5. Analyse des bisherigen Ablaufes
Die in der Einführung erwähnte manuelle Ausführung musste zunächst untersucht werden, um Änderungen oder Automatisierungen zu ermöglichen.
Hierfür wird in diesem Kapitel eine Übersicht an Arbeitsschritten mit einer jeweiligen Beschreibung der jeweils relevanten Informationen aufgestellt.
Hierbei wiederum werden zum besseren Nachschlagen oder Vergleichen die T-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] mit angegeben.

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?
FZ1.2 Welche Schritte des Arbeitsablaufes müssen manuell durchgeführt werden?

Davon wird FZ1.1 durch die folgenden Beschreibungen der Arbeitsschritte beantwortet.

Arbeitsschritt 1: T3.2: „Deployment Configuration aka Container Transformation“, also Kontainertransformation:
Hier wird per Export-Operation „MeachatronicUML“/“Container Model and Middleware Configuration“ mit den Einstellungen für die Middleware (hier „MQTT and I2C Middleware Configuration“) und einem Zielort aus der Datei „roboCar.muml“ die Datei „MUML_Container.muml_container“ mit den verschiedenen Kontainern für die verschiedenen Komponenten und deren Konfigurationen generiert.
Hierbei wird intern anhand der Systemallokationen-Ressource in „roboCar.muml“ [(((, den Modellen in in dem MUML-Plug-Ins „mechatronicuml-cadapter-component-container“ und „mechatronicuml-psm“))))] gearbeitet.

Arbeitsschritt 2: T3.6 und T3.7: „Container Code Generation“, also Generation des Kontainer-Codes:
Dies wird per [Rechtsklick auf die in dem vorherigen Schritt generierte „MUML_Container.muml_container“-Datei]/“mumlContainer“/“Generate Arduino Container Code“ gestartet und es erstellt einen Ordner namens „arduino-containers“. In diesem befinden sich der Ordner „APImappings“ für die verschiedenen APIs und die Ordner mit der Namensendung “ECU“ für die verschiedenen ECUs bzw. AMKs. In dem letzteren befinden sich Code-Dateien wie einige interne Bibliotheken, z.B. die Dateien „I2CCustomLib.cpp“ und „I2CCustomLib.hpp“ aus dem MUML-Plug-In-Projekt „mechatronicuml-cadapter-component-container“, Package „org.muml.arduino.adapter.container“ und Ordnerverzeichnis „resources/container_lib“.
Jedoch fehlt noch viel Code, beispielsweise für das Verhalten.
Intern wird „GenerateAll“ aus dem Package „org.muml.arduino.adapter.container.ui.common“ aus dem MUML-Plug-In-Projekt „mechatronicuml-cadapter-component-container“ bzw. dessen Unter-Projekt „org.muml.codegen.componenttype.export.ui“ verwendet, um die Generierung zu starten.

Arbeitsschritt 3: T3.3: „Component Code Generation“, also Generation der Komponentencodes:
Hier werden per Export-Operation „MeachatronicUML“/“Source Code_workspace“ mit den Einstellungen für die Zielplattform (hier „Component Type ANSI C99“) und einem Zielort aus der Datei „roboCar.muml“ die Komponentencodes generiert und in dem Eclipse-Projektordner in dem Ordner „fastAndSlowCar_v2“ platziert.

Arbeitsschritt 4: T3.8: „Program Building“:
Diese Phase besteht genau genommen aus dem Post-Processing und dem Deployment bzw. Hochladen auf die verschiedenen AMKs.
Beim Post-Processing wird zunächst der generierte, aber noch nicht direkt verwendbare, sondern unvollständige, Code in einen direkt verwendbaren und vollständigen Zustand gebracht. So kann die Arduino-IDE die verschiedenen „.ino“-Dateien jeweils korrekt kompilieren.
Stürner hat für seine Modelle und Roboterautos einen passenden Post-Processing-Ablauf in seiner Ausarbeitung [TODO: Verweis] beschrieben. Hierbei erklärt er aber auch, dass das Post-Processing infolge eines nicht korrekt kompilierenden Codegenerator-Quellcodes notwendig ist.
Für die unter dieser Ausarbeitung verwendeten Version der Roboterautos und der Bibliothek „Sofdcar-HAL“ hat Georg Reißner einen angepassten Verlauf beschrieben, der unter [https://github.com/SQA-Robo-Lab/Overtaking-Cars/blob/hal_demo/arduino-containers_demo_hal/deployable-files-hal-test/README.md] nachgeschlagen werden kann.
Das Deployment besteht darin, die verschiedenen „.ino“-Dateien mit der Arduino-IDE zu öffnen und jeweils auf den entsprechenden AMK hochzuladen [Stürner].

FZ 1.2 kann mit der folgenden Zusammenfassung zu den Beschreibungen von oben beantwortet werden: In den Arbeitsschritten 1 bis 3 wird jeweils ein interner Vorgang ggf. konfiguriert und gestartet. Von Schritt 4 ist das Post-Processing gänzlich manuell. Danach werden beim Deployment die verschiedenen „.ino“-Dateien über das Ordnersystem geöffnet. Bei jeder Datei werden entsprechend Boardtyp und Verbindungsport eingestellt sowie der Uploadvorgang gestartet.