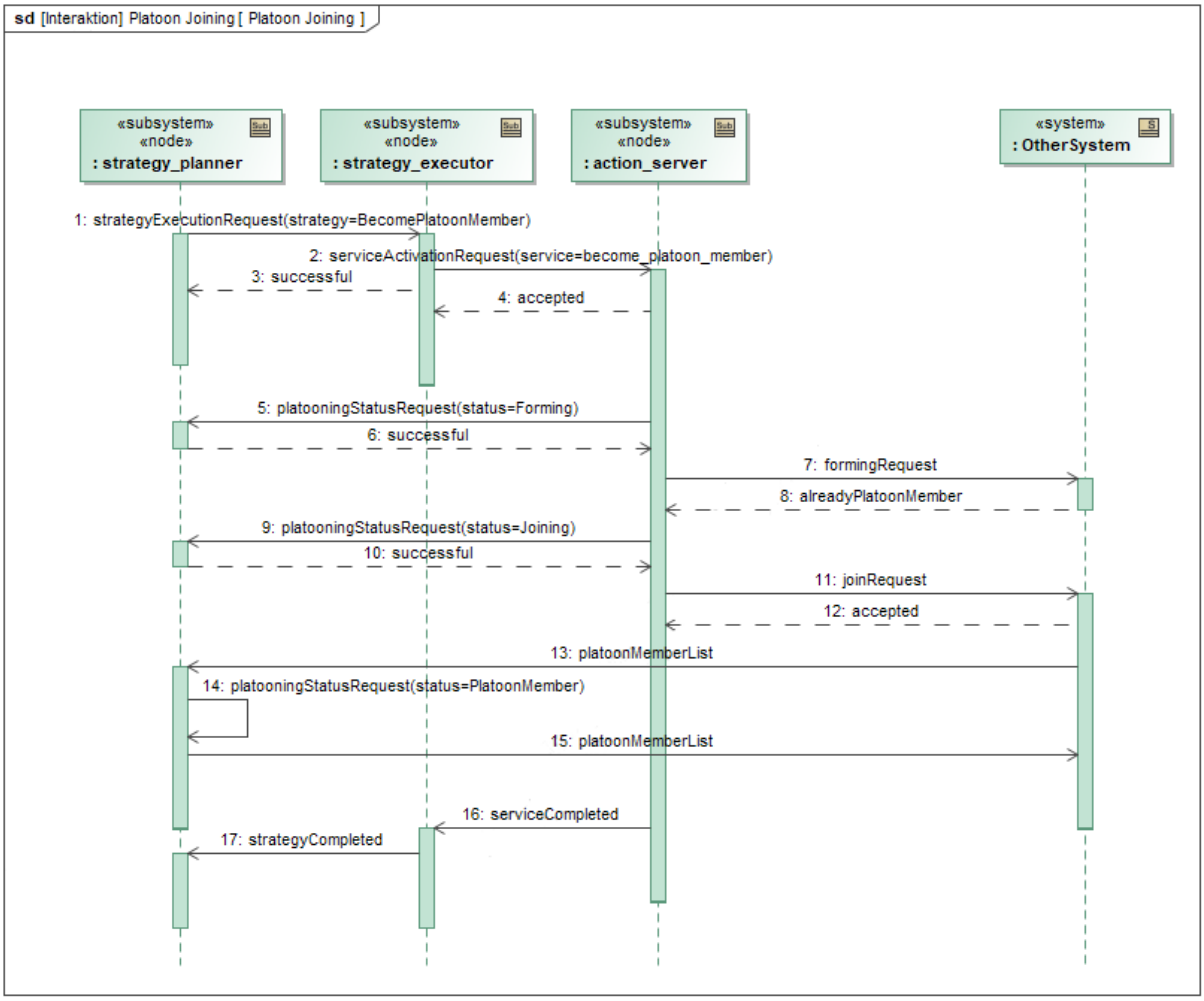
Anhang E: Zustandsdiagramm für den Platooning Status



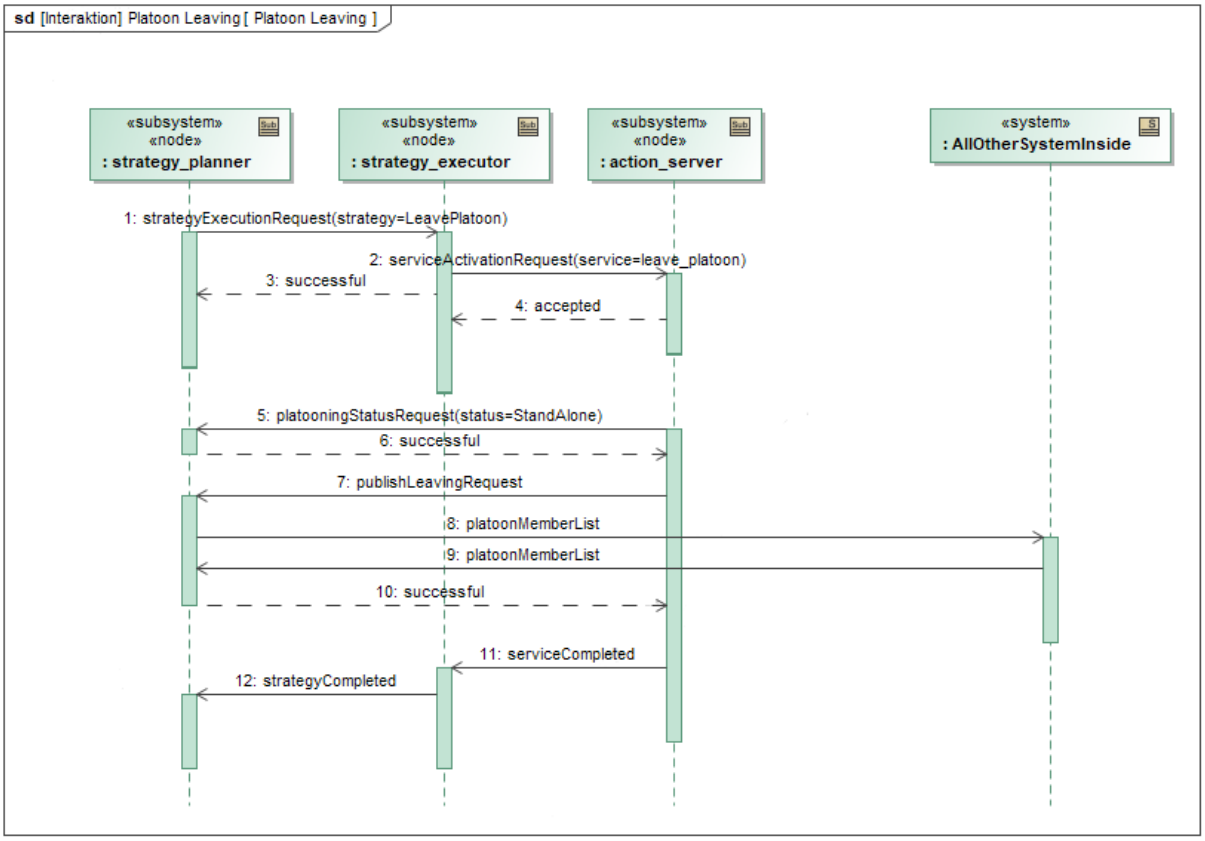
Anhang F: Aktivitätsdiagramm für die Auswertung von möglichen Strategien



Anhang G: Sequenzdiagramm für generischen Prozess der Strategieausführung



Anhang H: Sequenzdiagramm für das Platoon Joining



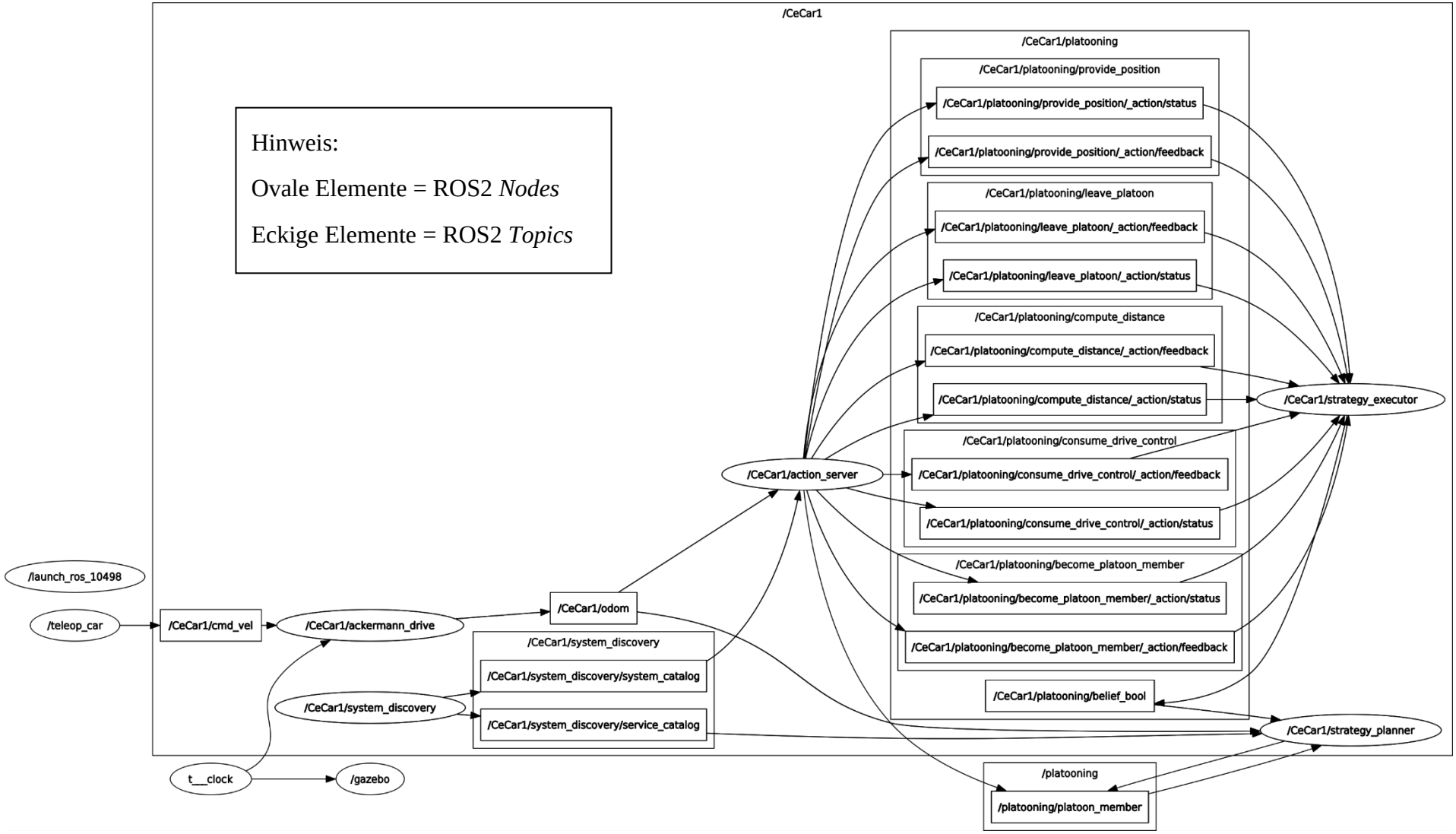
Anhang I: Sequenzdiagramm für das Verlassen eines Platoons

Anhang J: Testfälle und -auswertung für das CeCar Platooning-System

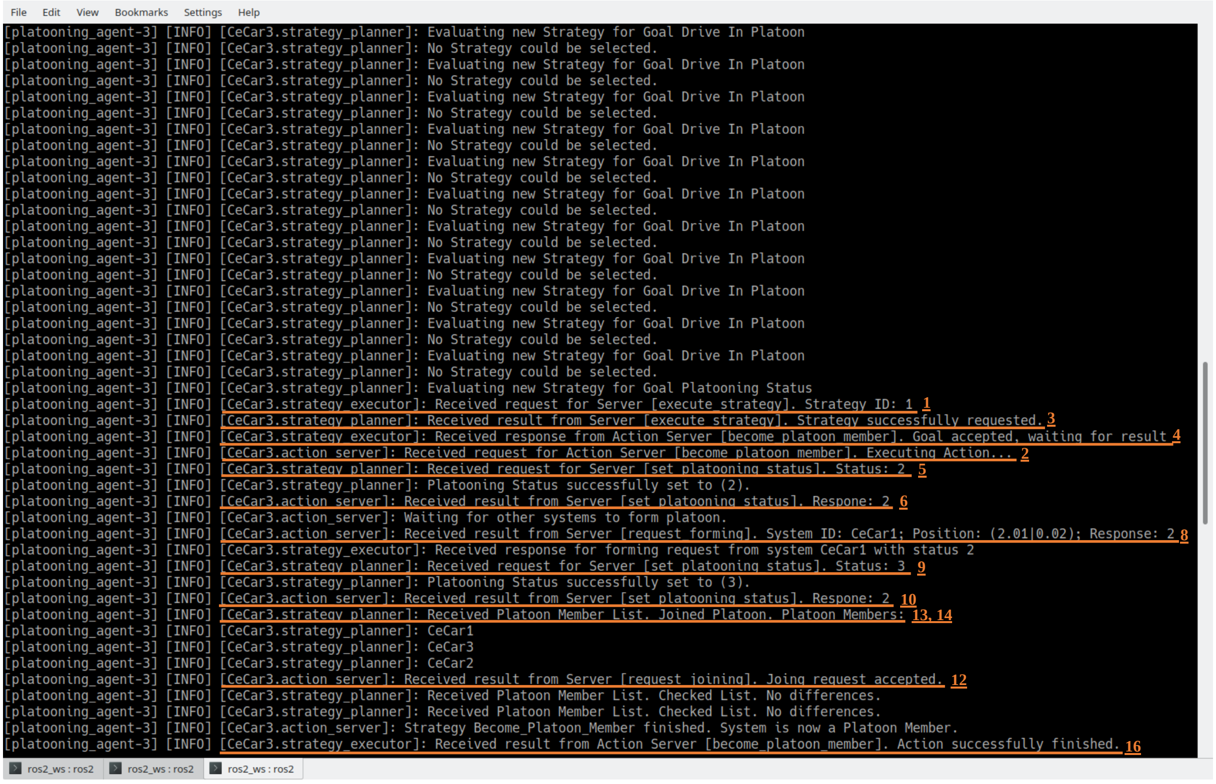
ID	Quelle	Ziel	Vorbedingung	Vorgehen	Erwartetes Ergebnis	Ergebnis	Anmerkung
TC-1	UC-07	Starten eines Platooning-Systems	Die Gazebo Simulation ist mit dem lib_gazebo_factory Plugin gestartet.	Über die Kommandozeile soll das launch-Skript für ein Platooning-System ausgeführt werden. Dabei werden ein Name und die Startposition für das Systems angegeben.	Es wird ein Modell für ein CeCar mit dem angegebenen Namen in der Gezbo Simulation erstellt. Anschließend werden die action_server, strategy_executor und strategy_planner Node gestartet und eine Strategie für das System geplant.	Bestanden	
TC-2	UC-01, UC-07, UC-08, UC-10, UC-11, UC-12	Formen eines Platoons	Zwei Platooning-Systeme wurden erfolgreich gestartet mit folgendem Wert für den *Platooning* Parameter der strategy_planner Node: *false*.	Über die Kommandozeile wird nacheinander für jedes der Platooning-Systeme der Parameter *Platooning* der strategy_planner Node auf *true* geändert.	In jedem Platooning-System wird jeweils nach dem Ändern des Platooning Parameters eine neue Strategie geplant. Dabei wird die Strategie *Become_Platoon_Member* ausgewählt und der entsprechende Service dieser Strategie wird aufgerufen. Nach Abschluss des Service haben sich die Systeme über die *platoonMemberList* auf das gleiche Verständnis der Platoon Zusammensetzung geeinigt und die korrekten Positionen der Systeme im Platoon ermittelt. Beide Systeme sind nun Teil des Platoons.	Bestanden	
TC-3	UC-02, UC-07, UC-08, UC-10, UC-11, UC-12	Joinen in ein bestehendes Platoons	Zwei Platooning-Systeme bilden bereits ein Platoon. Ein drittes Platooning-System wurde analog zu den Systemen aus TC-2 gestartet.	Über die Kommandozeile wird für das neue Platooning-System der Parameter *Platooning* der strategy_planner Node auf *true* geändert.	In dem neuen Platooning-System wird nach dem Ändern des *Platooning* Parameters eine neue Strategie geplant. Dabei wird die Strategie *Become_Platoon_Member* ausgewählt und der entsprechende Service dieser Strategie wird aufgerufen. Nach Abschluss des Service haben sich die Systeme über die *platoonMemberList* auf das gleiche Verständnis der Platoon Zusammensetzung geeinigt und die korrekten Positionen der Systeme im Platoon ermittelt. Alle drei Systeme sind nun Teil des Platoons.	Bestanden	

TC-4	UC-07, UC-08, UC-10, UC-11, UC-13, UC-14, UC-15	Gemeinsames Fahren mit der ersten Strategie im Platoon	Es existiert ein Platoon mit mehreren Mitgliedern. Der Parameter *Drive_In_Platoon* der strategy_planner Node der Platoon Member hat den Wert *true*. Der Parameter *Drive_In_Platoon_Priority* hat den Wert *3*. Nachdem das Platoon gebildet wurde, hat das System die Strategie *Drive_In_Platoon1* ausgewählt.	Die telep_car Node aus dem ackermann_vehicle_description Package wird gestartet mit der Angabe des Namens des führenden Fahrzeuges. Anschließend werden über die teleop_car Node mit der Tastatur Fahrkommandos an den Platoon Member gesendet.	Nachdem Fahrkommandos an den Platoon Leader gesendet werden, kopieren die anderen Platoon Member das Verhalten des Platoon Leaders.	Bestanden	
TC-5	UC-07, UC-08, UC-10, UC-11, UC-13, UC-14, UC-15	Gemeinsames Fahren mit der zweiten Strategie im Platoon	Es existiert ein Platoon mit mehreren Mitgliedern. Der Parameter *Drive_In_Platoon* der strategy_planner Node der Platoon Member hat den Wert *true*. Der Parameter *Drive_In_Platoon_Priority* hat den Wert *3*. Die teleop_car Node ist gestartet und publisht Fahrkommandos für den Platoon Leader.	Zunächst wird der *Drive_In_Platoon_Priority* Parameter der Platoon Member nacheinander zu *4* geändert. Anschließend werden über die teleop_car Node mit der Tastatur Fahrkommandos an den Platoon Member gesendet.	Nach dem Ändern des *Drive_In_Platoon_Priority* Parameters wählen die Platoon Member die Strategie *Drive_In_Platoon2* aus und aktivieren die entsprechenden Services. Nachdem nun Fahrkommandos an den Platoon Leader gesendet werden, regeln die anderen Platoon Member ihren Abstand zum vorderen System entsprechend des Wertes des *Distance* Parameters der action_server Node.	Bestanden	Die Abstandsregelung ist nicht perfekt ausgelegt. Eine Änderung des Abstandes zwischen den Fahrzeugen ist zu beobachten. Kommen die Fahrzeuge zum Stehen, wird der Abstand aber wieder korrekt eingeregelt.
TC-6	UC-04, UC-07, UC-08, UC-10, UC-11, UC-12	Verlassen des Platoons	Das System ist Teil eines Platoons. Der Parameter *Platooning* der strategy_planner Node hat den Wert *true*.	Der Parameter *Platooning* der strategy_planner Node des Systems wird zu *false* geändert.	Das System ändert nach der Parameteränderung seine Strategie. Die Strategie *leave_platoon* wird ausgewählt und der entsprechende Service der Strategie wird ausgeführt. Nach Abschluss der Strategie ist das System nicht mehr Teil des Platoons und die anderen Platoon Member wurden darüber informiert.	Bestanden	

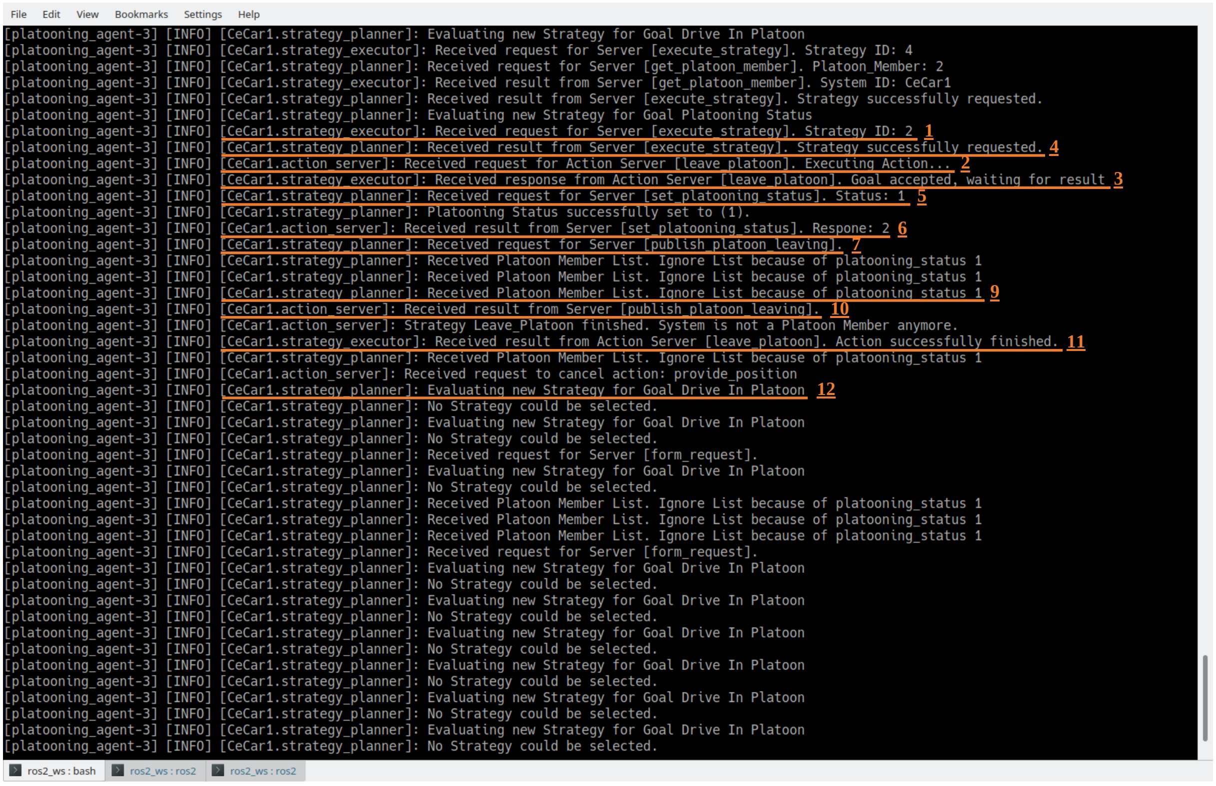
TC-7	UC-04, UC-07, UC-08, UC-10, UC-11	Verlassen des Platoons, weil keine gültige Strategie gefunden wurde	Das System ist Teil eines Platoons. Der Parameter *Drive_In_Platoon* der strategy_planner Node hat den Wert *true*.	Der Parameter *Drive_In_Platoon* der strategy_planner Node des Systems wird zu *false* geändert.	Das System ändert nach der Parameteränderung seine Strategie. Die Strategie *leave_platoon* wird ausgewählt und der entsprechende Service der Strategie wird ausgeführt. Nach Abschluss der Strategie ist das System nicht mehr Teil des Platoons und die anderen Platoon Member wurden darüber informiert.	Bestanden	Anschließend findet dennoch ein neuer Beitritt des Systems in das Platoon statt, da der *Platooning* Parameter nach wie vor den Wert *true* hat.
TC-8	UC-05	Auflösen des Platoons	Alle Systeme bis auf eines haben ein existierendes Platoon verlassen.	Keine Handlung erforderlich.	Das System erkennt, dass alle anderen Systeme das Platoon verlassen haben. Dadurch ist das Platoon aufgelöst und der Platooning Status wird auf *Stand_Alone* gesetzt.	Bestanden	



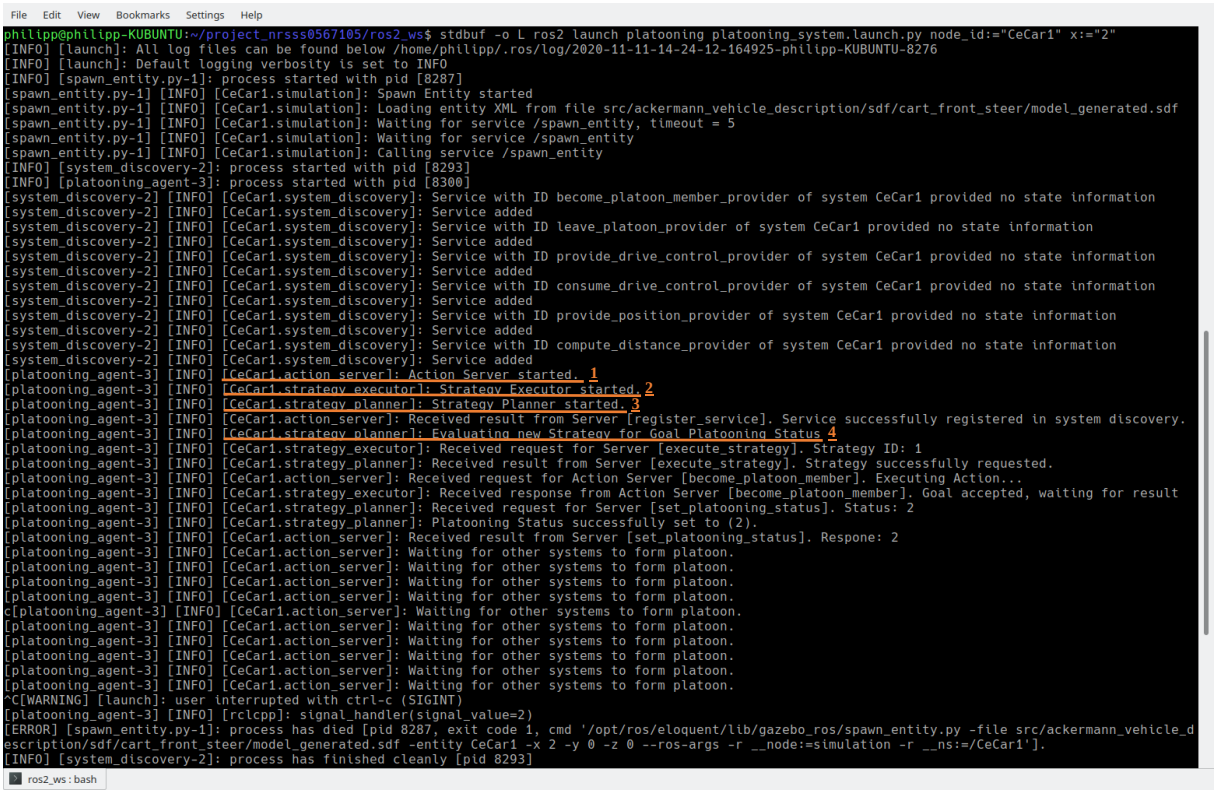
Anhang K: ROS-Graph für die Testumgebung eines CeCars



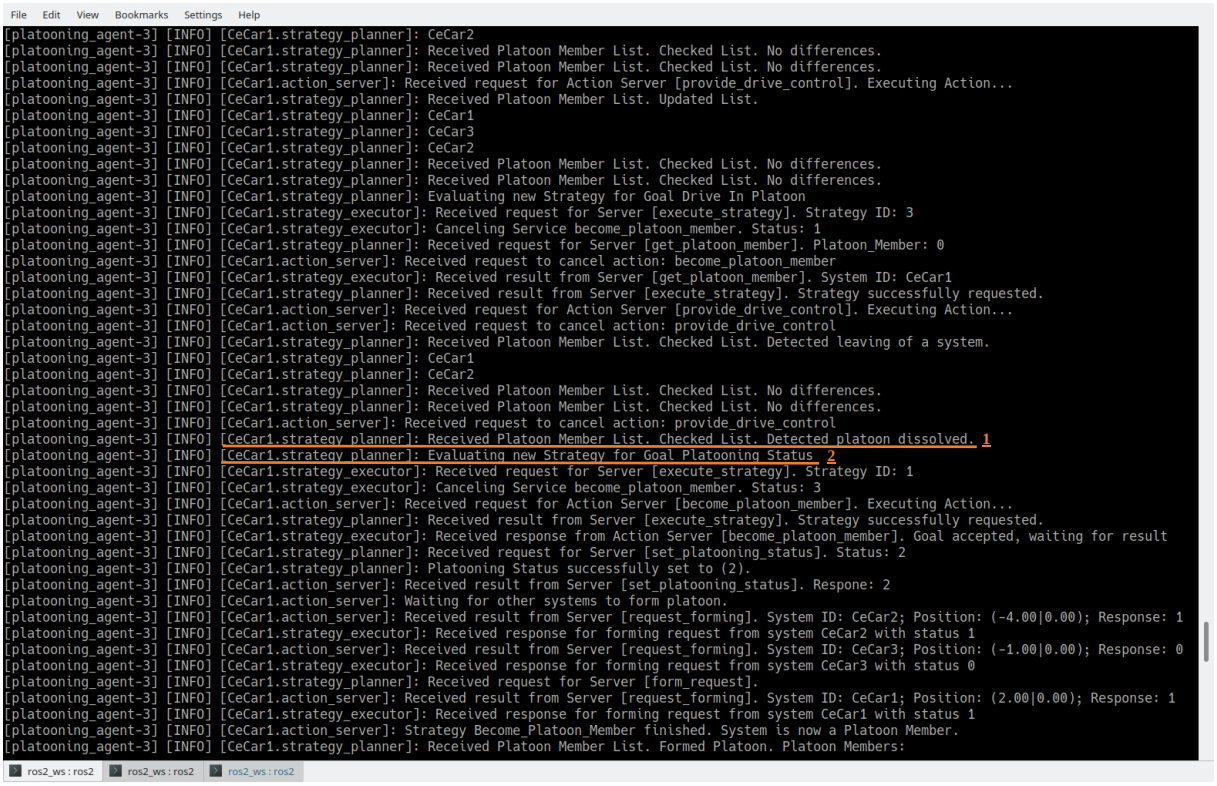
Anhang L: Konsolenausgabe für das Platoon Joining



Anhang M: Konsolenausgabe für das Platoon Leaving



Anhang N: Konsolenausgabe für das Starten des Platooning Systems



Anhang O: Konsolenausgabe für das Platoon Dissolving