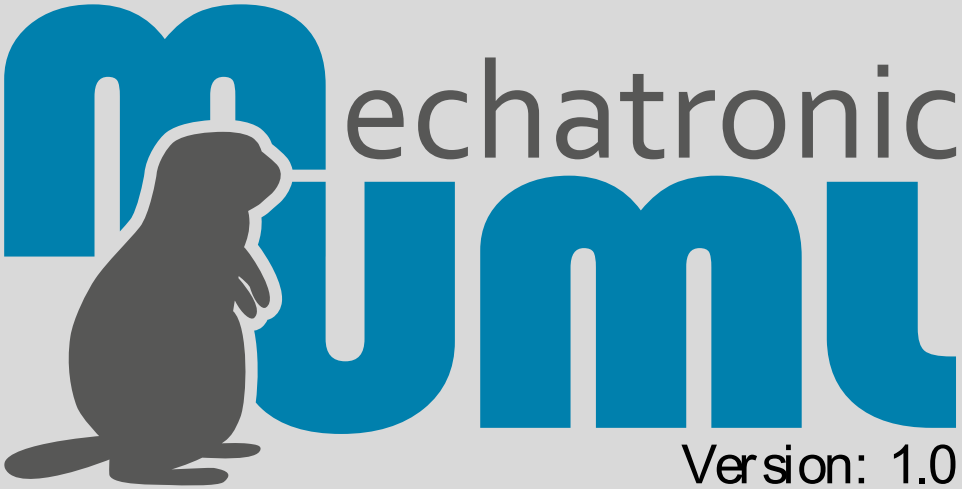
Universität Stuttgart

Automatisierung von Code-Generation, -Integration und -Deployment von autonomen Fahrfunktionen

Masterarbeits-Abschlusspräsentation
Betreuer: Marcel Weller, M.Sc.



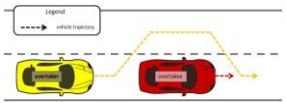
Version: 1.0 [27]

Sebastian Baumfalk

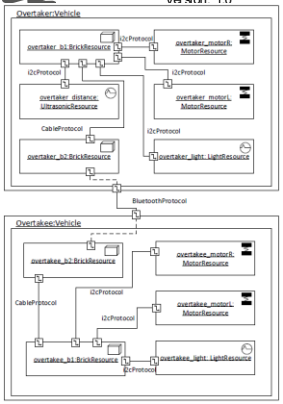
Motivation (1/4)

Problem (1/2)

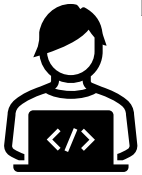
[1]

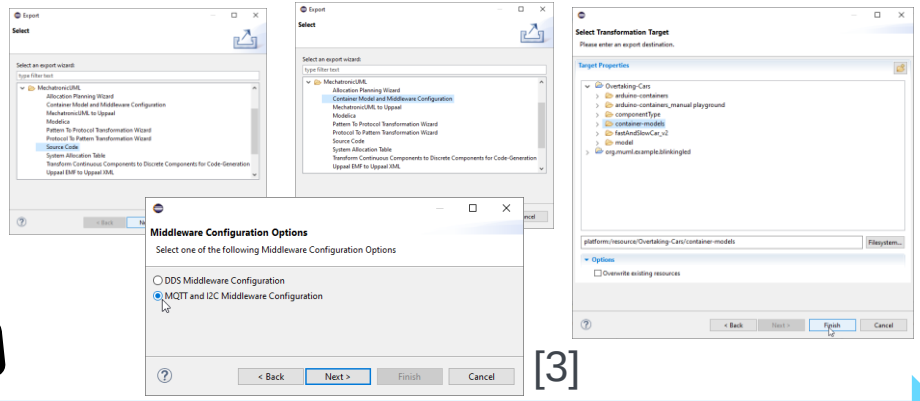


[27]



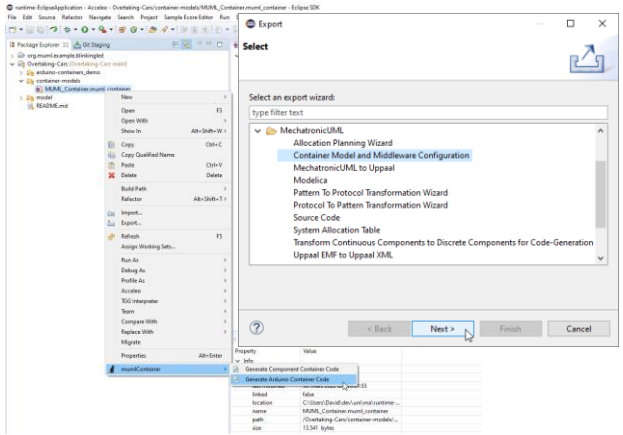
[1]

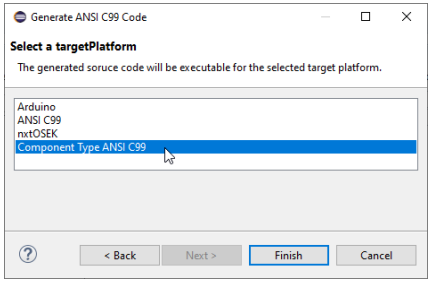




[3]

Manuelle Generierung





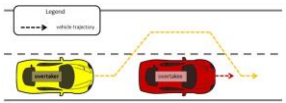
[3]

Rohe unvollständige Code-Dateien

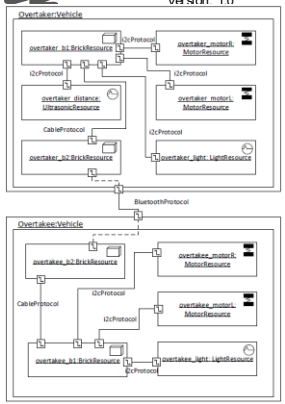
Motivation (2/4)

Problem (2/2)

[1]



[27]

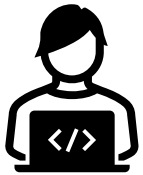


[1]

Nach einer Anleitung [Coda]

1. viele Dateien kopiert oder verschoben
2. Korrektur der Pfade in den #include-Anweisungen
3. Korrektur von „#include clock.h“
4. Nachtragen von Befehlen

...



Code-Dateien

Manueller Post-Processing-Ablauf [Reib]

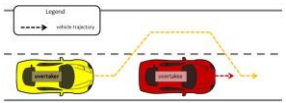
- Hoher Zeitbedarf
- Hohe Fehlerwahrscheinlichkeit

[29]

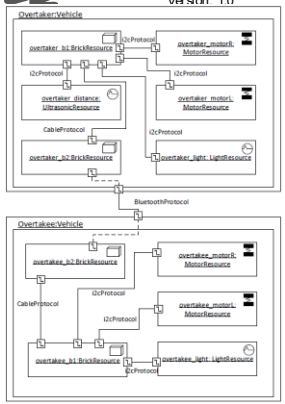
Motivation (3/4)

Ziel (1/2)

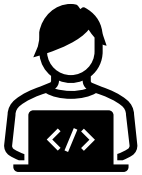
[1]



[27]



[1]



Automatische Generierung und Nachbearbeitung

- Zuverlässig
- Schnell
- Flexibel

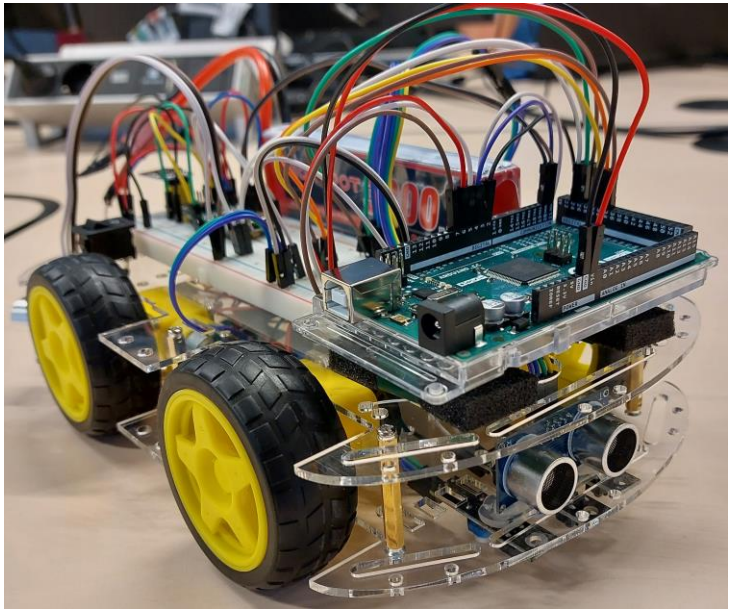
Code-Dateien

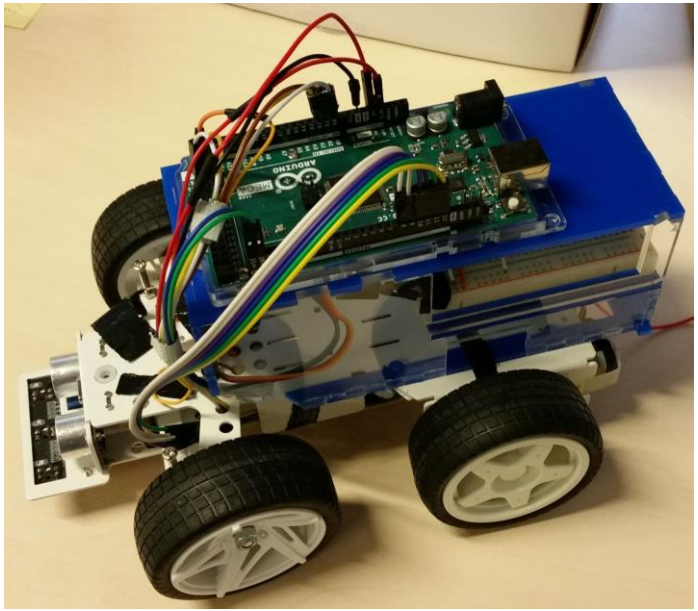
[29]

Motivation (3/4) (Löschen)

Ziel (2/2)

- Anpassung an aktuelle Roboterautos und Bibliothek „Sofdcar-HAL“

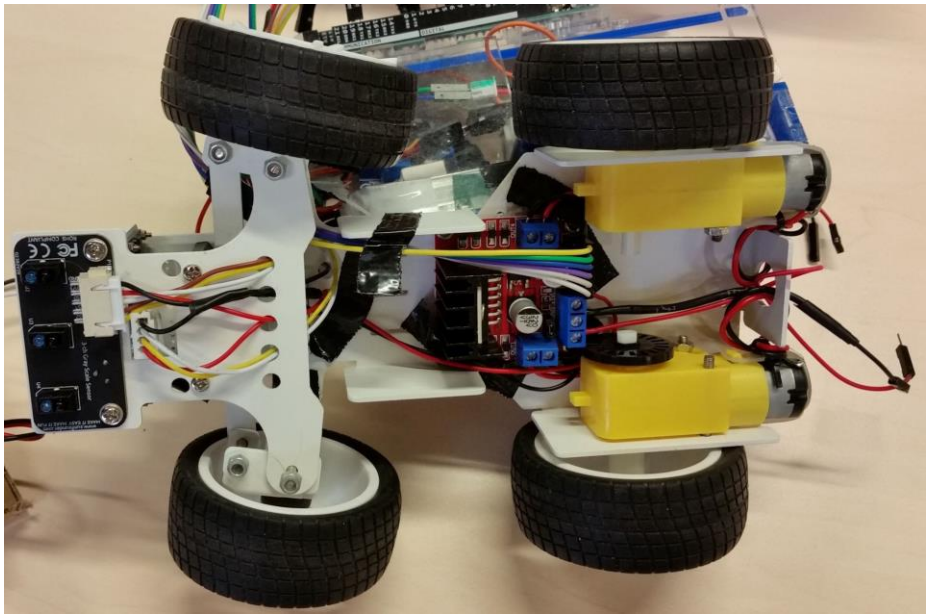


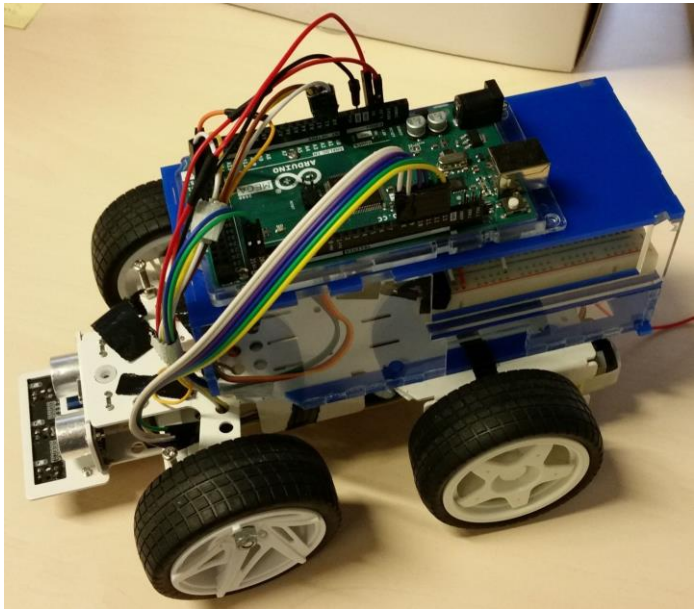


Motivation (3/4) (Löschen)

Ziel (2/2)

- Anpassung an aktuelle Roboterautos und Bibliothek „Sofdcar-HAL“

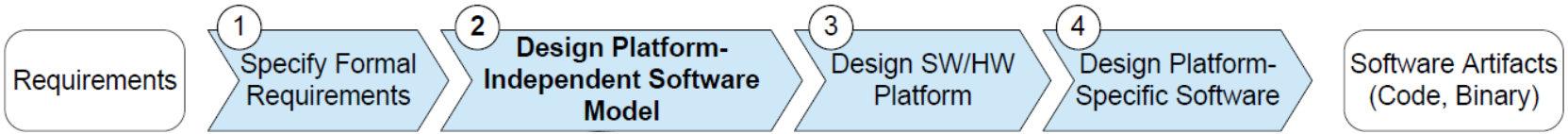




Lösungsansatz

- Analysen
- Anpassung an aktuelle Roboterautos und Bibliothek „Sofdcar-HAL“
- Konfigurierbare Pipeline

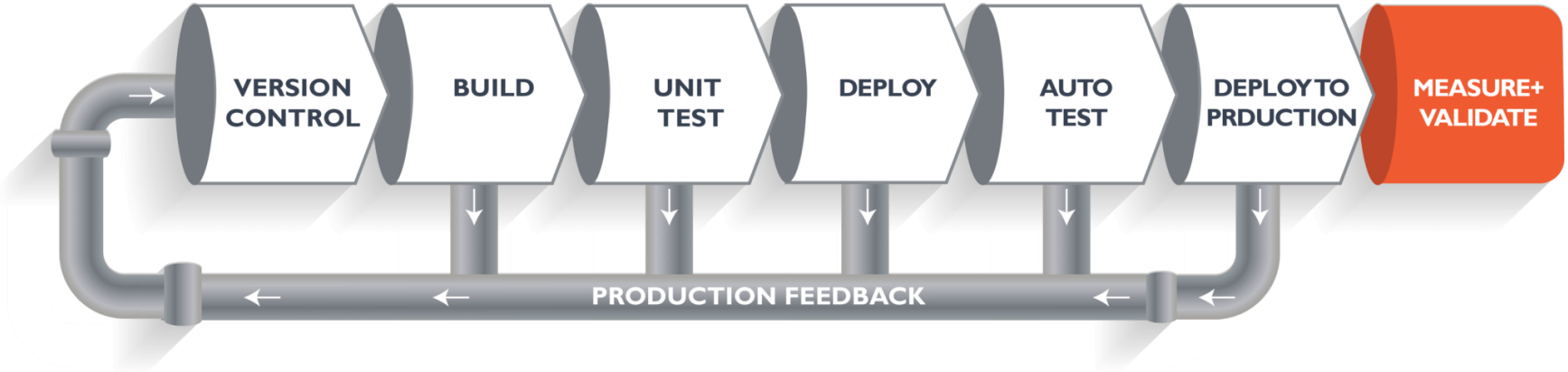
Grundlagen (1/3)
MechatronicUML



[7] (zugeschnitten)

Grundlagen (3/3)

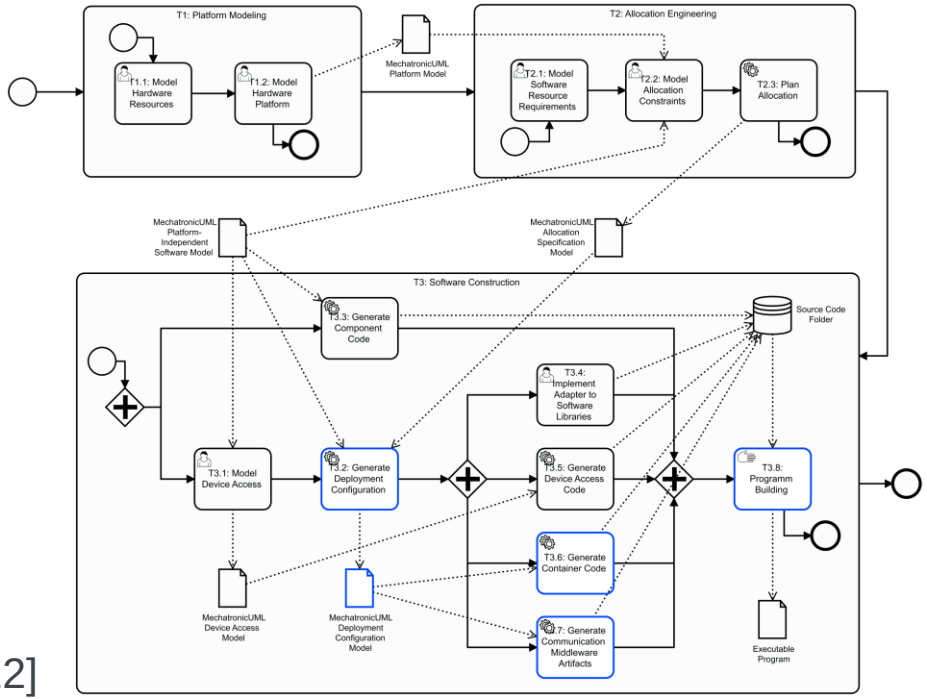
Kontinuierliche Integration/Kontinuierliches Deployment (Continuous Integration/Continuous Deployment)



[6]

Verwandte Arbeiten (1/5)

MechatronicUML mit Stürners Erweiterung



[12]

Verwandte Arbeiten (3/5)

Automatisierungsansätze für Eclipse

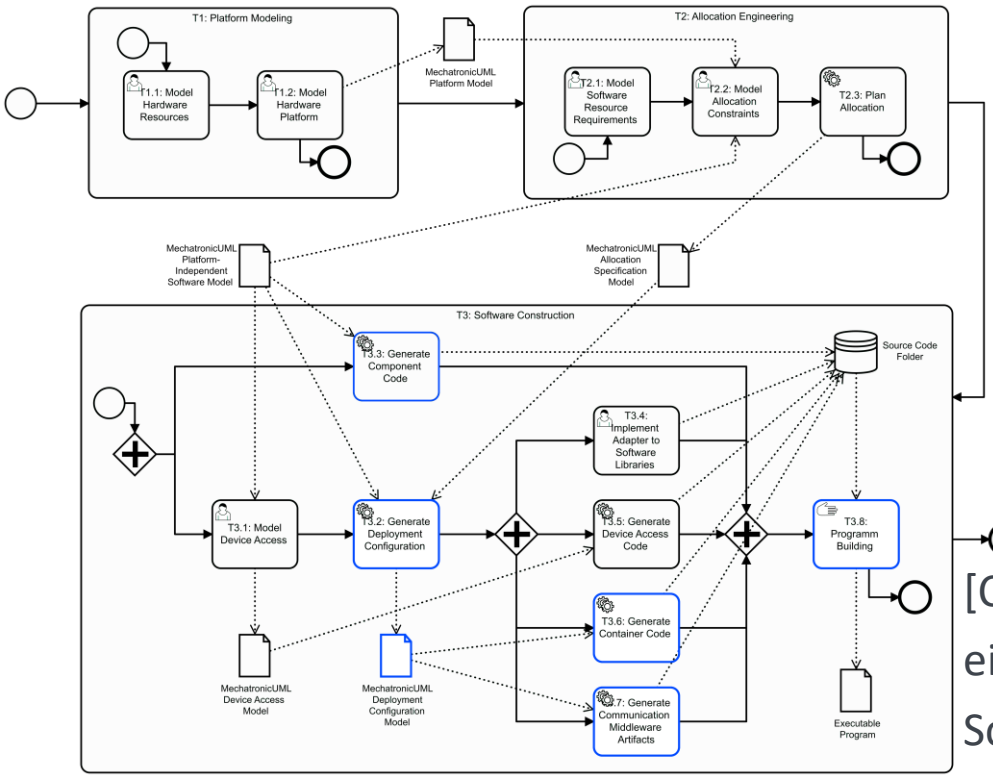
- Versuchte Automatisierungsansätze für Eclipse:
  - EASE
  - TEA
  - Apache Ant
  - Apache Maven
  - Gradle
- Keiner davon wurde verwendet

Verwandte Arbeiten (?/?)

CI/CD-Pipeline

- Nach Stephen J. Bigelow [Big]:
  - Geschwindigkeit
  - Konsistenz
  - Enge Versionskontrolle
  - Automatisierung
  - Integrierte Rückmeldungsschleifen
  - Durchgängige Sicherheit (wird nicht verwendet)

Analyse des bisherigen Arbeitsablaufes (Löschen)



[Coda], aber mit einer Änderung: Der Schritt T3.3 wurde blau markiert.

Analyse des bisherigen Arbeitsablaufes (Löschen)

- Arbeitsschritt 1: T3.2: „Deployment Configuration aka Container Transformation“[Coda], also Container-Transformation
  - Ablauf: Aufrufen der Export-Operation „MeachatronicUML“ /„Container Model and Middleware Configuration“ und Eintragen der Einstellungen wie z.B. Zielort
  - Verwendet:
    - Einstellungen für Middleware (hier „MQTT and I2C Middleware Configuration“) und Zielort
    - Datei „roboCar.muml“
  - ↓

Analyse des bisherigen Arbeitsablaufes (Löschen)

- Arbeitsschritt 1: T3.2: „Deployment Configuration aka Container Transformation“[Coda], also Container-Transformation
  - ↓
  - Generiert/Ergebnis: Datei „MUML_Container.muml_container“
  - Verwendete Komponenten: Interner Start per ContainerWizard aus MUML-Plug-In-Projekt „PlatformSpecificModeling“ bzw. Unterprojekt „org.muml.psm.container.transformation“ verantwortlich.

Analyse des bisherigen Arbeitsablaufes (Löschen)

- Arbeitsschritt 2: T3.6 und T3.7: „Container Code Generation“[Coda], also Generation des Containercodes
  - Ablauf: [Rechtsklick auf „MUML_Container.muml_container“ Datei]/ „mumlContainer“/ „Generate Arduino Container Code“
  - Verwendet: „MUML_Container.muml_container“-Datei
  - ↓

Analyse des bisherigen Arbeitsablaufes (Löschen)

- Arbeitsschritt 2: T3.6 und T3.7: „Container Code Generation“[Coda], also Generation des Containercodes
  - ↓
  - Generiert/Ergebnis: Ordner „APImappings“ mit unvollständigen Codes für APIs und Mikrokontroller
  - Verwendete Komponenten:
    - einige interne Bibliotheken, z.B. „I2CCustomLib“
    - Generierungsstart über GenerateAll aus MUML-Plug-In-Projekt „mechatronicuml-cadapter-component-container“ bzw. Unterprojekt „org.muml.codegen.componenttype.export.ui“

Analyse des bisherigen Arbeitsablaufes (Löschen)

- Arbeitsschritt 3: T3.3: „Component Code Generation“[Coda], also Generation der Komponentencodes
  - Ablauf: Aufrufen der Export-Operation „MeachatronicUML“/„Source Code_workspace“ und Eintragen der Einstellungen
  - Verwendet:
    - Einstellungen für die Zielplattform (hier „Component Type ANSI C99“) und einem Zielort
    - Datei „roboCar.muml“
  - ↓

Analyse des bisherigen Arbeitsablaufes (Löschen)

- Arbeitsschritt 3: T3.3: „Component Code Generation“[Coda], also Generation der Komponentencodes
  - ↓
  - Generiert/Ergebnis: Komponentencodes
  - Verwendete Komponenten:
    - Internes Starten per Klasse C99SourceCodeExport und deren Methode generateSourceCode(EObject element, IContainer targetFolder, IProgressMonitor monitor) aus MUML-Plug-In-Projekt „mechatronicuml-cadapter-component-container“ bzw. Unterprojekt „org.muml.c.adapter.componenttype.ui“.

Analyse des bisherigen Arbeitsablaufes (Löschen)

- Arbeitsschritt 4: T3.8: „Program Building“[Coda]
  - Ablauf:
    - Post-Processing-Teil nach Stürner [Stü22] wird gänzlich von Hand durchgeführt:
      - viele Dateien werden kopiert oder verschoben
        - In manchen Dateien Korrektur der Pfade in den #include-Anweisungen und des Einbindens von „clock.h“
        - In anderen Dateien Nachtragen des entsprechenden Befehls
      - ↓

Analyse des bisherigen Arbeitsablaufes (Löschen)

- Arbeitsschritt 4: T3.8: „Program Building“[Coda]
  - ↓
  - Nachtragen von Konfigurationen
    - Z.B. die von dem jeweiligen Roboterauto zu verwendende Geschwindigkeit
    - Anmeldeinformationen des zu verwendenden WLAN-Netzwerks und für die
    - Verbindungsinformationen mit dem zu nutzenden MQTT-Server
    - Bei der neueren Version der Roboterautos musste die Nutzung der „Sofdcar-HAL“-Bibliothek nachgetragen
  - langer und gänzlich manueller Ablauf
- ↓

Analyse des bisherigen Arbeitsablaufes (Löschen)

- Arbeitsschritt 4: T3.8: „Program Building“[Coda]
  - ↓
  - Upload-Teil:
    - Für jeden Sketch:
      - Öffnen mit der Arduino-IDE
      - Verbinden mit dem entsprechenden Mikrokontroller
  - ↓

Analyse des bisherigen Arbeitsablaufes (Löschen)

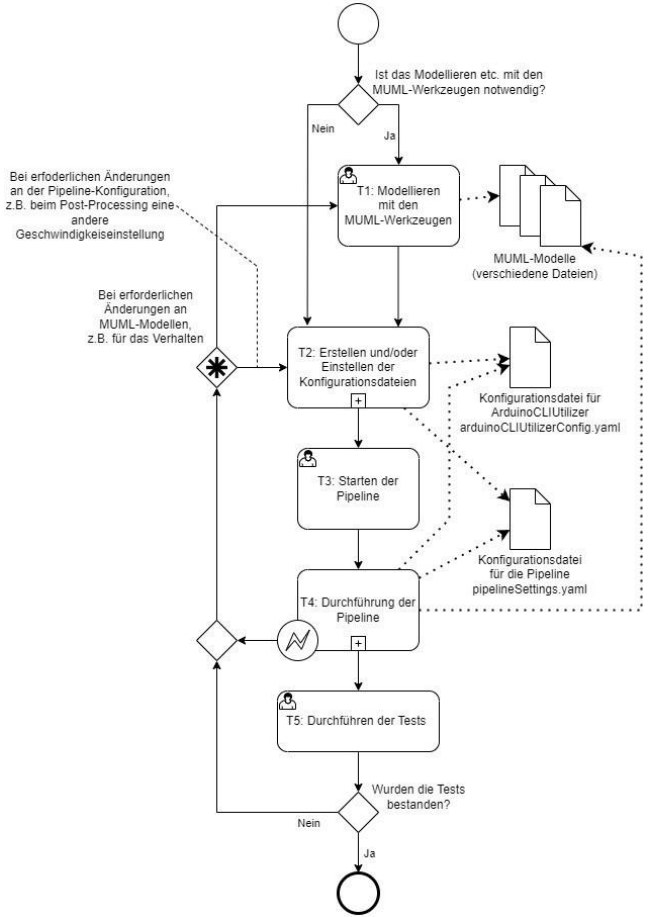
- Arbeitsschritt 4: T3.8: „Program Building“[Coda]
  - ↓
  - Verwendet: Generierter unvollständiger Code und Roboterautos bzw. deren Mikrokontroller
  - Generiert/Ergebnis: Nutzungsbereiter vollständiger Arduinocode/Sketch, der kompiliert wurde und auf die Mikrokontroller hochgeladen wurde
  - Verwendete Komponenten: Keine

Entwurf (?/?)

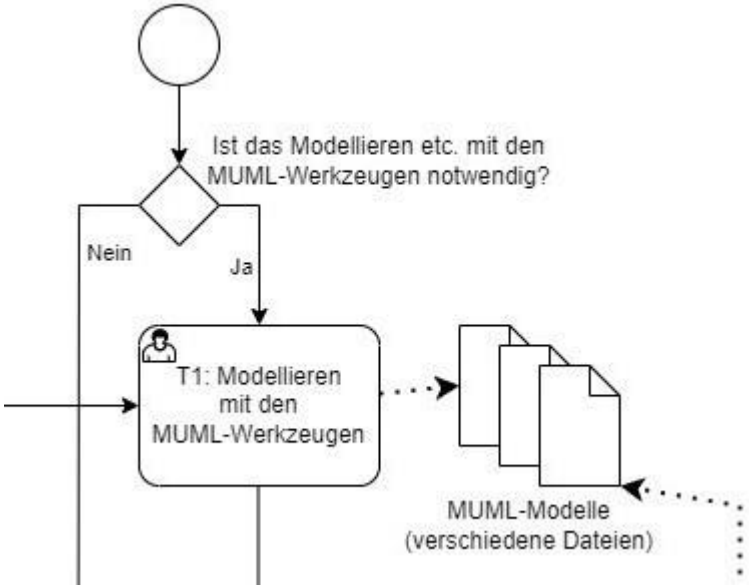
MUML: Automatisierung der Handgriffe durch Plug-Ins

- Automatisierung der Schritte und Handgriffe als Eclipse-Plug-Ins:
  - MUML-Werkzeuge für rohen generierten Code
  - Post-Processing-Ablauf
  - Kompilieren und Upload per Arduino-Software
- Einstellmöglichkeiten per Konfigurationsdatei

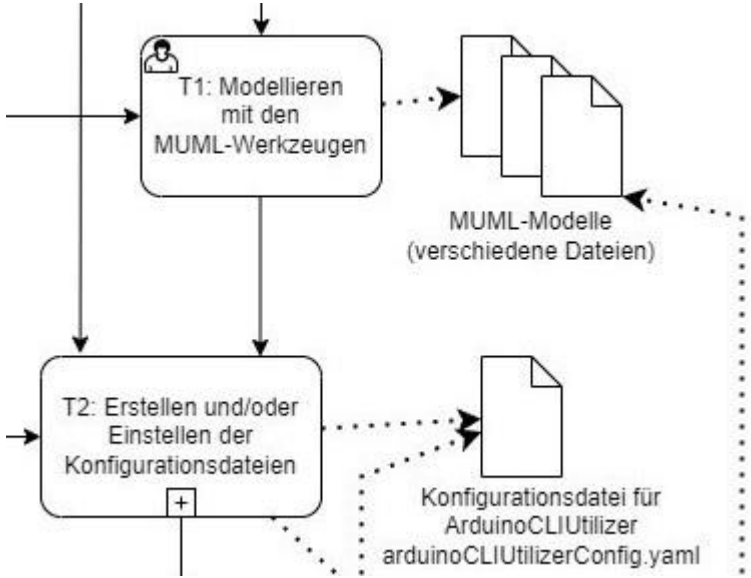
Entwurf
Nutzungsablauf



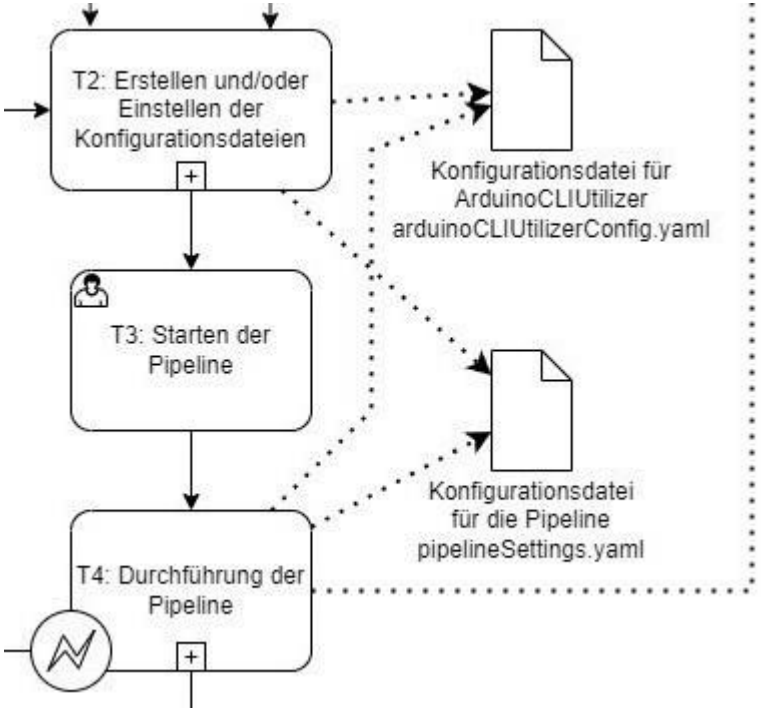
Entwurf
Nutzungsablauf



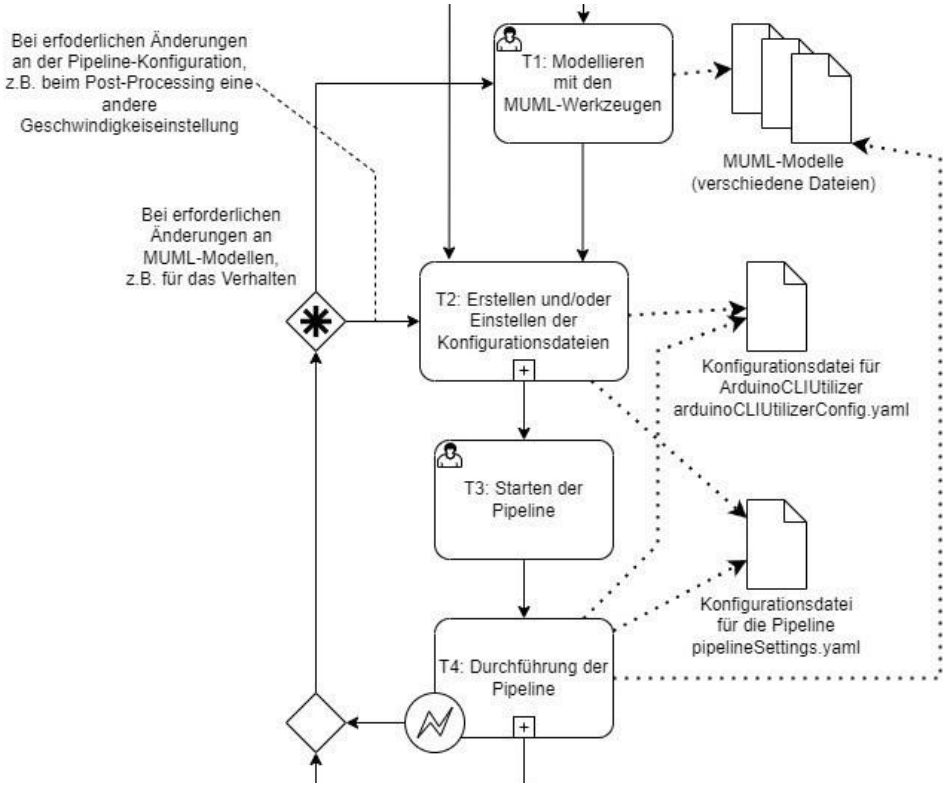
Entwurf
Nutzungsablauf



Entwurf
Nutzungsablauf

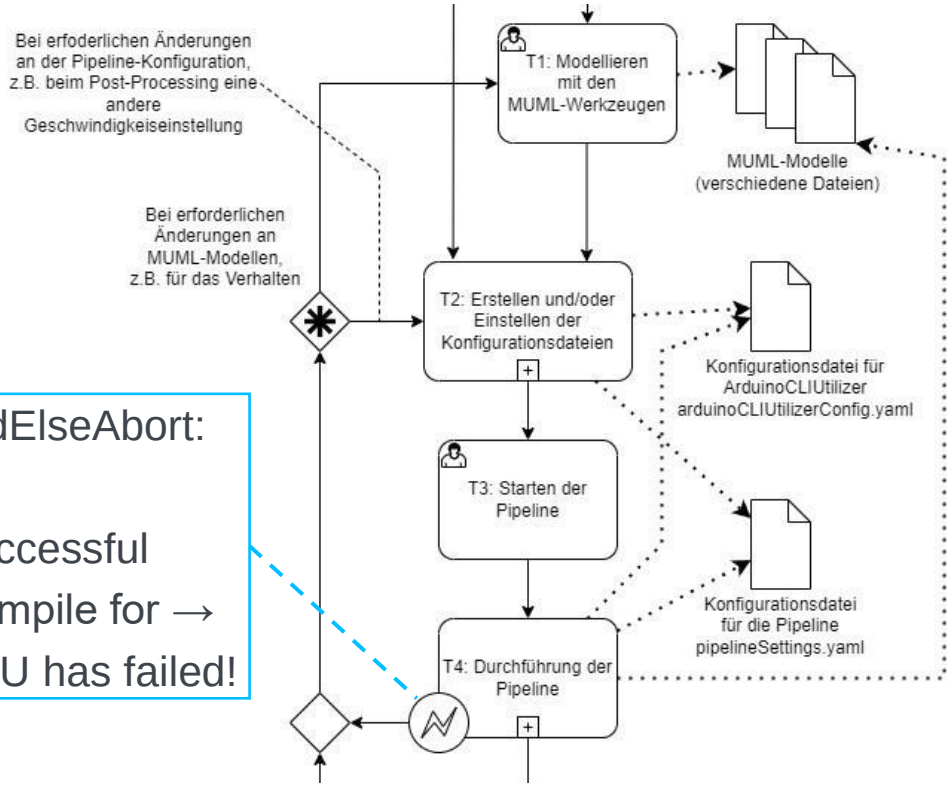


Entwurf
Nutzungsablauf



Entwurf

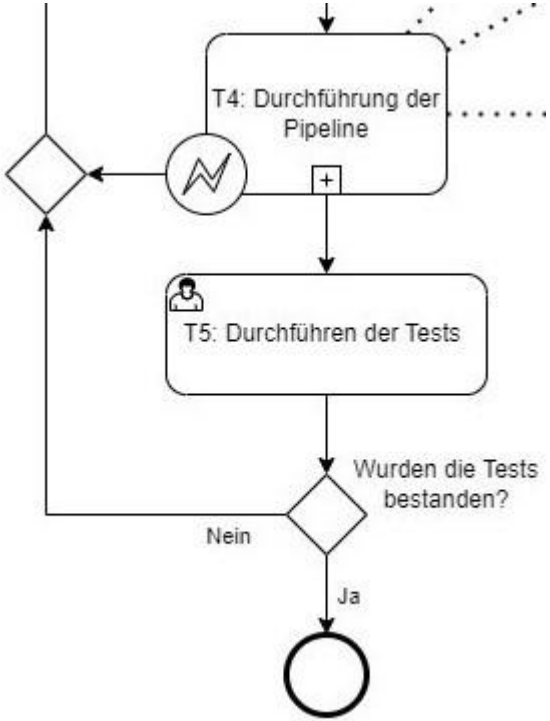
Nutzungsablauf



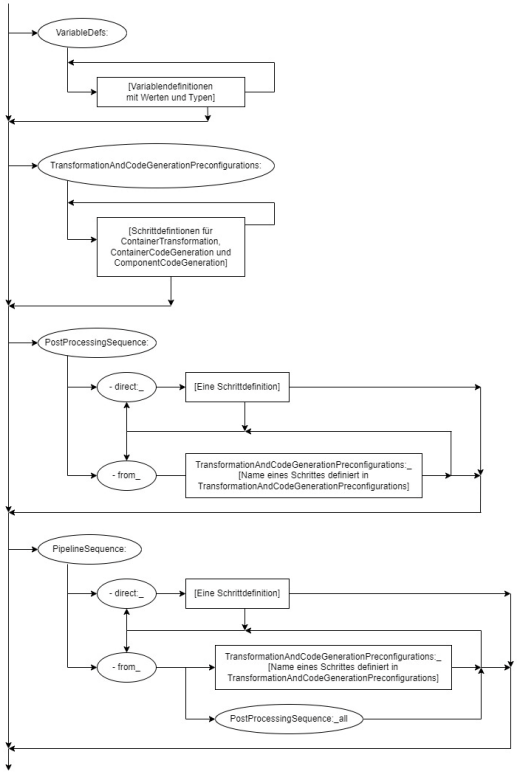
Beispiel: Abbruch bei erfolglosem Schritt

```
OnlyContinueIfFulfilledElseAbort:
  in:
    condition: from ifSuccessful
    message: direct Compile for → fastCarCoordinatorECU has failed!
```

Entwurf
Nutzungsablauf



Entwurf
Übersicht Aufbau Pipeline



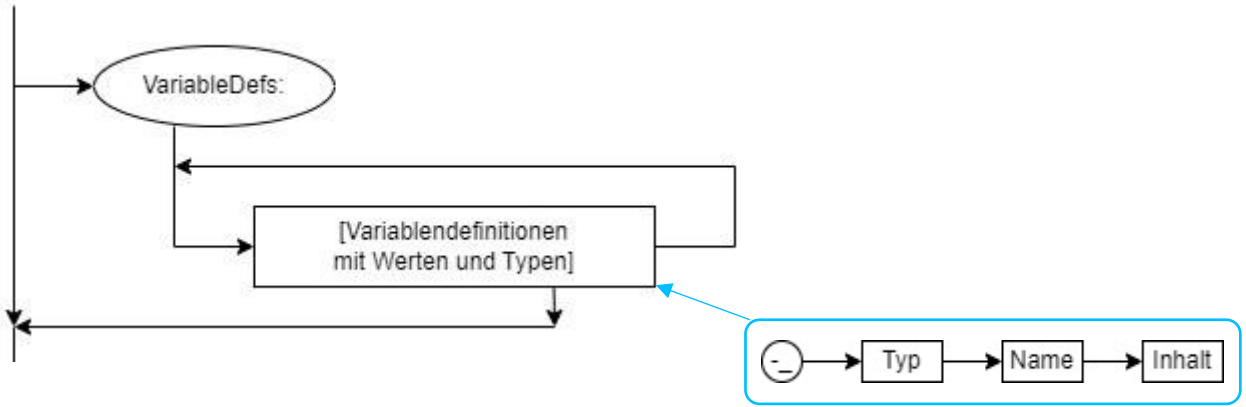
INFO

INFO

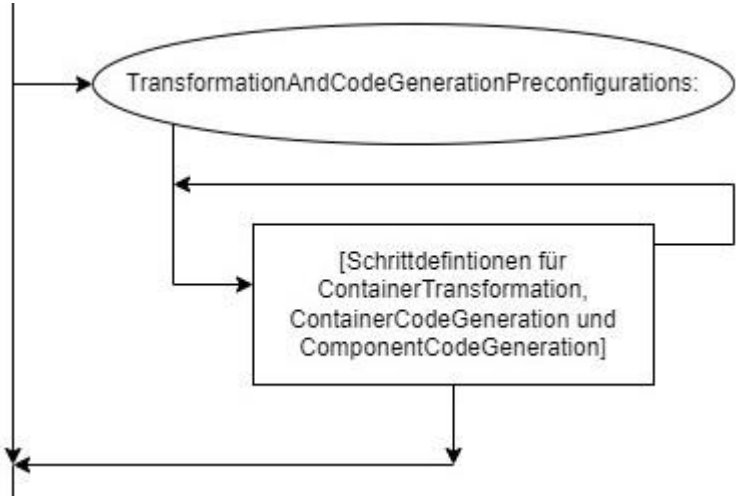
- Schrittbeispiele wie in der Zwischenpräsentation kommen wahrscheinlich später noch.!

Entwurf

Übersicht Aufbau Pipeline

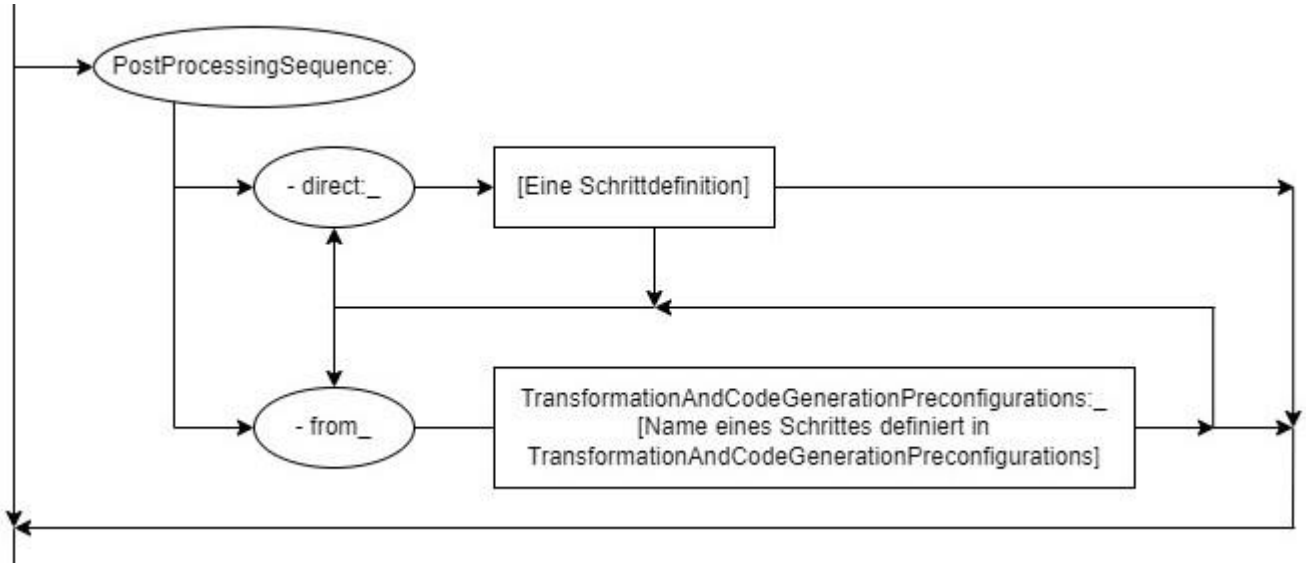


Entwurf
Übersicht Aufbau Pipeline

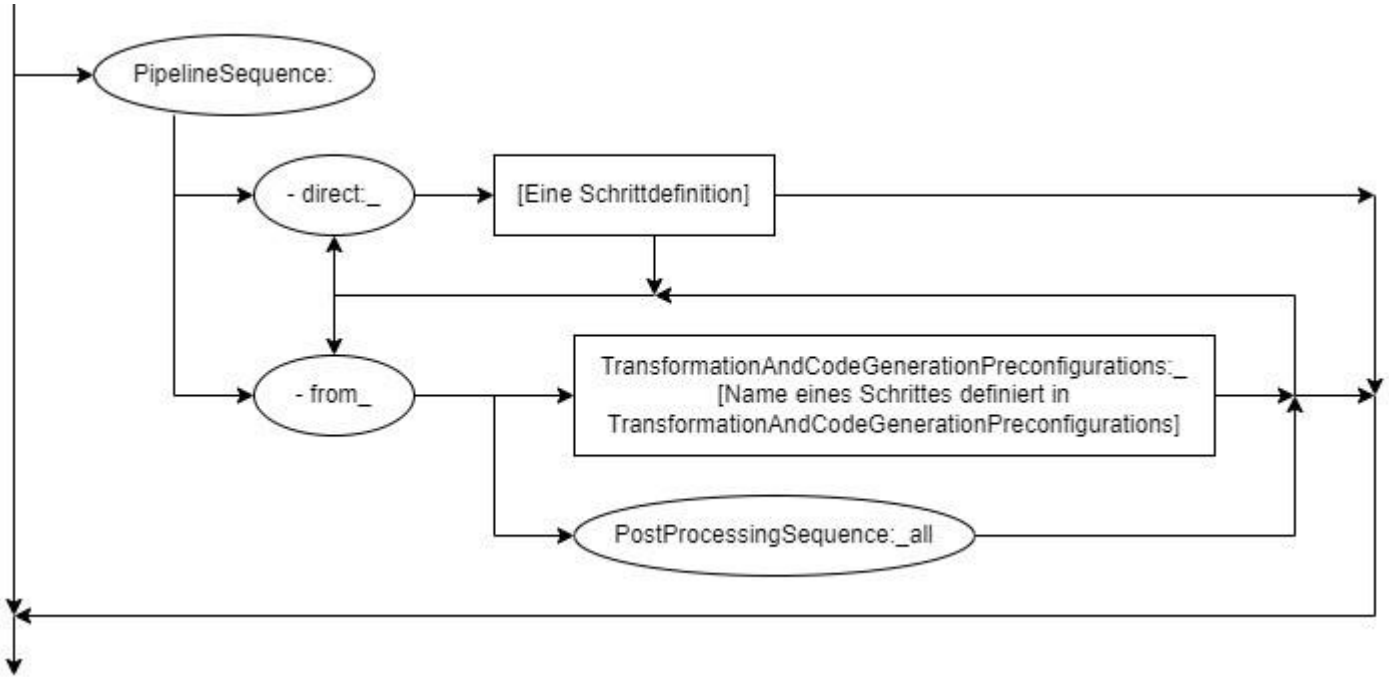


Entwurf

Übersicht Aufbau Pipeline

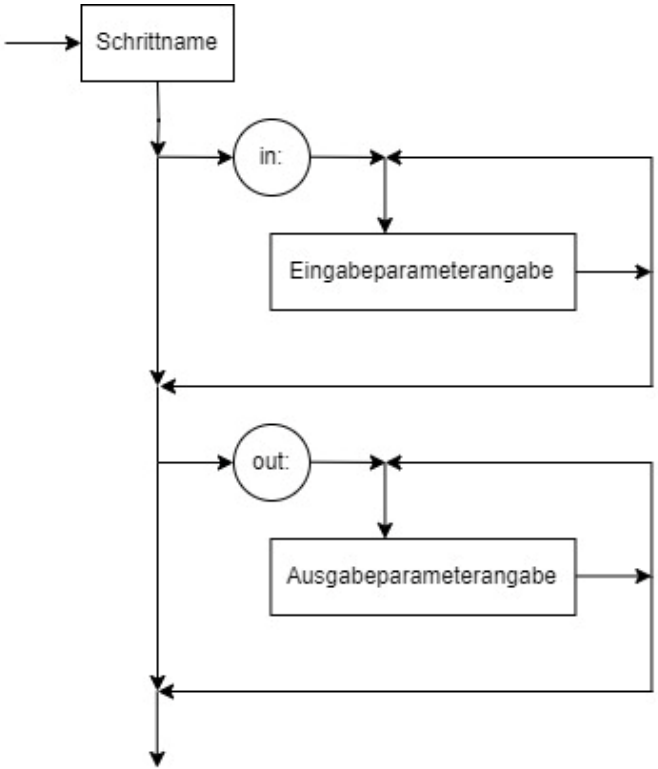


Entwurf
Übersicht Aufbau Pipeline



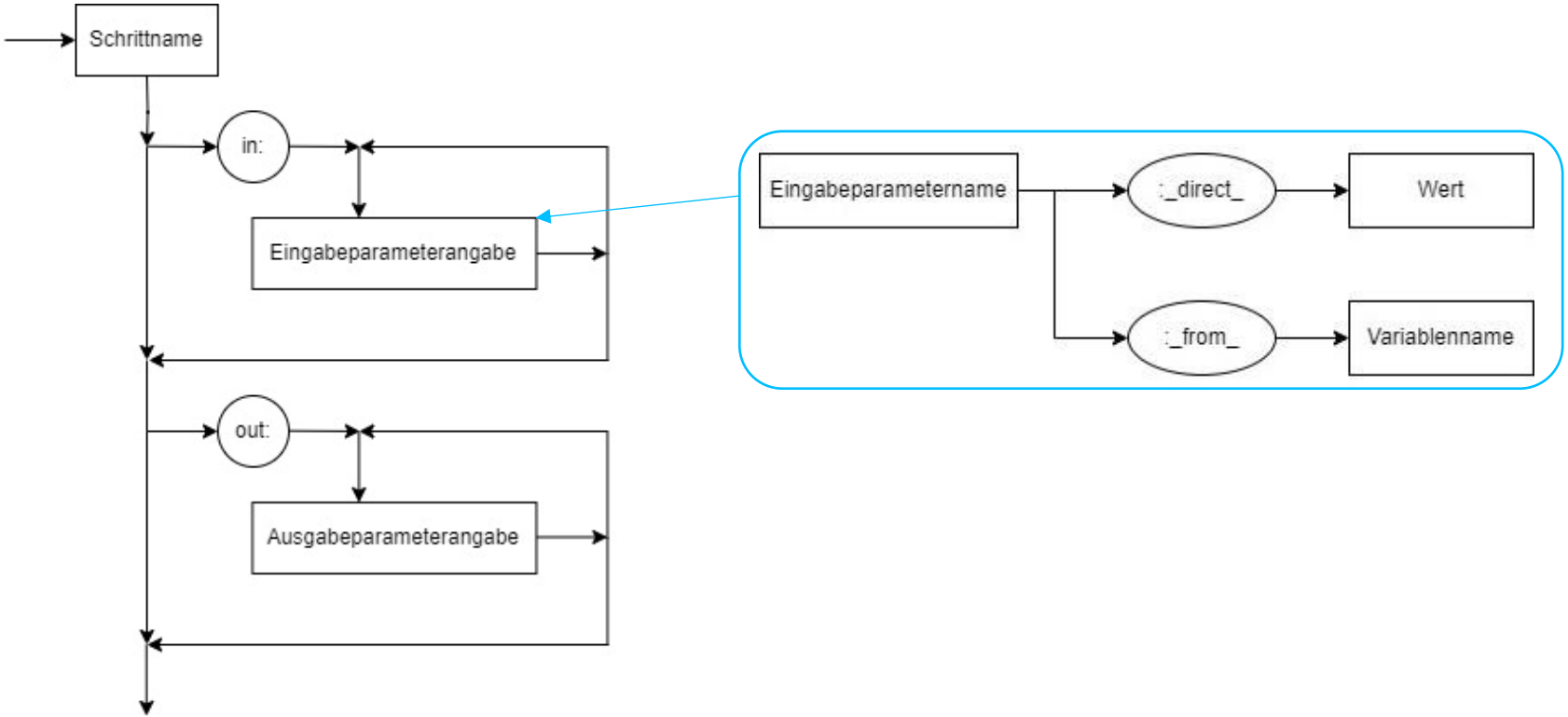
Entwurf

Aufbau Pipelineschritteinträge



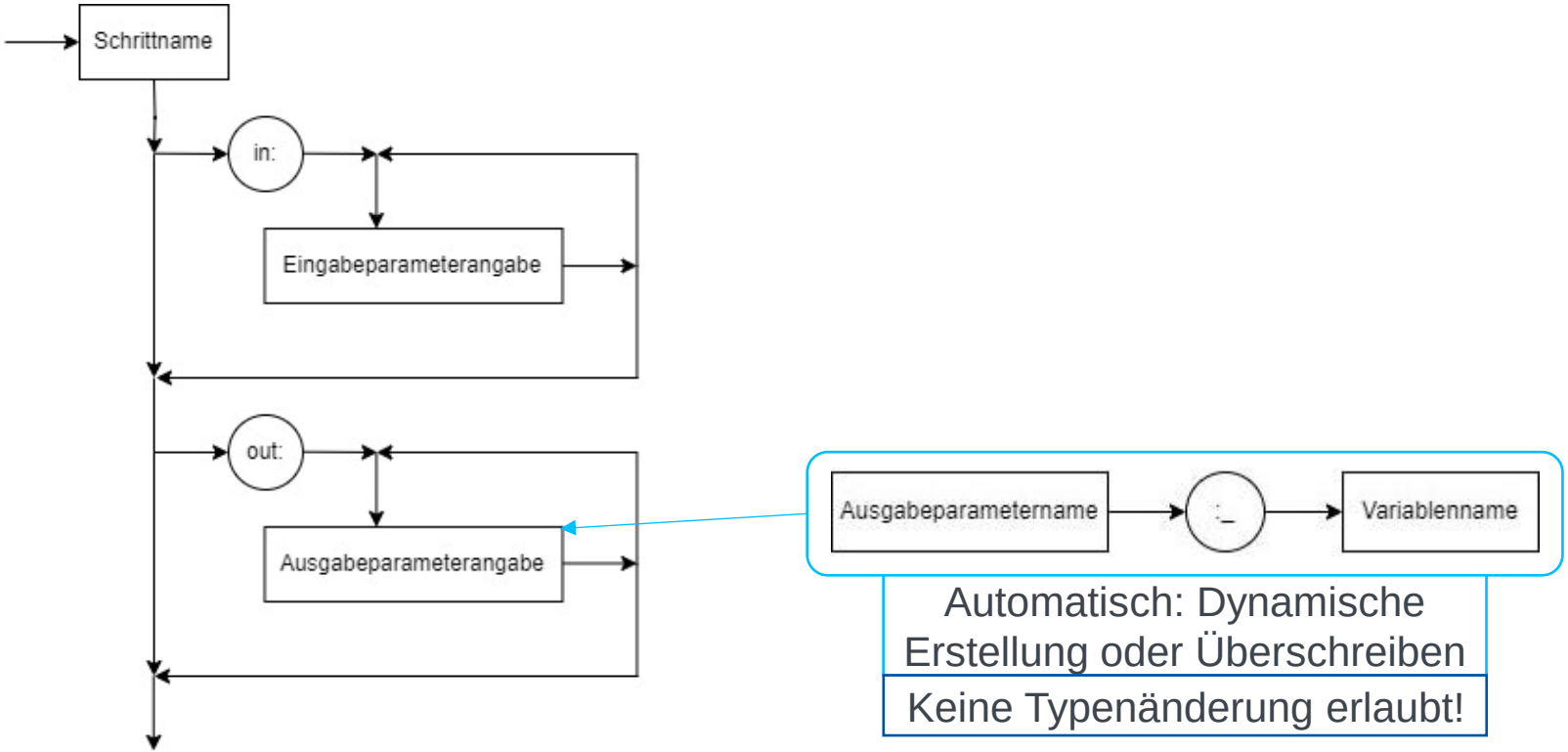
Entwurf

Aufbau Pipelineschritteinträge



Entwurf

Aufbau Pipelineschritteinträge



Umsetzung 1: Volle Unterstützung von neuer oder geänderter Hardware, Software und Bibliotheken (Löschen?)

Modelländerungen

- Ansätze
  1. Beibehalten der alten Architektur und Ersetzen der alten Komponenten mit entsprechenden neuen Komponenten aus Sofdcar-HAL.
     1. Architekturänderung für das Richtungslenken.
     2. Durchbrechen der Modularisierung

  ↓

Umsetzung 1: Volle Unterstützung von neuer oder geänderter Hardware, Software und Bibliotheken (Löschen?)

Modelländerungen

- Ansätze

  ↓

  2. Priorisieren der Modularisierung, d.h. Ersetzen der alten Antriebskomponente mit Kindklasse von DriveController
     1. Nutzen von Sofdcar-HAL wie beabsichtigt
     2. Verwechslungsvermeidung: Umbenennung von DriveControl zu CourseControl
        1. DriveControl/CourseControl nur noch für das Festlegen des Kurses auf Fahrbahnen
- Ansatz 2 gewählt

Umsetzung 1: Volle Unterstützung von neuer oder geänderter Hardware, Software und Bibliotheken

Modelländerungen, wichtigstes:

1. Umbenennungen von Komponenten und Modellen in verschiedenen Diagrammen:
   - PowerTrain → DriveController und DriveControl → CourseControl
2. Ports für die Winkel-Werte: Bei DriveController und CourseControl hinzugefügt
3. Hinzufügen des Lenk-Servos + Ports + Verbindungen

↓

Umsetzung 1: Volle Unterstützung von neuer oder geänderter Hardware, Software und Bibliotheken

Modelländerungen, wichtigstes:

↓

4. Entfernen überschüssiger Motoren, d.h. nur noch ein Motor namens „fixedMotor“
5. Anpassen der APIs in „roboCarLibraries.osdsl“ für Sofdcar-HAL
6. Anpassung der Allokationen: Coallokationsdefinitionen PowerTrain und DriveControl entsprechend mit DriveController und CourseControl ersetzt

Umsetzung 1: Volle Unterstützung von neuer oder geänderter Hardware, Software und Bibliotheken (Löschen?)

Kommunikationsänderung innerhalb von Roboterautos von I2C auf Seriell

- Projekt “org.muml.arduino.adapter.container”,
  Ordner „resources/container_lib “,
  Dateien „I2cCustomLib.hpp“ und „I2cCustomLib.cpp“:
  - Enthielten Code von Stürner [12] für die I2C-Kommunikation
  - Wurden durch Code für die Serielle Kommunikation ersetzt:
    - Dateien „SerialCustomLib.hpp“ und „SerialCustomLib.cpp“
    - Alle Bezüge und Kommentare wurden angepasst

Umsetzung 1: Volle Unterstützung von neuer oder geänderter Hardware, Software und Bibliotheken

Änderungen am Post-Processing-Ablauf

- Änderungen an den Modellen und Ressourcen änderten die generierten unvollständigen Codeteile:
  - Anders benannte oder zusätzliche Dateien
  - Leicht andere Dateiinhalte

↓

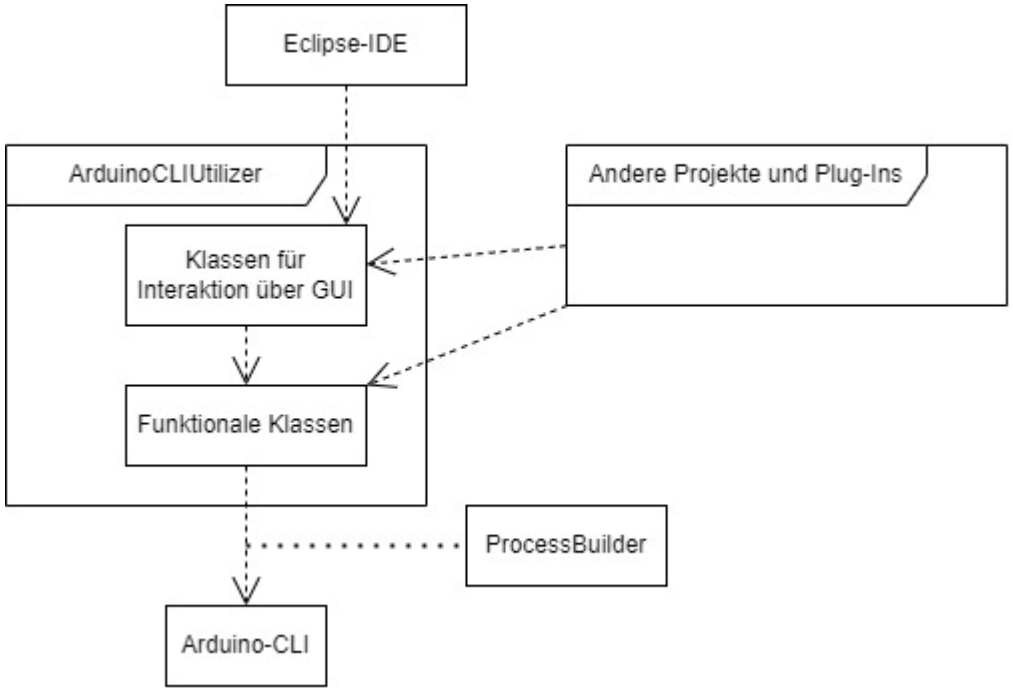
Umsetzung 1: Volle Unterstützung von neuer oder geänderter Hardware, Software und Bibliotheken

Änderungen am Post-Processing-Ablauf

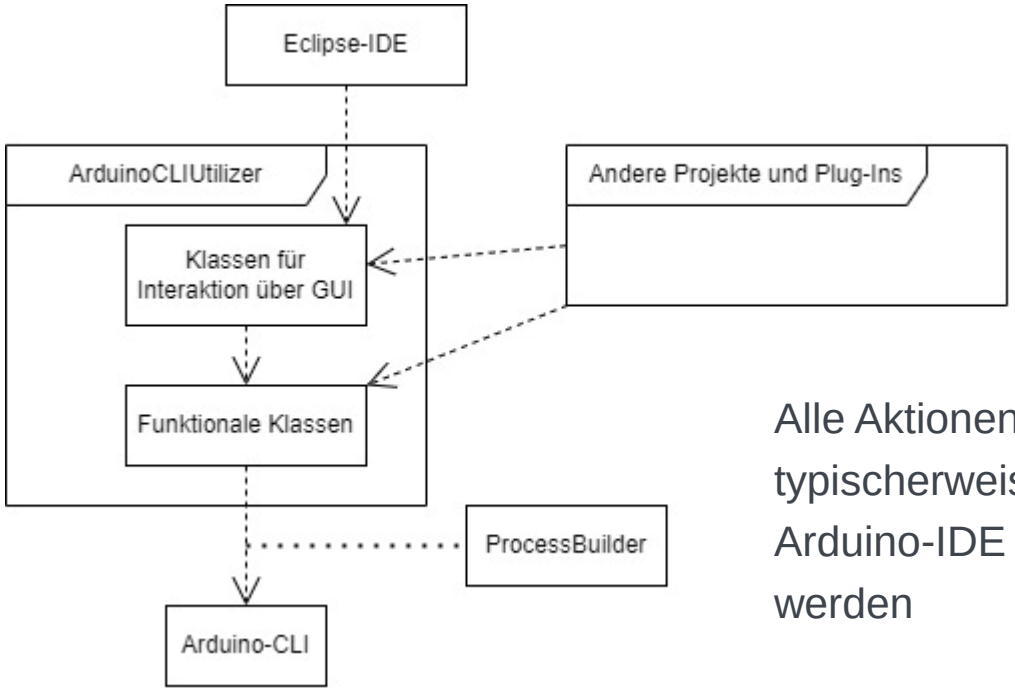
↓

- Anpassungen an manchen Arbeitsschritten
  - Berücksichtigung anders benannter Dateien
    - Z.B. driveControl -> courseControl
  - Weitere auszufüllende und nachzubearbeitende API-Dateien
    - Z.B. CI_DRIVECONTROLLERFDRIVECONTROLLERanglePortaccessCommand.c, in der *angle = SimpleHardwareController_DriveController_GetAngle(); nachgetragen wird

Umsetzung 2: Integration per Arduino-CLI



Umsetzung 2: Integration per Arduino-CLI

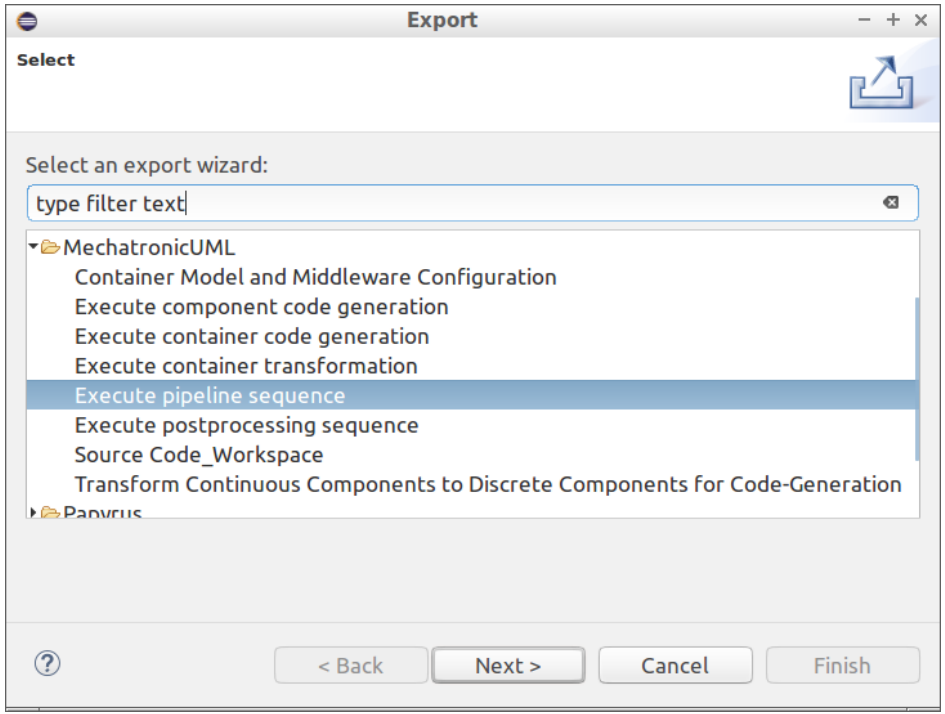


Alle Aktionen, die typischerweise mit der Arduino-IDE durchgeführt werden

Umsetzung 3 - Automatisierung per Pipeline

Starten

- Export-Wizard als Workaround



Umsetzung 3 - Automatisierung per Pipeline

Ausführung

- Lese-/Interpretations-Reihenfolge:
  1. VariableDefs
  2. TransformationAndCodeGenerationPreconfigurations
  3. PostProcessingSequence
  4. PipelineSequence

Umsetzung 3 - Automatisierung per Pipeline
Ausführung

- Instanz der PipelineSettingsReader-Klasse enthält die interpretierte Konfigurationsdatei
  - Zugriff auf TransformationAndCodeGenerationPreconfigurations per Name
    - IsEntryInTransformationAndCodeGenerationPreconfigurations(String name)
    - getTransformationAndCodeGenerationPreconfigurationsDef(String stepName)
  - Zugriff auf PostProcessingSequence und PipelineSequence ähnlich wie Listen gestaltet
    - Für Pipeline:
      - hasNextPipelineSequenceStep()
      - getNextPipelineSequenceStep()
      - resetPipelineSequenceProgress()
    - Für Post-Processing analog

Umsetzung 3 - Automatisierung per Pipeline (Löschen)
Fehlerprüfung

- Breit aufgestellte Fehlersuche
  - Teils beim Einlesen/Interpretieren (Fehler beim Einlesen und Interpretation)
  - teils Explizit per Funktion (sonstige Fehler, z.B. Typenänderungen)

Umsetzung 3 - Automatisierung per Pipeline
Fehlerprüfung

- Kurze Zusammenfassung:
  - Fehlende Konfigurationsdateien
  - Aufbau-Fehler
  - Werteangaben, die gegen die YAML-Standards verstoßen
  - Umgangsfehler bei Variablen und deren Typen
  - Ungültige Einbindeversuche von Pipelineschritten
    - (z.B. Verwendung von nicht vorhandenen Einträgen aus TransformationAndCodeGenerationPreconfigurations)

Umsetzung 3 - Automatisierung per Pipeline (Löschen)

Export-Änderung an MUML-Plug-In „mechatronicuml-cadapter-component-container“

- Klasse GenerateAll aus dem Package „org.muml.arduino.adapter.container.ui.common“ aus dem MUML-Plug-In-Projekt „mechatronicuml-cadapter-component-container“ bzw. dessen Unterprojekt „org.muml.codegen.componenttype.export.ui“ wird benötigt
- Änderung in Export-Einstellungen von „org.muml.codegen.componenttype.export.ui“:
  - Eintragen von diesem Package

Evaluation
Vorgehensweisen

- MUML-Modellanpassungen
  - Grobe Vergleiche zwischen alten und neuen generierten Dateien sowie den jeweils erwarteten Unterschieden

Evaluation
Vorgehensweisen

- Post-Processing-Änderungen
  - Alte und neue nachbearbeitete Codedateien sowie jeweils erwartete Unterschiede miteinander abgeglichen

Evaluation (Löschen?)

Vorgehensweisen

- Änderungen an den Modellautos
  - Manuell geprüft, ob Hardware wie gewünscht arbeitet
  - Geänderte Kommunikation wurde getestet, indem an den Datenleitungen, die den Koordinator mit dem Fahrer verbinden, ein weiterer Mikrokontroller angeschlossen wurde, der darauf programmiert wurde, die Kommunikation mit aufzunehmen und an den tragbaren PC weiterzuleiten.
  - Prüfen der Zuverlässigkeit der WiFi-Module teils auf ähnlichem Weg und teils auf Basis bereits vorhandener serieller Debug-Nachrichten

Evaluation
Vorgehensweisen

- ArduinoCLIUtilizer
  - Bei jeder entwickelten Klasse: Zunächst Debug-Ausgaben zur Beobachtung der dynamischen Ausführung
  - Prüfung, ob Arduino-CLI gleiche Ergebnisse wie Arduino-IDE liefert:
    - Beispielprogramme jeweils zweimal auf einen Arduino Uno hochgeladen
      - 1. mal per Arduino-IDE und 2. mal per Arduino-CLI
    - Jedes Mal wurde Verhalten beobachtet und mit verwendetem Testcode verglichen
    - Danach Analyse auf Verhaltensunterschiede zwischen den beiden Methoden
    - Später Tests mit Arduino MEGA 2560 und Nano für das Beispielszenario

Evaluation
Vorgehensweisen

- MUMLACGPPA
  - Wo möglich: Automatisierte Tests per Junit
  - Ansonsten: Manuelle Tests per Debug-Ausgaben und Untersuchung der Ergebnisse

Evaluation
Vorgehensweisen

- PipelineExecution
  - Jede Komponente manuell getestet
  - Interne Abläufe per Debug-Ausgaben beobachtet
  - Ergebnisse wurden beobachtet
    - Hierbei gezielt entworfene Abläufe in der Pipelinekonfiguration genutzt

↓

Evaluation
Vorgehensweisen

- PipelineExecution

  ↓

  - Vollständige Verhaltenstests mit einer Pipeline für das Beispielszenario
    - Neben ausgeliehenen Arduino Nanos auch private Arduino Mega 2560-er verwendet
      - Abgesehen von fehlender Roboterauto-Hardware Durchführung von Nutzungsversuchen für Beispielszenario exakt nachgestellt

Evaluation

Vorgehensweisen

- Qualitätsbewertungen
  - Entwickelter Code: ISO25010-Standard

Evaluation

Vorgehensweisen

- Qualitätsbewertungen
  - Entwickelter Code: ISO25010-Standard

Evaluation
Vorgehensweisen

- Entwickelte CI/CD-Pipeline
  - Nach Stephen J. Bigelow [Big], Erinnerung:
    - Geschwindigkeit
    - Konsistenz
    - Enge Versionskontrolle
    - Automatisierung
    - Integrierte Rückmeldungsschleifen

Evaluation
Vorgehensweisen

- Beispielszenario “Kooperatives Überholen“
  - Schritt 1: CI/CD-Pipeline wird auf modifizierte MUML-Modelle angewandt, um nutzungsbereite Sketches zu erhalten
    - Hierfür wird die generierbare Pipelinekonfiguration geöffnet und angepasst:
      - Schritte 1.1 – 1.3: Eintragen aller Daten, z.B. WLAN
      - Schritt 1.4: Schritte für das Kompilieren und Hochladen wurden entfernt.

↓

Evaluation
Vorgehensweisen

↓

- Schritt 2: Sketches über die Arduino-IDE auf die entsprechenden AMKs hochgeladen
- Schritt 3: Danach Platzieren und Starten der beiden Roboterautos auf der Teststrecke, um ihr Verhalten beobachten zu können

Evaluation
Ergebnisse

- Erfüllungsstufen:
  - 3: Erfüllt alle beschriebenen Eigenschaften oder/und besitzt alle beschriebenen Fähigkeiten
  - 2: Beschriebene Eigenschaften zu einem hohen Anteil oder/und hoher Anteil der beschriebenen Fähigkeiten erfüllt
  - 1: Beschriebene Eigenschaften zu einem geringen Anteil oder/und geringer Anteil der beschriebenen Fähigkeiten erfüllt
  - 0: Keine der beschriebenen Eigenschaften oder/und Fähigkeiten erfüllt

Evaluation

Ergebnisse Softwarequalität

Richtlinie	Durchschnittliche Erfüllungsstufe
Funktionale Eignung	2
Kompatibilität – Koexistenz	3
Kompatibilität – Interoperabilität	1
Verwendbarkeit	2,33
Zuverlässigkeit	2,5
Wartbarkeit	2,25
Portabilität	2,5

- Bewertungen der Aspekte fast immer zusammengefasst und Durchschnitt eingetragen
- Portabilität – Ersetzbarkeit unklar

Evaluation

Ergebnisse CI/CD-Pipelinequalität

Eigenschaft oder Fähigkeit	Durchschnittliche Erfüllungsstufe
Geschwindigkeit	3
Konsistenz	3
Enge Versionskontrolle	1
Automatisierung	2
Integrierte Rückmeldungsschleifen	2

Evaluation
Ergebnisse Beispielszenario „Kooperatives Überholen“

- Beobachtungen:
  - Schritt 1:
    - Die Pipeline konnte komplett durchgeführt werden und schloss erfolgreich ab.
    - Vergleiche des erzeugten Codes zeigten nur die gewollten oder erwarteten Unterschiede
      - Code von Stürner [Codb]
      - manuell nachbearbeiteter Code [Reie]
  - Schritt 2:
    - Keine Zwischenfälle beim Hochladen auf die verschiedenen Mikrokontroller

↓

Evaluation

Ergebnisse Beispielszenario „Kooperatives Überholen“

↓

- Schritt 3:
  - Roboterautos
    - Stellten Verbindungen, z.B. WLAN, her
    - Beim Fahren
      - Gewünschte Geschwindigkeitsunterschiede
      - Befolgen der Fahrbahn
      - Kommunikation für Erlaubnis des Überholmanövers schwer auslösbar
        - Nicht verifizierter Teil von Stürners Ergebnissen

Evaluation
Diskussion

- INFO: Versuchen, mündlich zu integrieren !!!!

Schlussfolgerung
Vorteile

- Alle Vorgänge von den Modelltransformationen bis zum Hochladen auf Modellautos automatisch durchführbar.
  - gesamter Arbeitsablauf in weniger als einer Minute
  - menschliche Fehlerrate und mögliche Verwirrung vermieden
    - Vermeidung von Fehlern im Umgang mit Dateien oder hochzuladendem Code

↓

Schlussfolgerung

Vorteile

↓

- flexibler Aufbau der Pipeline und Schritteinträgen erleichtert Rekonfiguration oder Erweiterung
- Pipelineschritt TerminalCommand für Skripte oder Programme

Schlussfolgerung

Wichtigste Limitierungen und zukünftige Arbeiten

Limitierung	Zukünftige Arbeiten
Wegen MUML-Werkzeugen Export-Wizard-Workaround	Implementierung einer besseren Integration
Kaum Unterstützung der verschiedenen Versionsverwaltungssysteme	Implementierung von Schritten für diese
Keine Unterstützung von automatischen Tests	Integration von automatischen Tests, z.B. per Transformation von MUML- zu Matlab-Modellen (siehe Heinzemann et al. [HRB+14])
Keine unterschiedlichen Ausführungspfade	Verbesserung des Kontrollflusses

Zusammenfassung

- Ermöglichung der CI/CD-Methode per Pipeline
  - Analysen durchgeführt
  - Anpassungen vorgenommen
    - MUML-Modelle
    - Post-Processing

↓

Zusammenfassung

- Ermöglichung der CI/CD-Methode per Pipeline

  ↓

  - CI/CD-Pipeline
    - Wurde für Verwendbarkeit, Flexibilität und Konfigurierbarkeit entworfen
    - In Eclipse als Plug-In integriert
    - Kann Build-Prozess zuverlässig und automatisch durchführen
    - Hohe Zeitersparnis

Universität Stuttgart

Vielen Dank!

Sebastian Baumfalk

st114908@stud.uni-stuttgart.de

Institut für Software Engineering
Universitätsstraße 38

Quellen (1/6)

- [1] J. Bobolz, M. Czech, A. Dann, J. Geismann, M. Huwe, A. Krieger, G. Piskachev, D. Schubert, R. Wohlrab. Final Document. Tech. rep. Project Group Cyberton, Heinz Nixdorf Institute, University of Paderborn, 2014
- [2] G. Reißner. New Motor Driver (Hardware abstraction libarary). https://github.com/SQARobo-Lab/Sofdcar-HAL
- [3] https://github.com/SQA-Robo-Lab/MUML-CodeGen-Wiki/blob/main/user-documentation/main.md
- [4] G. Reißner. New Motor Driver (Hardware abstraction libarary). https://github.com/SQARobo-Lab/Sofdcar-HAL.
- [6] Samarpit Tuli. Learn How to Set Up a CI/CD Pipeline From Scratch. https://dzone.com/articles/learn-how-to-setup-a-cicd-pipeline-from-scratch

Quellen (2/6)

- [7] S. Dziwok, U. Pohlmann, G. Piskachev, D. Schubert, S. Thiele, C. Gerking. The MechatronicUML Design Method: Process and Language for Platform- Independent Modeling, Technical Report tr-ri-16-352. 2016
- [8] A. P. Dann, U. Pohlmann. The MechatronicUML Hardware Platform Description Method - Process and Language. Techn. Ber. tr-ri-14-336. v. 0.1. Heinz Nixdorf Institute, University of Paderborn, Feb. 2014
- [10] https://www.istockphoto.com/de/vektor/vektor-sat-turm-in-der-isometrischen-perspektive-isoliert-auf-wei%C3%9Fem-hintergrund-gm649545494-118161691?phrase=funkturm
- [11] https://www.istockphoto.com/de/vektor/router-symbol-auf-transparentem-hintergrund-gm1282351407-380110383

Quellen (3/6)

- [12] D. Sturner. „Generating Code for Distributed Deployments of Cyber-Physical Systems Using the MechatronicUML“. Magisterarb. Universitatsstrase 38, D-70569 Stuttgart: University of Stuttgart, Mai 2022
- [13] Arduino. Offizielle Dokumentation zu Arduino Mega 2560 Rev3. https://docs.arduino.cc/hardware/2560
- [14] Arduino. Offizielle Dokumentation zu Arduino Nano. https://docs.arduino.cc/hardware/nano
- [15] D. Bachfeld. Microcontroller flashen: Arduino Uno als In-System-Programmer. https://www.heise.de/hintergrund/Arduino-Uno-als-In-System-Programmer-2769246.html

Quellen (4/6)

- [16] W. Badawy, A. Ahmed, S. Sharf, R. A. Elhamied, M. Mekky, M. A. Elhamied. „On Flashing Over The Air „FOTA” for IoT Appliances - An ATMEL Prototype“. In: 2020 IEEE 10th International Conference on Consumer Electronics (ICCE-Berlin). 2020, S. 1–5. DOI: 10.1109/ICCE-Berlin50680.2020.
- [17] T. von Eicken. Readme.md von esp-link auf Github. https://github.com/jeelabs/esplink
- [18] SISTEMAS O.R.P.. Programando un Arduino remotamente con el módulo ESP8266.https://www.sistemasorp.es/2014/11/11/programando-un-arduino-remotamentecon-el-modulo-esp8266/
- [19] The Apache Software Foundation. Homepage von Ant. https://ant.apache.org/
- [21] The Apache Software Foundation. Homepage von Maven. https://maven.apache.org/

Quellen (5/6)

- [22] The Apache Software Foundation. Feature Summary. https://maven.apache.org/maven-features.html
- [23] Baeldung. Ant vs Maven vs Gradle. https://www.baeldung.com/ant-maven-gradle
- [24] Gradle Inc. . Gradle Guides. https://gradle.org/guides/
- [25] The Apache Software Foundation. Apache Ant. https://de.wikipedia.org/wiki/Apache_Ant
- [26] The Apache Software Foundation. Gradle. https://de.wikipedia.org/wiki/Gradle
- [27] The Fraunhofer Institute for Mechatronic Systems Design IEM. MechatronicUML. https://www.mechatronicuml.org/index.html

Quellen (6/6)

- [28] PlatformIO. Your Gateway to Embedded Software Development Excellence – PlatformIO. https://platformio.org/
- [29] Arduino. Arduino_Logo_Registered. https://de.wikipedia.org/wiki/Arduino_(Plattform)#/media/Datei:Arduino_Logo_Registered.svg
- [30] Eclipse Foundation. Eclipse TEA. https://eclipse.dev/tea/index.php
- [31] Eclipse Foundation. EASE Scripting | The Eclipse Foundation. https://eclipse.dev/ease/
- [32] Eclipse Foundation. Eclipse Advanced Scripting Environment | The Eclipse Foundation . https://eclipse.dev/ease/documentation/writing_modules/
- All links have been checked on 12th December 2023 12:04.