TODOs:
1.: Einbinden der Forschungsfragen in Text
2.: Ausformulieren der Forschungsfragen

6.1 Volle Unterstützung von neuer oder geänderter Hardware, Software und Bibliotheken

Dieses Kapitel beschreibt die Änderungen an den MUML-Modellen, den internen Ressourcen und dem Post-Processing-Ablauf sowie die Analysen und Abwägungen, die jeweils zuvor durchgeführt wurden.
[TODO: Noch etwas mehr hinschreiben]

Im Rahmen dieses Kapitels werden also die folgenden Forschungsziele beantwortet:

FZ1.1 Welche Schritte des Arbeitsablaufes werden von welchen Komponenten und mit welchen Daten ausgeführt? [TODO]
FZ1.3 Welche Kriterien sind für die Auswertung von Ansätzen relevant?
FZ2.1 Werden die entsprechend betroffenen Artefakte standardmäßig integriert? Und falls ja, wie?
FZ2.2 Welche validen Anpassungsansätze gibt es?
FZ2.3 Welcher Anpassungsansatz erfüllt die Kriterien am besten und wird umgesetzt werden?
[(((TODO: Bessere Formulierung:)))]
Die FZ 1.3, 2.1, 2.2 und 2.3 sind nicht nur für ein Thema oder einen Teil relevant, sondern durch die verschiedenen Änderungsstellen mehrmals. Deshalb treten sie in diesem Kapitel jeweils mehrmals Bezüge auf sie auf bzw. werden für die jeweilige Stelle als beantwortet erwähnt.

Die notwendigen Anpassungen an den aktuellen Stand der Roboterautos werden durch die folgenden Anforderungen spezifiziert:

ANF 6.1: Anpassung der MUML-Modelle an die von dieser Ausarbeitung verwendete Version der Roboterautos
ANF 6.2: Bessere Integration der Bibliothek SofdCar-HAL
ANF 6.3: Serielle Kommunikation statt I2C-Kommunikation
ANF 6.4: Anpassung der Post-Processing-Schritte
[(((Vielleicht: Die exakte Anforderungsbeschriftung muss noch verbessert werden, aber die Idee/ das Konzept kommt herüber und die Aufteilung sollte passen. Bitte bessere Ideen/Vorschläge.)))]

Die Analysen und vorgenommenen Änderungen sind auf die folgenden Abschnitte aufgeteilt:

In 6.1.1 analysiert, welche Schritte des Arbeitsablaufes werden von welchen Komponenten und mit welchen Daten ausgeführt werden.
In 6.1.2 wird die Anpassungen an den MUML-Modellen beschrieben.
In 6.1.3 wird die Änderung der Kommunikationsressourcen von I2C auf Seriell erklärt.
In 6.1.4 wird die Analyse des Post-Processings und seine Anpassung beschrieben.

6.1.1: Analyse, welche Schritte des Arbeitsablaufes werden von welchen Komponenten und mit welchen Daten ausgeführt werden:

In diesem Abschnitt wird beschrieben, welche Schritte des Arbeitsablaufes von welchen Komponenten und mit welchen Daten ausgeführt werden. FZ1.1 wurde mangels Erfahrung beim Schreiben von Forschungsarbeiten aus Angst, wegen möglichen nicht im Proposal erwähnten

Schritten schlecht bewertet zu werden, breit formuliert. Die eigentlichen für die Leserschaft relevanten Informationen zu diesem Forschungsziel werden in diesem Abschnitt beschrieben. Hierbei werden zum besseren Vergleichen die Nummerierungen der Schritte aus [oberes Diagramm von https://github.com/SQA-Robo-Lab/MUML-CodeGen-Wiki/blob/main/user-documentation/main.md#t32-deployment-configuration-aka-container-transformation] übernommen.

[(((Wie weit soll ich hier ins Detail gehen? Ich weiß einfach nicht, wie weit ich der Leserschaft die für sie wahrscheinlich nicht notwendigen Details ersparen soll und was ich noch schreiben muss, um wissenschaftlich korrekt zu sein.)))]

T3.2: „Deployment Configuration aka Container Transformation“, also Kontainertransformation: Hier wird per Export-Operation „MeachatronicUML“/“Container Model and Middleware Configuration“ aus der Datei „roboCar.muml“, genauer gesagt anhand der Systemallokationen darin, den Modellen in in dem MUML-Plug-Ins „mechatronicuml-cadapter-component-container“ und „mechatronicuml-psm“, den Einstellungen für die Middleware und dem Zielort die Datei „MUML_Container.muml_container“ mit den verschiedenen Kontainern für die verschiedenen Komponenten und deren Konfigurationen generiert.

…

6.1.2: Anpassungen an den MUML-Modellen:
Dieser Abschnitt beschreibt zusammengefasst die Änderungen, die an den verschiedenen MUML-Modellen vorgenommen wurden, um Kompatibilität zu den unter dieser Ausarbeitung verwendeten Roboterautos herzustellen. Für die genauen Details siehe Ergänzungsmaterial „Änderungen an MUML-Modellen“ [TODO: Verweis].
Hier ist die Antwort auf FZ 2.1, dass die Modelle der verschiedenen Komponenten und deren Instanzen sowie die Ports und Verbindungen zwischen diesen in den verschiedenen Diagrammen eingetragen werden.

6.1.2.1: Mögliche Ansätze:
Bei den Analysen, wie die Anpassungen durchgeführt werden können, wurden hinsichtlich der Software-Komponenten zwei Ansätze gefunden:

1. Der erste Ansatz beruht auf dem Beibehalten der alten Architektur und dem Ersetzen der alten Komponenten mit entsprechenden Komponenten. Architekturänderungen wären einzig für das Richtungslenken erfolgt. Die Komponente „DriveController“ hätte also seitens des Modells direkten Zugriff auf konkrete Kindklassen der abstrakten Klassen „SteerableAxle“ und „Motor“ aus Sofdcar-HAL erhalten. Diese repräsentieren das Einstellen der Fahrgeschwindigkeit und die Lenkachse und sollen eigentlich von einer „DriveController“-Klasse verwendet werden. Das Konzept der Modularisierung würde also durchbrochen werden.
2. Das Priorisieren der Modularisierung, d.h. die Komponente „PowerTrain“, die das Einstellen der Geschwindigkeit repräsentiert, würde mit einer Kindklasse der abstrakten Klasse „DriveController“ ersetzt werden. So würde die Modularisierung beibehalten werden und alles aus dieser Bibliothek könnte so arbeiten oder verwendet werden, wie es jeweils vorgesehen ist. Um Verwechslungen vorzubeugen, würde „DriveControl“ zu „CourseControl“ umbenannt werden. Dies würde auch zu einer passenden Namensgebung passen, weil dann diese Kontrollkomponente vielmehr den Kurs auf den Fahrbahnen festlegt, aber nicht für Lenken beim Befolgen der Bahn verantwortlich ist.

Diese beiden Möglichkeiten sind die Antwort auf FZ 2.2 für dieses Thema.

Es konnten keine Bewertungskriterien für Modelländerungen gefunden werden, also werden Kriterien für die Code-Qualität verwendet. Dies beantwortet für dieses Thema FZ 1.3. in der

Codequalität ist Modularisierung ein Qualitätsaspekt (siehe [Taxonimie, Codequalität]), also wurde beschlossen, Ansatz 2 umzusetzen.

6.1.2.2: Umsetzung von Ansatz 2:
Im folgenden werden die hierfür durchgeführten Änderungen zusammengefasst beschrieben. Hierbei ist zu beachten, dass sozusagen beim Durchführen von Änderungen in der korrekten Stelle diese an andere Dateien weiter propagiert werden und man deshalb für manche Vorgänge nicht an allen Stellen die Änderungen vornehmen muss. So wird z.B. unter „Model/component/“ in „roboCar.component_diagram“ die statische strukturierte Komponente „RoboCar“ um einen Komponententeil namens “steeringAngle“ erweitert und MUML propagiert dies u.a. nach „Model/instance/roboCar2.componentinstanceconfiguration_diagram“ weiter.

1. Umbenennungen von Komponenten:
   In verschiedenen Diagrammen wurden die Komponenten „PowerTrain“ und „DriveControl“ entsprechend nach „DriveController“ und „CourseControl“ umbenannt.
2. Anpassung der Datentypen an den Ports für Geschwindigkeitswerte:
   In der Bibliothek Sofdcar-HAL ist in den Funktionen der abstrakten Klasse „DriveController“ die Geschwindigkeit als Wert des Datentyps „int16“ eingetragen. Folglich wurden bei den ehemaligen „PowerTrain“ und „DriveControl“ bzw. nun „DriveController“ und „CourseControl“ die Ports für die Geschwindigkeit auf „Primitive Data Type int16“ geändert.
3. Ports für die Winkel-Werte:
   Bei „DriveController“ und „CourseControl“ wurden die Ports für die Übertragung der Winkelwerte hinzugefügt und miteinander verbunden. Der Datentyp „Primitive Data Type int8“ wurde gewählt, weil in der Bibliothek Sofdcar-HAL in den Funktionen der abstrakten Klasse „DriveController“ der Winkel als Wert des Datentyps „int16“ eingetragen ist.
4. Hinzufügen des Lenk-Servos:
   Für den Lenkservo wurden in verschiedenen Diagramme die entsprechenden Elemente hinzugefügt. Innerhalb der Plattform-Diagramme ist er als „steeringServo“ modeliert.
   Für die Kommunikation mit dem Servo wurde das Protokoll „Link Protocol ServoControl“ definiert, die verschiedenen Ports erstellt und in den Diagrammelementen für die Fahrer-Boards bzw. ECUs ist sein Port als „HW Port SteeringAxle“ definiert. Es wurden auch die entsprechenden Verbindungen vorgenommen.
5. Entfernen überschüssiger Motoren: Die Modelle arbeiteten mit zwei virtuellen Elektromotoren. Für das Modellieren wurde der linke Motor „leftMotor“ zu „fixedMotor“ (fixierter Motor) umbenannt und der rechte Motor „rightMotor“ entfernt.
6. Anpassung von „roboCarLibraries.osdsl“ [TODO: Dateitypennamen nachsehen]:
   Für das Anpassen der zu verwendenden APIs wurde „MotorDriver“ mit „DriveController“ ersetzt und in dessen Funktionsdefinitionen bzw. angegebenen Schnittstellen wurde in „setSpeed“ der Parametertyp auf „int16“ angepasst und die Funktionsdefinition „void setAngle(int8 angle);“ hinzugefügt.
   Dies hat den folgenden Hintergrund: Aus der Bibliothek „MotorDriver“ wurde die Klasse „MotorDriver“ wurde von Stürner […, S. 131 bzw. Ergänzungsmaterial A „Supplementary Material for the Robot Cars“] für das Ansteuern der Motoren verwendet. Diese Funktionalität wird in der „Sofdcar-HAL“-Bibliothek indirekt von der abstrakten Klasse „DriveController“ durchgeführt, aber die Nutzung ist ähnlich, so dass die beschriebenen Änderungen ausreichen.
7. Anpassung der Allokationen:
   Hier wurden in den Coallokationsdefinitionen „PowerTrain“ und „DriveControl“ entsprechend mit „DriveController“ und „CourseControl“ ersetzt.

Somit wurde der ANF 6.1 „Anpassung der MUML-Modelle an die von dieser Ausarbeitung verwendete Version der Roboterautos“ erfüllt. Zusätzlich wurde dabei die Integration der Bibliothek „SofdCar-HAL“ verbessert, wodurch ANF 6.2 „Bessere Integration der Bibliothek SofdCar-HAL“ ebenfalls erfüllt wird.

6.1.3: Änderung der Kommunikationsprotokolls zwischen den Boards innerhalb von einem Roboterauto von I2C auf Seriell:
Um die Zuverlässigkeit der Kommunikation beim Nachrichtenaustausch innerhalb desselben Roboterautos zwischen dem Koordinator-AMK und dem Fahrer-AMK zu verbessern, hat Georg Reißner vorgeschlagen, das Kommunikationsprotokoll von I2C auf Seriell abzuändern. [TODO: Noch überlegen, wie ich das mit den Interfaces und dem Test erkläre.]

Nach einer erfolglosen Suche, wie die serielle Kommunikation in MUML oder seinen Erweiterungen verwaltet wird, wurde beschlossen, innerhalb der MUML-Plug-In-Projekte die hierfür verwendeten Dateien zu ersetzen und alle Beschreibungen und Bezüge darauf anzupassen. Das Hinzufügen der Seriellen Kommunikation als Typ ist wahrscheinlich prinzipiell möglich, aber die zuvor genannte Suche ist ein starkes Indiz für erforderliche komplexe Erweiterungen, für die die Kenntnisse fehlten. Dies beantwortet hierfür FZ 2.1 und 2.2. Die anderen FZ 1.3 und 2.3 entfallen mangels möglicher bekannter oder konzipierbarer Möglichkeiten.

Im Laufe dieses Kapitels wird also die Erfüllung von ANF 6.3 „Serielle Kommunikation statt I2C-Kommunikation“ beschrieben.

6.1.3.1: Einfügen der Code-Dateien für die serielle Kommunikation:
Die Generation des Komponenten-Codes bezieht die Dateien „I2cCustomLib.cpp“ und „I2cCustomLib.hpp“ aus internen Ressourcen.
In dem Sourcecode bzw. in den Eclipse-Plug-In-Projekten waren die beiden „I2cCustomLib“-Dateien, in dem MUML-Plug-In-Projekt „mechatronicuml-cadapter-component-container“ unter dem Package „org.muml.arduino.adapter.container“ unter dem Ordnerverzeichnis „resources/container_lib“ abgelegt. Dort wurden sie mit Reißners Code, also „SerialCustomLib.cpp“ und „SerialCustomLib.hpp“ ersetzt.

6.1.3.2: Anpassen der Bezüge:
In dem selben Package unter „/bin/org/muml/arduino/adapter/container/main“ wurden in den Dateien „MainFile.mtl“ und „MainFile.emtl“ die Bezüge von „I2cCustomLib.hpp“ auf „SerialCustomLib.hpp“ geändert.

In dem selben Ordnerverzeichnis wurde in den Dateien „main.mtl“ und „main.emtl“ sowie in „src/org/muml/arduino/adapter/container/main/main.mtl“ wurde der Kommentartext „For I2C communication, the Arduino is expected to be using its I2C pins. See more information in the I2cCustomLib.hpp in the respective ECU's directory.“ nach „For the serial communication, the Arduino is expected to be using its I2C pins. See more information in the SerialCustomLib.hpp in the respective ECU's directory.“ umgeschrieben.

6.1.4: Anpassung der Post-Processing-Schritte:
Das Post-Processing nach [Link] besteht aus einer Vielzahl an manuell auszuführenden Schritten, also werden in diesem Abschnitt die Änderungen nur zusammengefasst bzw. die groben Details beschrieben. Bei einem Vorher-Nachher-Vergleich der generierten unvollständigen Codeteile vor und nach den Änderungen an den Modellen und Ressourcen stellte sich heraus, dass die Änderungen an den MUML-Modellen zu anders benannten und zusätzlichen Dateien sowie zu leicht anderen Dateiinhalten führten. Nicht alle von diesen erfordern Änderungen an dem Post-

Processing-Ablauf, weil über die Code-Generatoren beispielsweise die Referenzen auf verschiedene Objekte bereits angepasst wurden.
Die genauen Details können im Ergänzungsmaterial „Änderungen am Post-Processing-Ablauf“ [TODO: Verweis] nachgelesen werden.
In den folgenden Abschnitten werden zu den Änderungen die jeweiligen Anpassungen beschrieben. Diese stellen auch die Erfüllung von ANF 6.4 „Anpassung der Post-Processing-Schritte“ dar.

Die erwähnte manuelle Ausführung beantwortet FZ 2.1. Die notwendigen Änderungen sind sozusagen eindeutig und unterscheiden sich lediglich in der Reihenfolge der Ausführung, was FZ 2.2 beantwortet. Somit entfällt FZ 1.3. FZ 2.3 wird kompakt in den folgenden Abschnitten beantwortet und in dem oben erwähnten Ergänzungsmaterial [TODO: Verweis] ausführlich.

6.1.4.1: Berücksichtigung anders benannter Dateien:
Wegen der in 6.1.2 [TODO: Verweis] beschrieben Namensänderung kommt es in den generierten unvollständigen Codeteilen zu anderen Namen, was sich in Form von anders benannten Dateien und angepassten Referenzen auswirkt.
So wurde beispielsweise aus „driveControl“ in allen Dateinamen und Bezügen „courseControl“, was wiederum beim Post-Processing-Ablauf berücksichtigt werden muss.

Alt	Aktuell
driveControl	courseControl
PowerTrain	DriveController
I2CCustomLib	SerialCustomLib

6.1.4.2: Weitere auszufüllende und nachzubearbeitende API-Dateien:
Das Hinzufügen von weiteren Ports, beispielsweise für den Winkel bei der Anpassung von „PowerTrain“, verursacht weitere Dateien und zu implementierende Befehle. So gibt es in den API-Mappings (Ordner „APImappings“) die neu hinzugkommenen Dateien „CI_DRIVECONTROLLERFDRIVECONTROLLERanglePortaccessCommand.c“, „CI_DRIVECONTROLLERFDRIVECONTROLLERanglePortaccessCommand.h“, „CI_DRIVECONTROLLERSDRIVECONTROLLERanglePortaccessCommand.c“ und „CI_DRIVECONTROLLERSDRIVECONTROLLERanglePortaccessCommand.h“, in denen der Befehl „*velocity = SimpleHardwareController_DriveController_GetSpeed();“ eingetragen werden muss. [(((Entfernt: Diese Dateien verursachen jedoch keine zusätzliche Arbeit durch ihr Einbinden per „#include“-Anweisungen, weil die Code-Generation flexibel genug ist, um dies bereits eigenständig durchzuführen.)))]