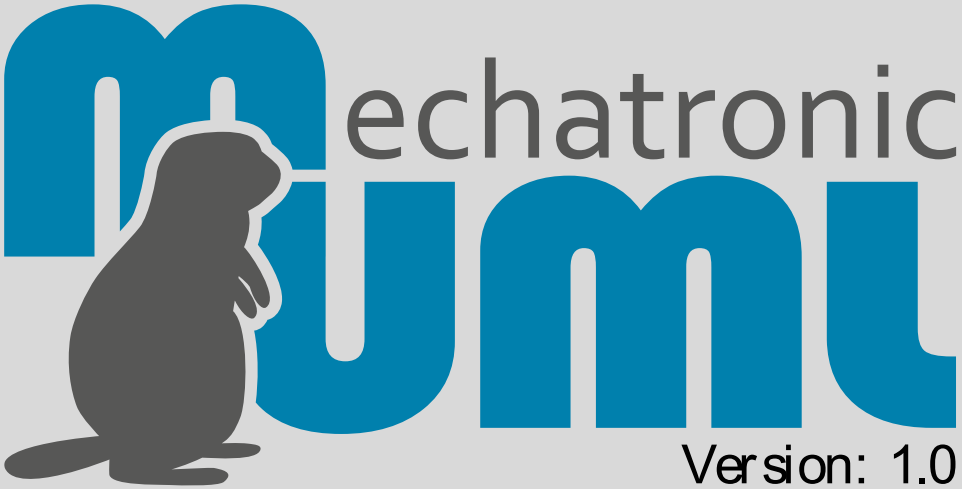
Universität Stuttgart

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

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



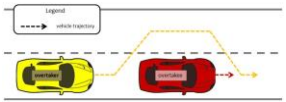
Version: 1.0 [27]

Sebastian Baumfalk

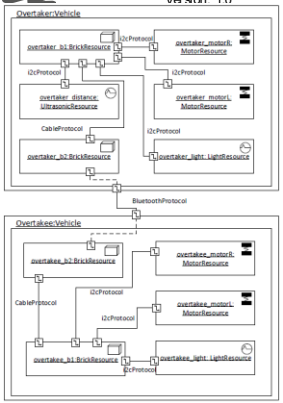
Motivation (1/2)

Problem

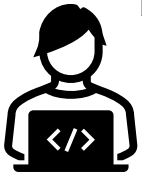
[1]

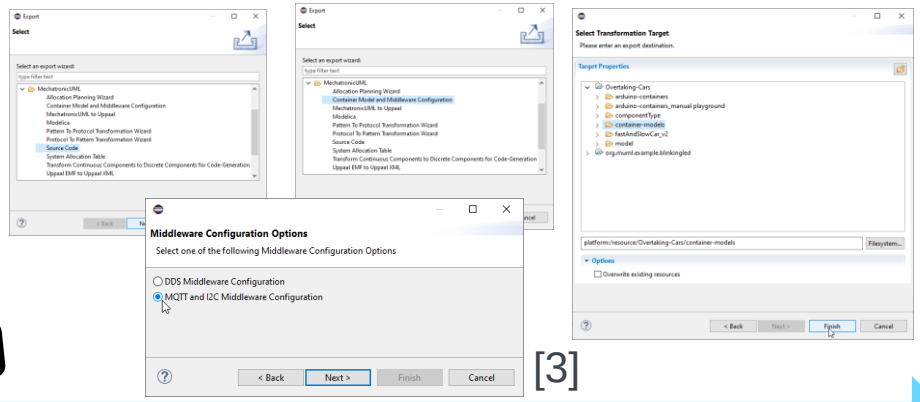


[27]



[1]





[3]

Manuelle Generierung

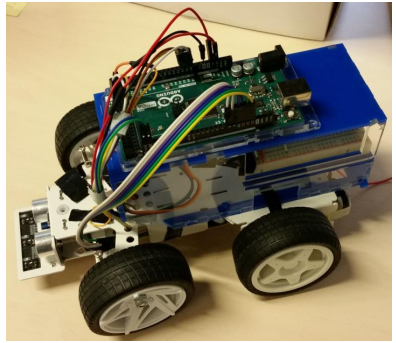
Post-Processing-Ablauf [Coda]:

1. viele Dateien kopiert oder verschoben
2. Korrektur der #include-Anweisungen
3. Nachtragen von Befehlen
4. Eintragen von Daten

...

Code

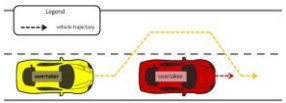
Arduino-Sketches



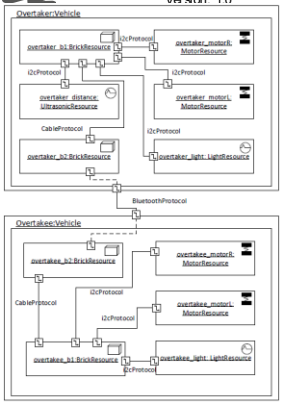
Motivation (2/2)

Ziel

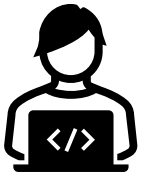
[1]



[27]



[1]



Automatische Generierung und Nachbearbeitung

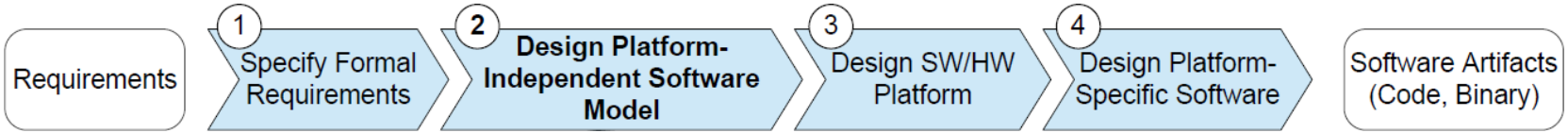
- Zuverlässig
- Schnell
- Flexibel

nutzungsbereite Code-Dateien

[29]

Grundlagen (1/2)

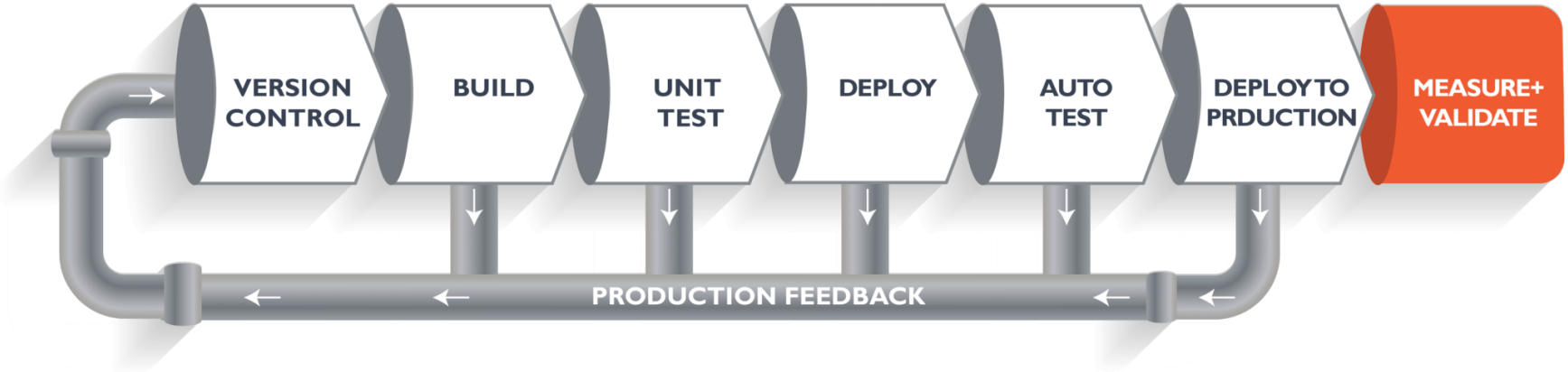
MechatronicUML



[7] (zugeschnitten)

Grundlagen (2/2)

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



[6]

Verwandte Arbeiten (2/3)

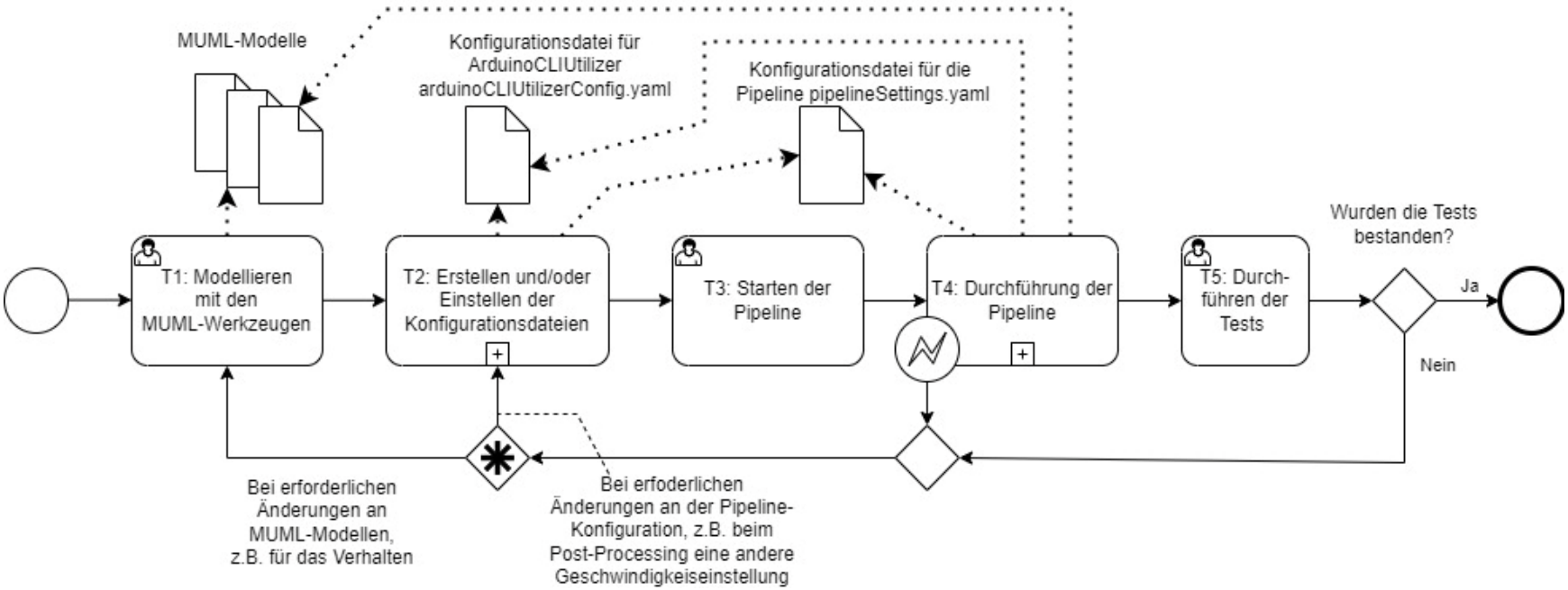
Automatisierungsansätze für Eclipse

- Versuchte Automatisierungsansätze für Eclipse:
  - EASE
  - TEA
  - Apache Ant
  - Apache Maven
  - Gradle
- Waren nicht direkt nutzbar

  ➡ Erstellung von eigenem Plug-In für Automatisierung

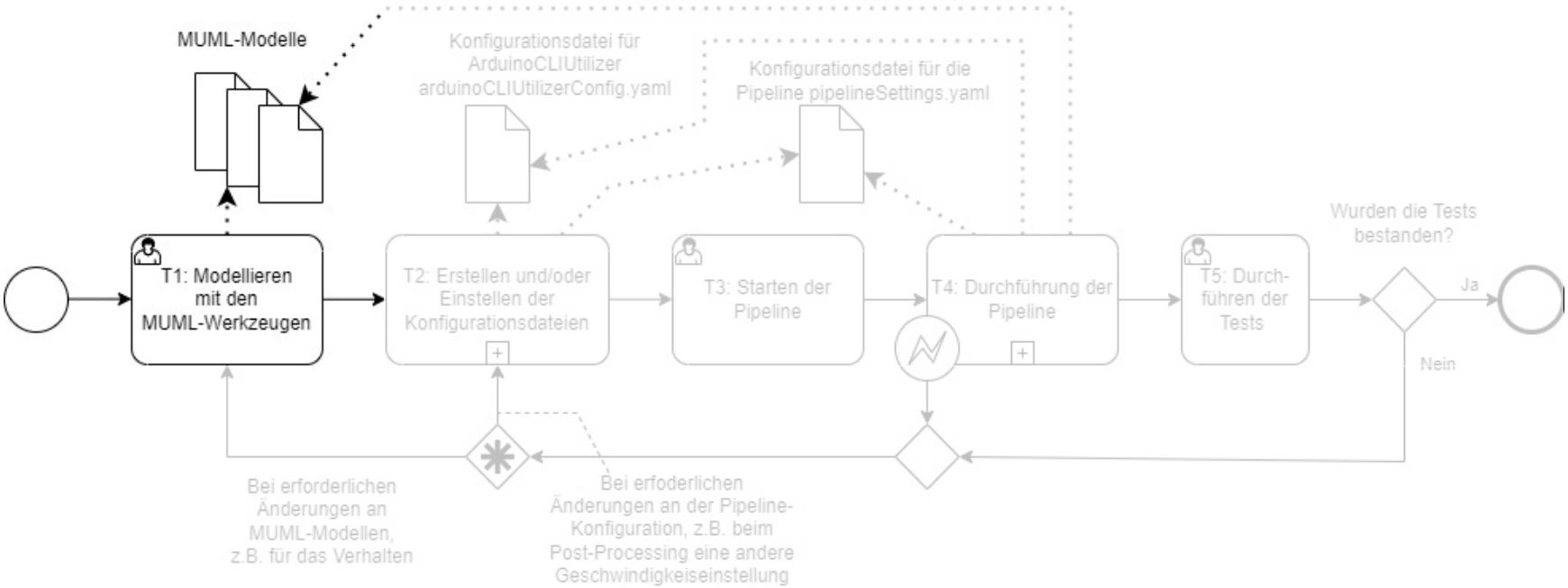
Verwandte Arbeiten (3/3)

CI/CD-Pipeline



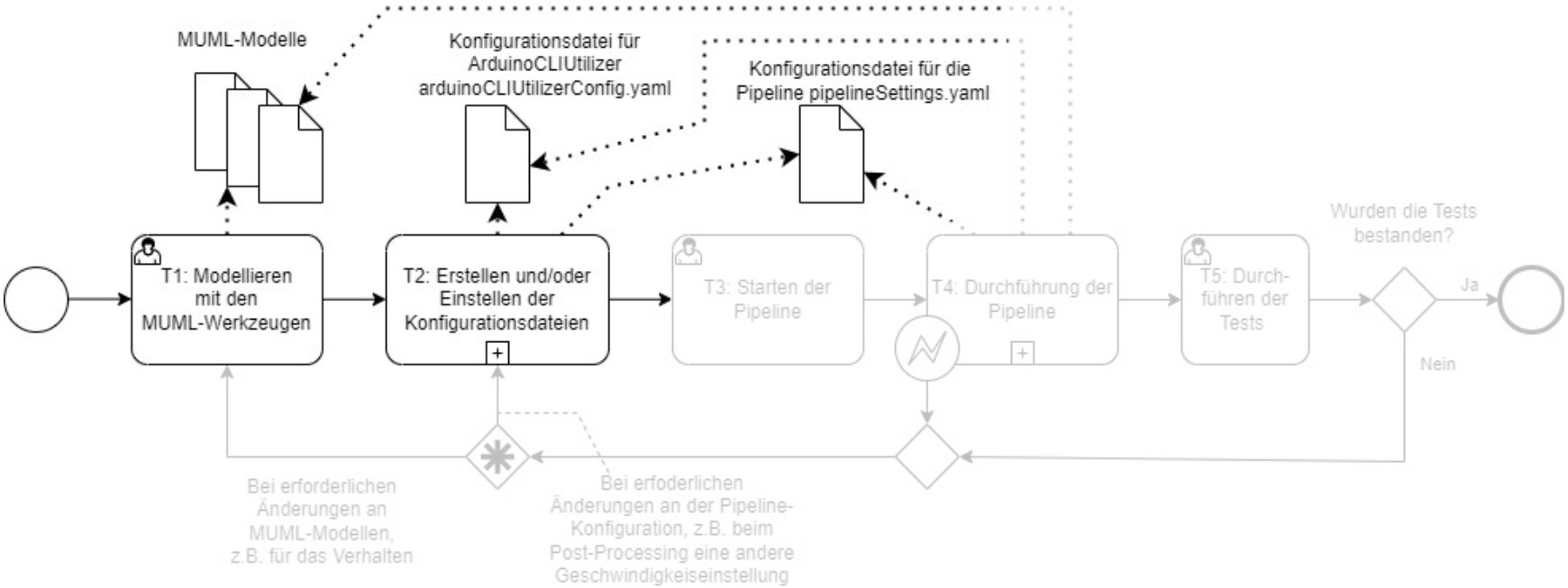
Entwurf (1/11)

Nutzungsablauf ()



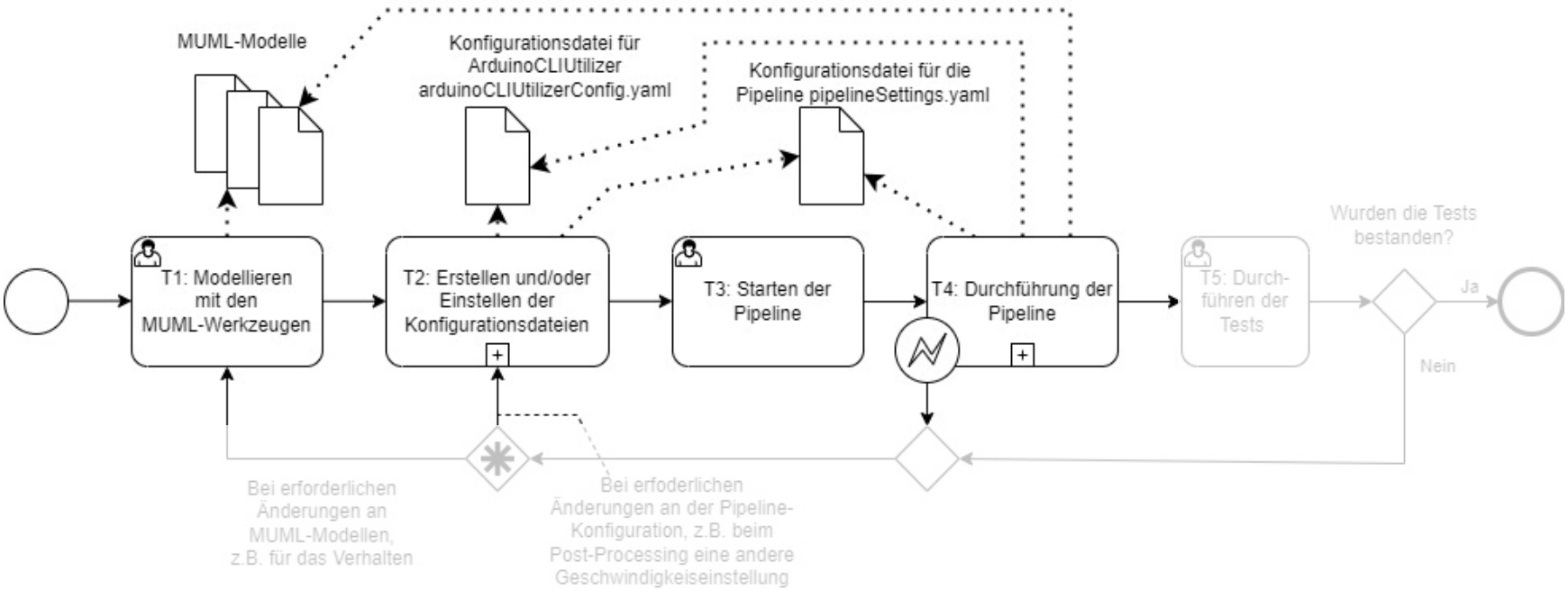
Entwurf (1/11)

Nutzungsablauf ()



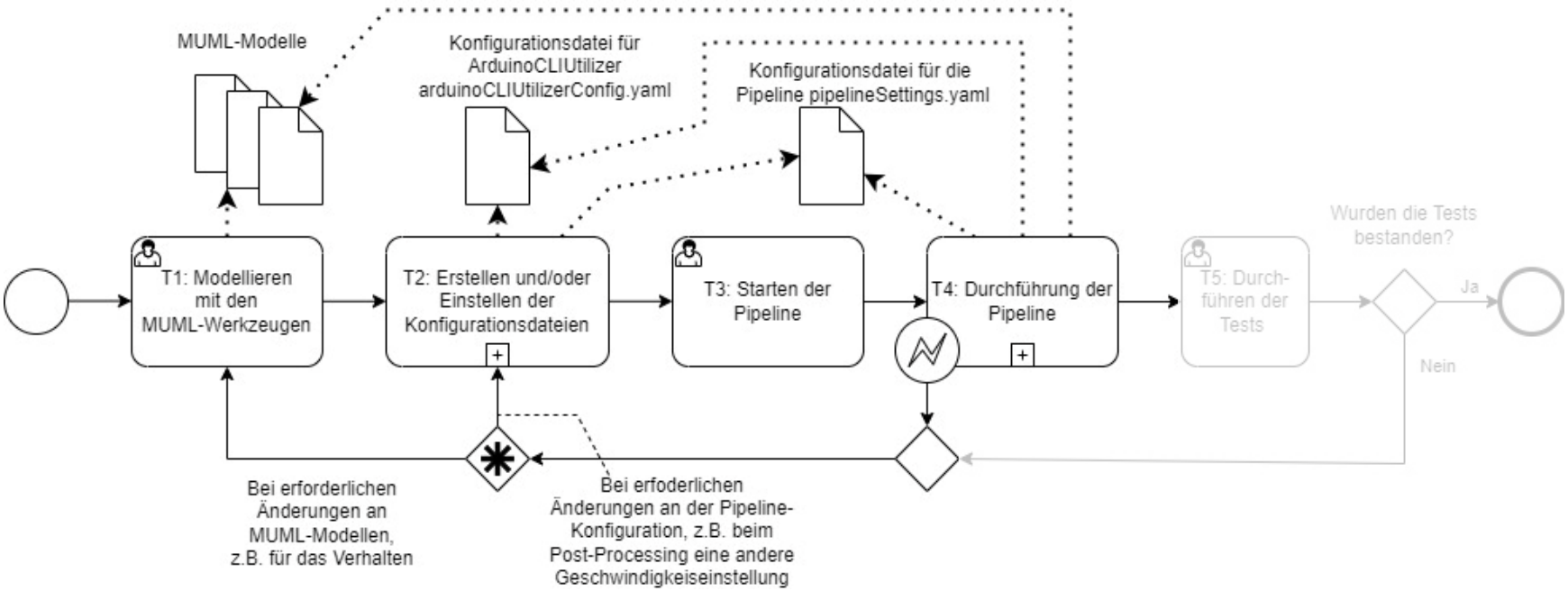
Entwurf (1/11)

Nutzungsablauf ()



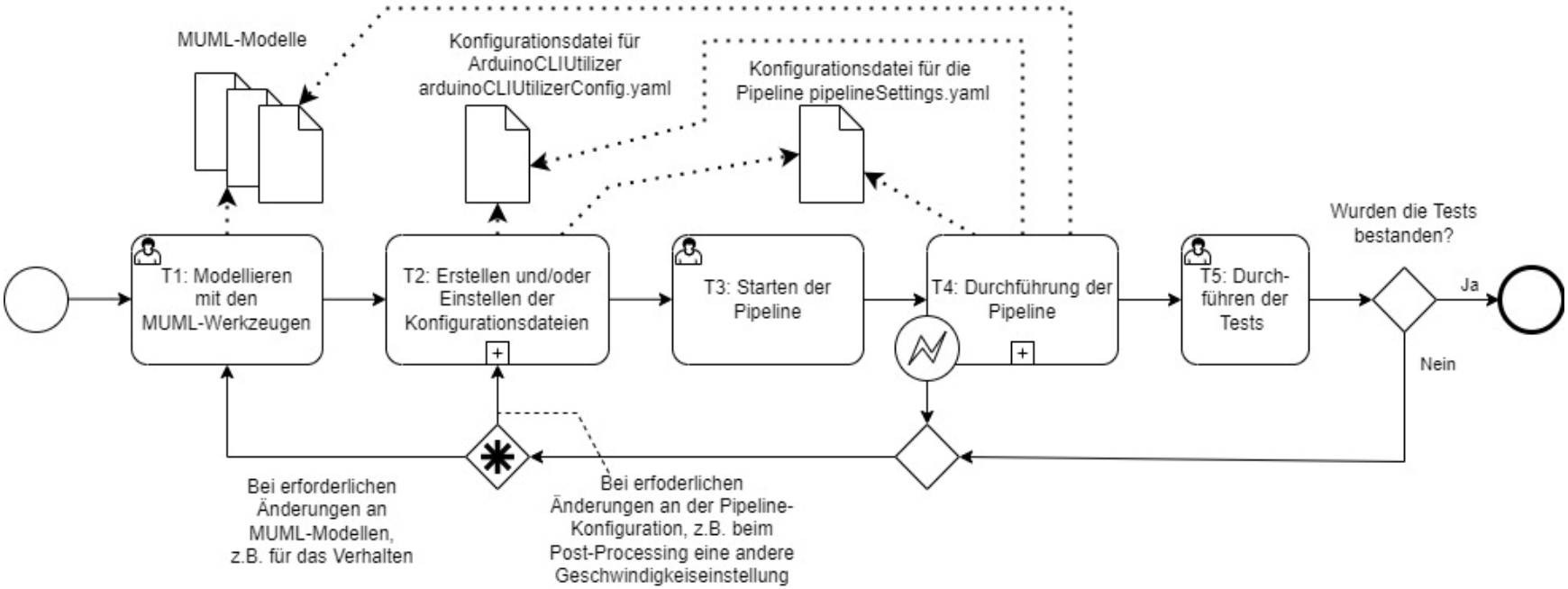
Entwurf (1/11)

Nutzungsablauf ()



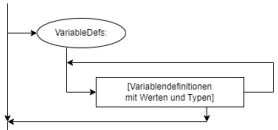
Entwurf (1/11)

Nutzungsablauf ()

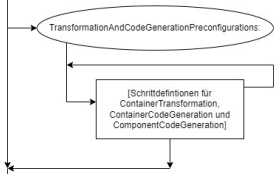


Entwurf (6/11)

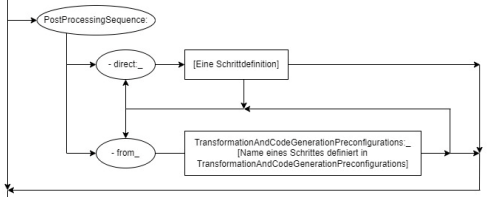
Übersicht Aufbau Pipelinekonfiguration (1/5)



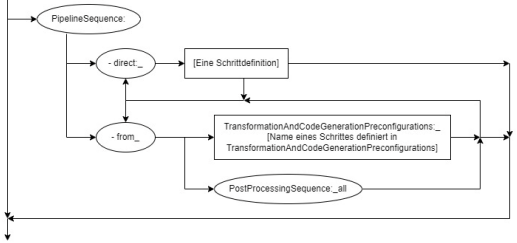
- Variablendefinitionen als Liste
  - [Typ] [Name] [Inhalt]



- Vorkonfigurationen für MUML-Werkzeuge



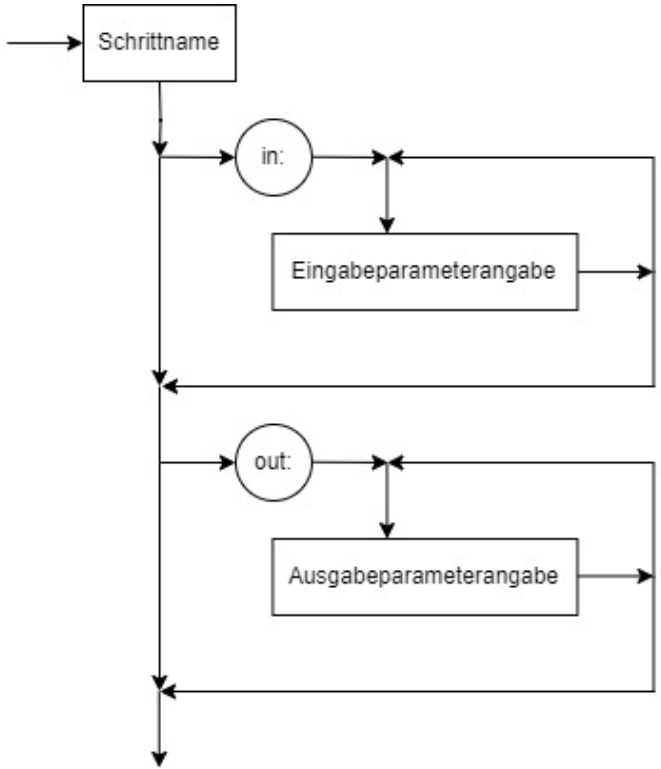
- Sequenz des Post-Processing-Ablaufs



- Sequenz der eigentlichen Pipelinenutzung

Entwurf (11/11)

Aufbau Pipelineschritteinträge



LookupBoardBySerialNumber:

in:

boardSerialNumber: from → fastCarCoordinatorECUBoardSerialNumber

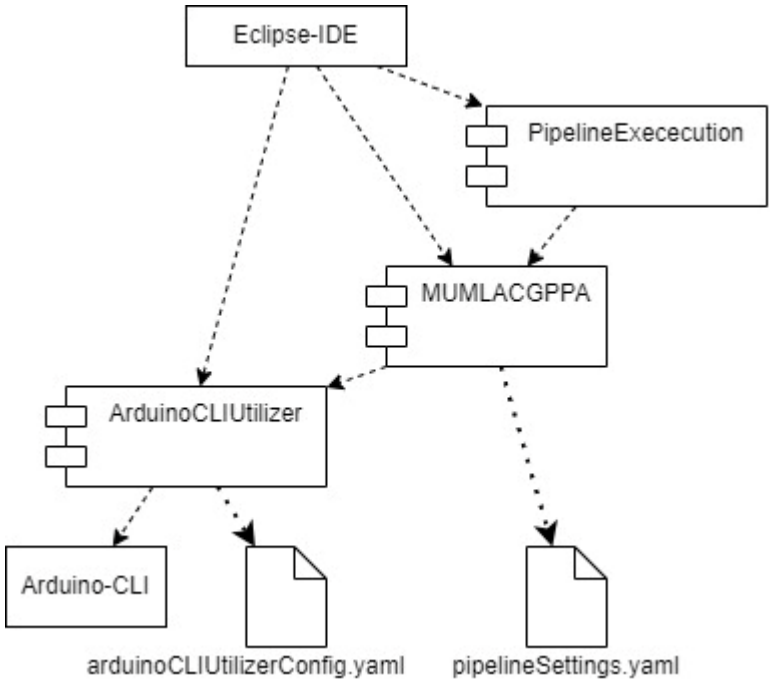
boardTypeIdentifierFQBN: from → usedCoordinatorBoardIdentifierFQBN

out:

ifSuccessful: ifSuccessful

foundPortAddress: foundPortAddress

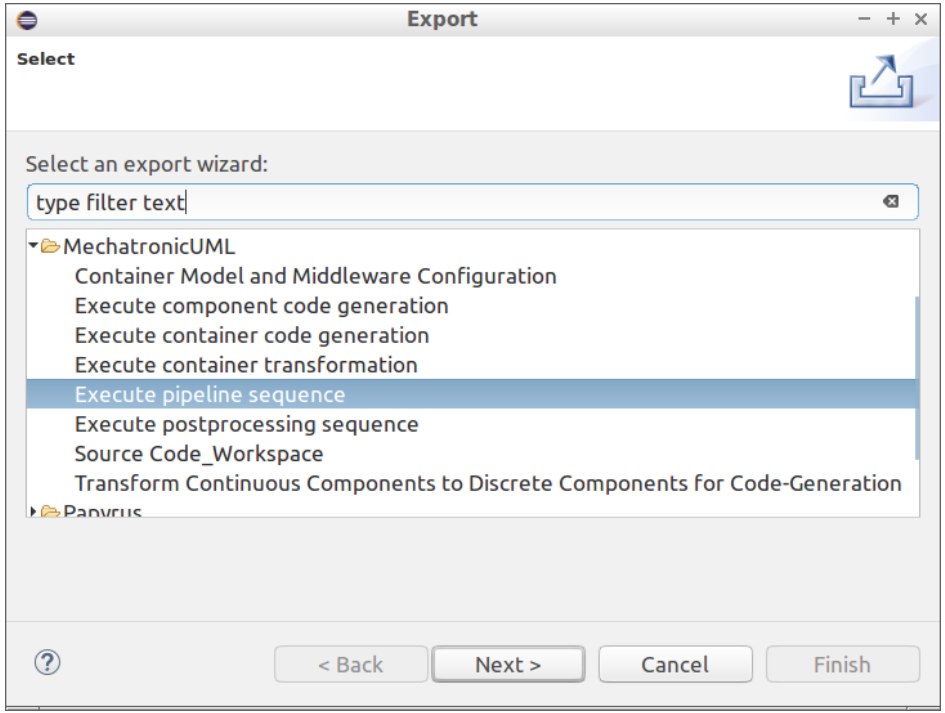
Umsetzung
Übersicht



Umsetzung 3 - Automatisierung per Pipeline (1/4)

Starten

- Export-Wizard als Workaround



Umsetzung 3 - Automatisierung per Pipeline (2/4)

Ausführung (1/2)

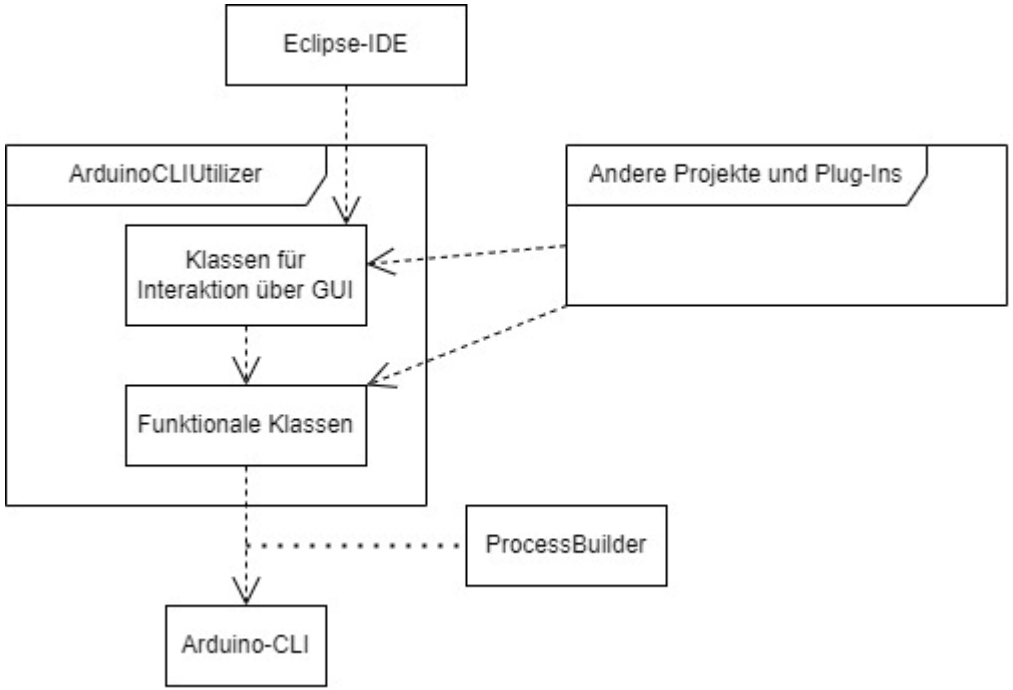
- Lese-/Interpretations-Reihenfolge:
  1. VariableDefs
  2. TransformationAndCodeGenerationPreconfigurations
     - Direkter Zugriff
  3. PostProcessingSequence
     - Zugriff wie bei Listen
  4. PipelineSequence
     - Zugriff wie bei Listen

Umsetzung 3 - Automatisierung per Pipeline (4/4)

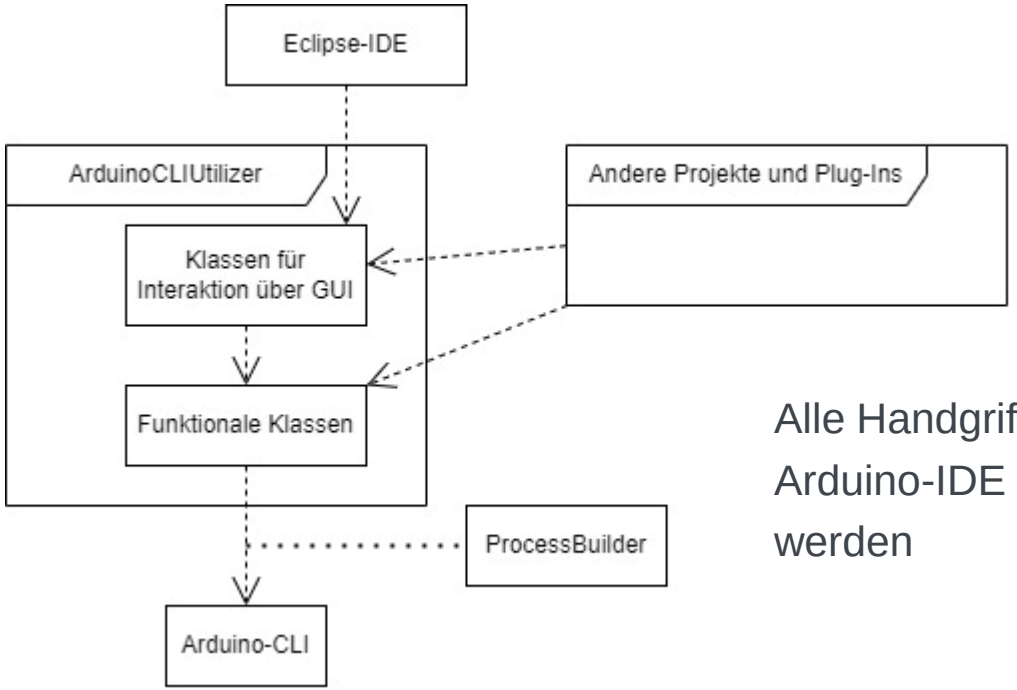
Fehlerprüfung

- 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 2: Integration per Arduino-CLI



Umsetzung 2: Integration per Arduino-CLI



Alle Handgriffe, die mit der Arduino-IDE durchgeführt werden

Evaluation

Was bewertet wurde

- Codequalität der entwickelten Eclipse-Plug-Ins
- Pipelinequalität
- Anwendungsszenario als Praxistest und Vergleichspunkt

Evaluation (8/15)

Vorgehensweisen (8/10)

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

Evaluation (9/15)

Vorgehensweisen (9/10)

- 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 (10/15)

Vorgehensweisen (10/10)

↓

- 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 (14/15)

Ergebnisse Beispielszenario „Kooperatives Überholen“ (1/2)

- 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 (15/15)

Ergebnisse Beispielszenario „Kooperatives Überholen“ (2/2)

↓

- 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

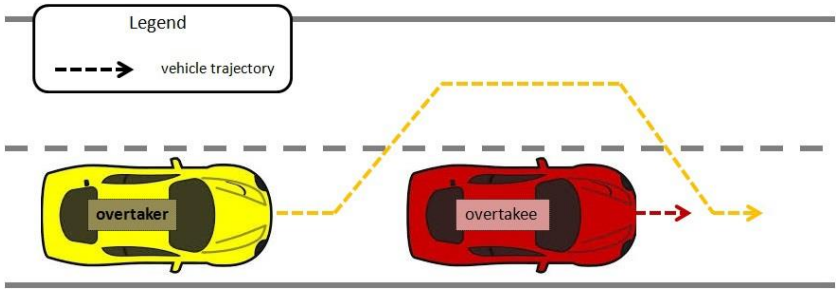
Schlussfolgerung

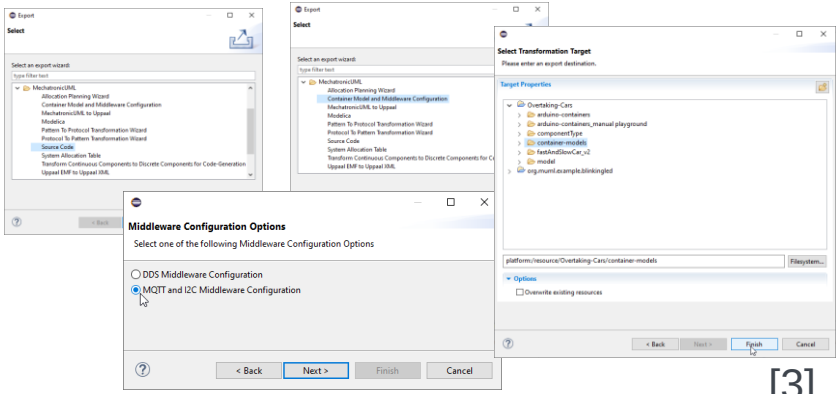
- Pipeline automatisiert gesamten Ablauf

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

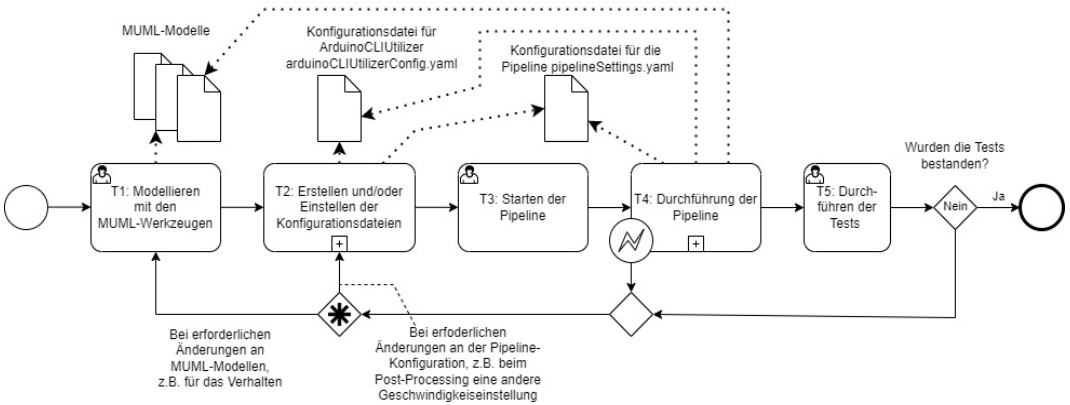
Zusammenfassung

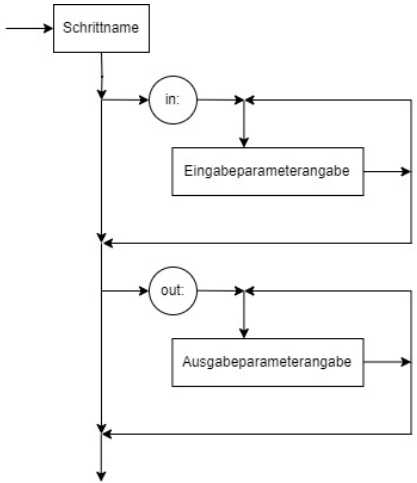
[1]





[3]





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

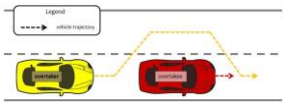
Quellen (6/6)

- [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.

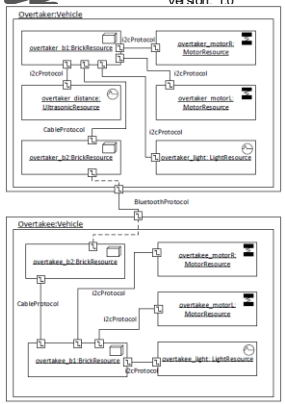
Motivation (2/4) X

Problem (2/2)

[1]



[27]

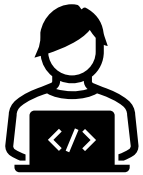


[1]

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

...



vollständige Code-Dateien

ARDUINO

[29]

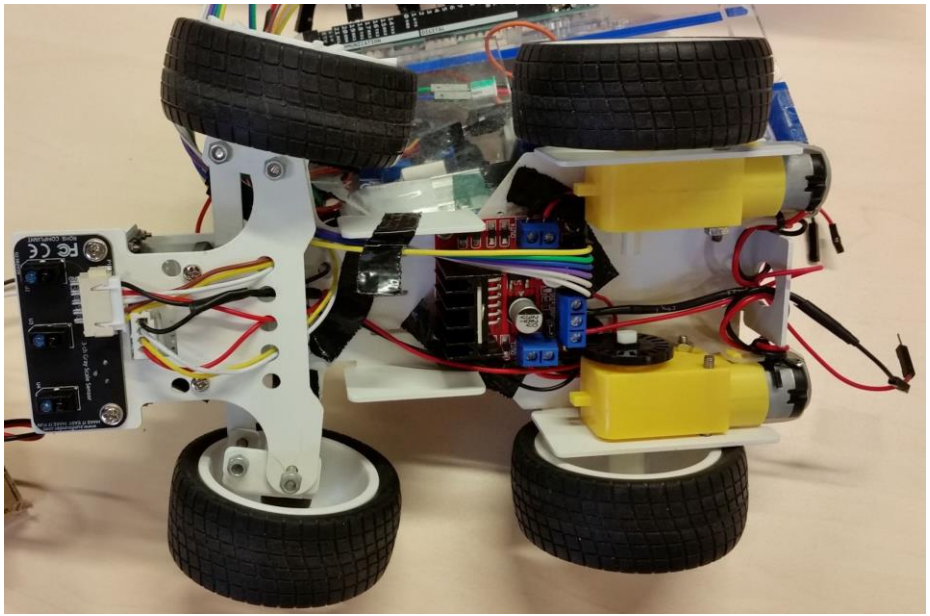
Manueller Post-Processing-Ablauf [Reib]

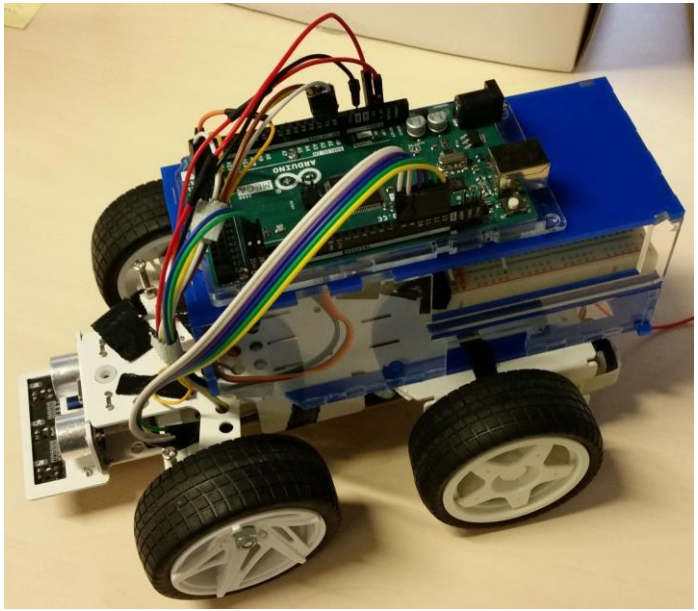
- Hoher Zeitbedarf
- Hohe Fehlerwahrscheinlichkeit

Motivation (4/4) X

Ziel (2/2)

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



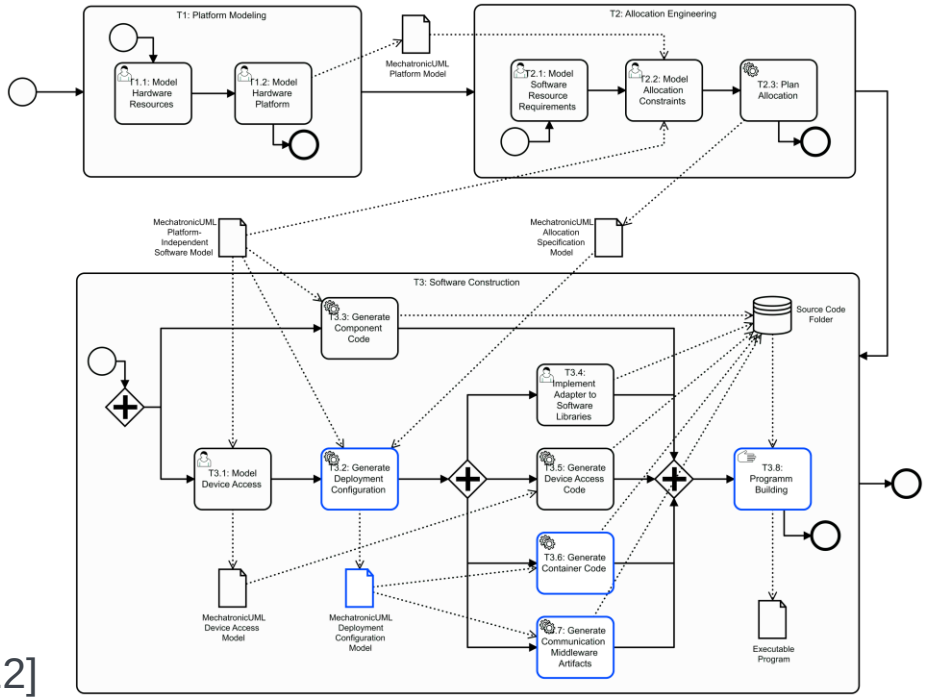


Lösungsansatz X

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

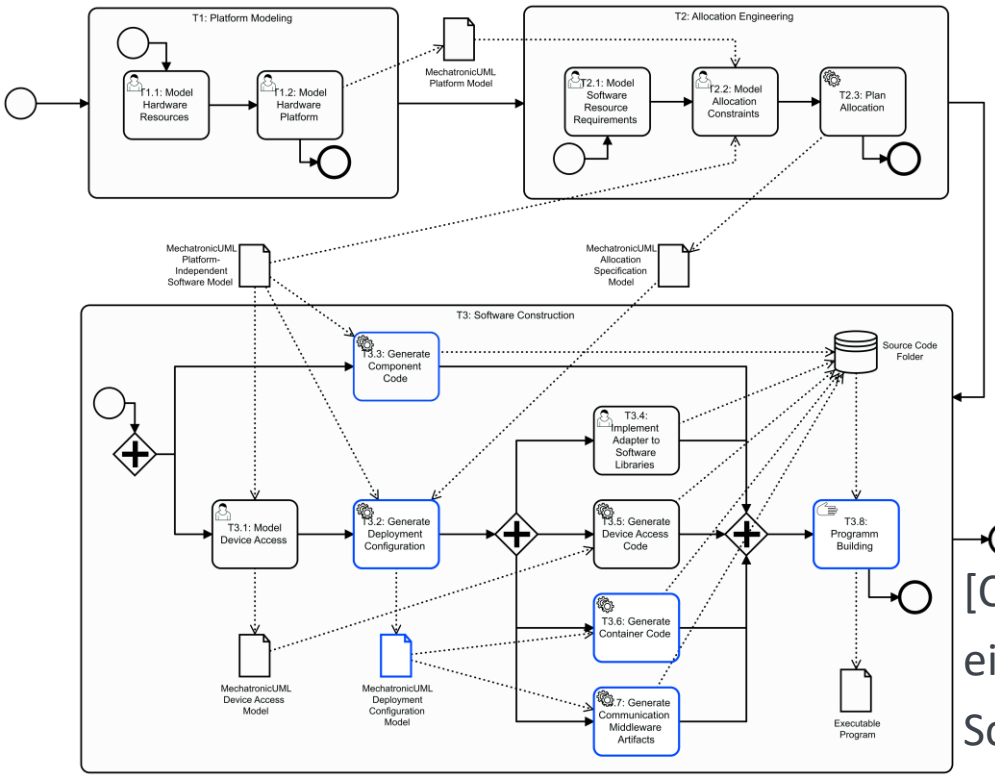
Verwandte Arbeiten (1/3) X

MechatronicUML mit Stürners Erweiterung



[12]

Analyse des bisherigen Arbeitsablaufes X



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

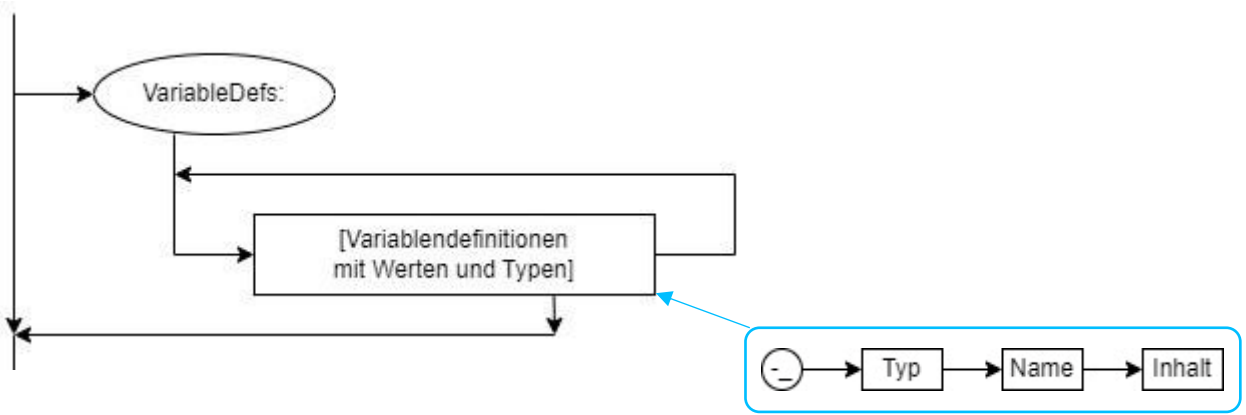
INFO

INFO

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

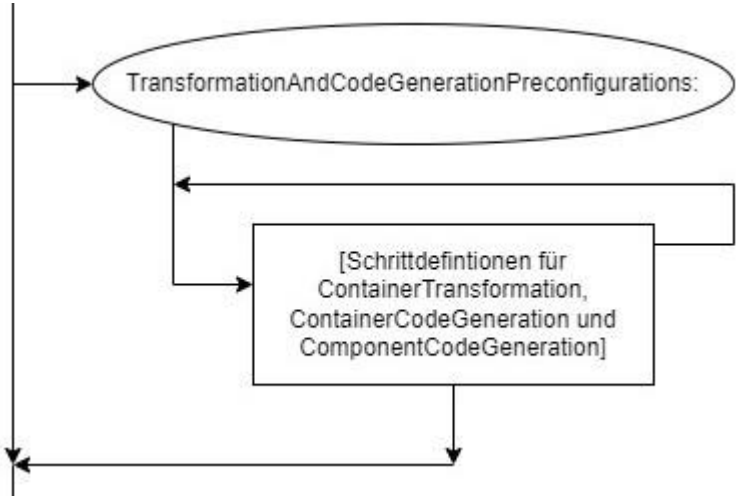
Entwurf (7/11) X

Übersicht Aufbau Pipelinekonfiguration (2/5)



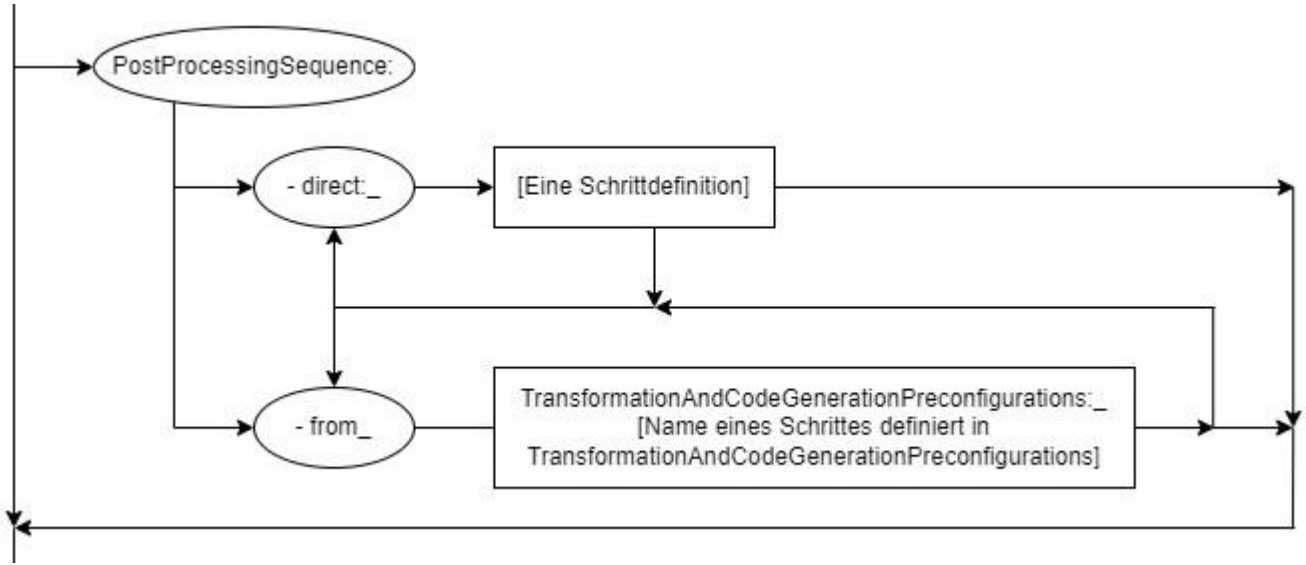
Entwurf (8/11) X

Übersicht Aufbau Pipelinekonfiguration (3/5)



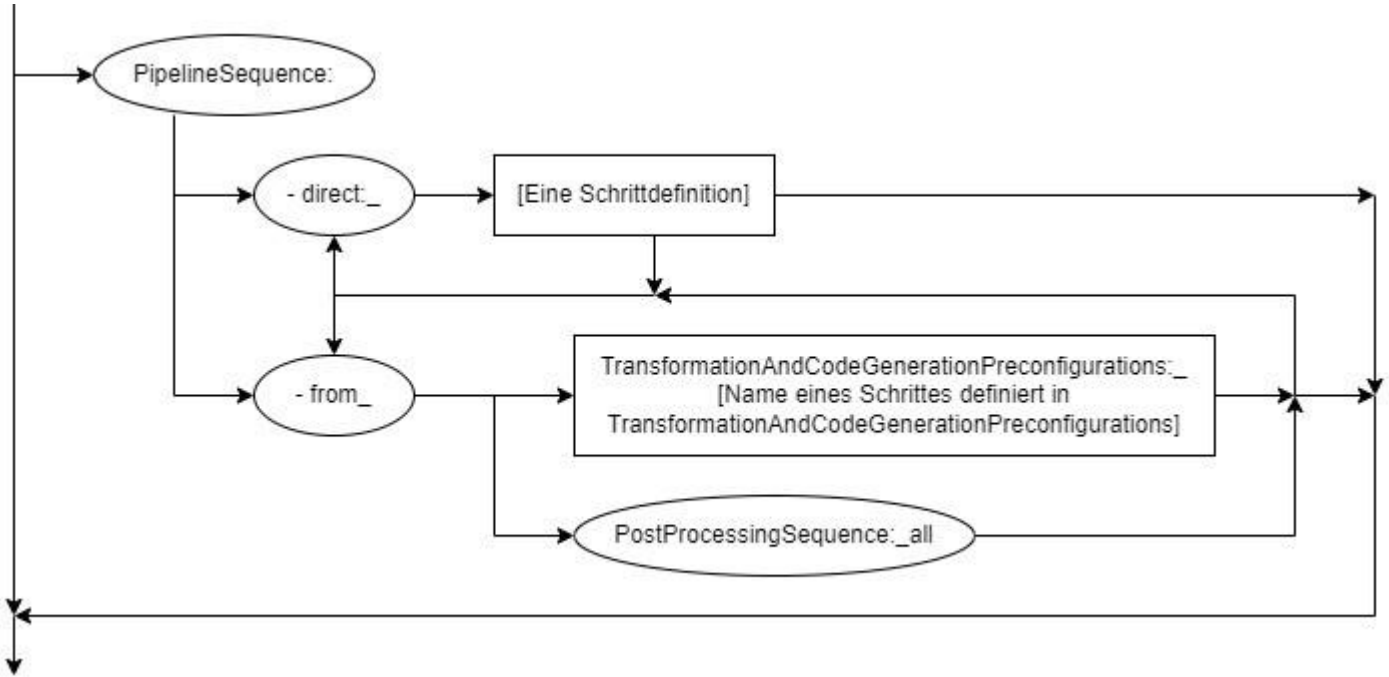
Entwurf (9/11) X

Übersicht Aufbau Pipelinekonfiguration (4/5)



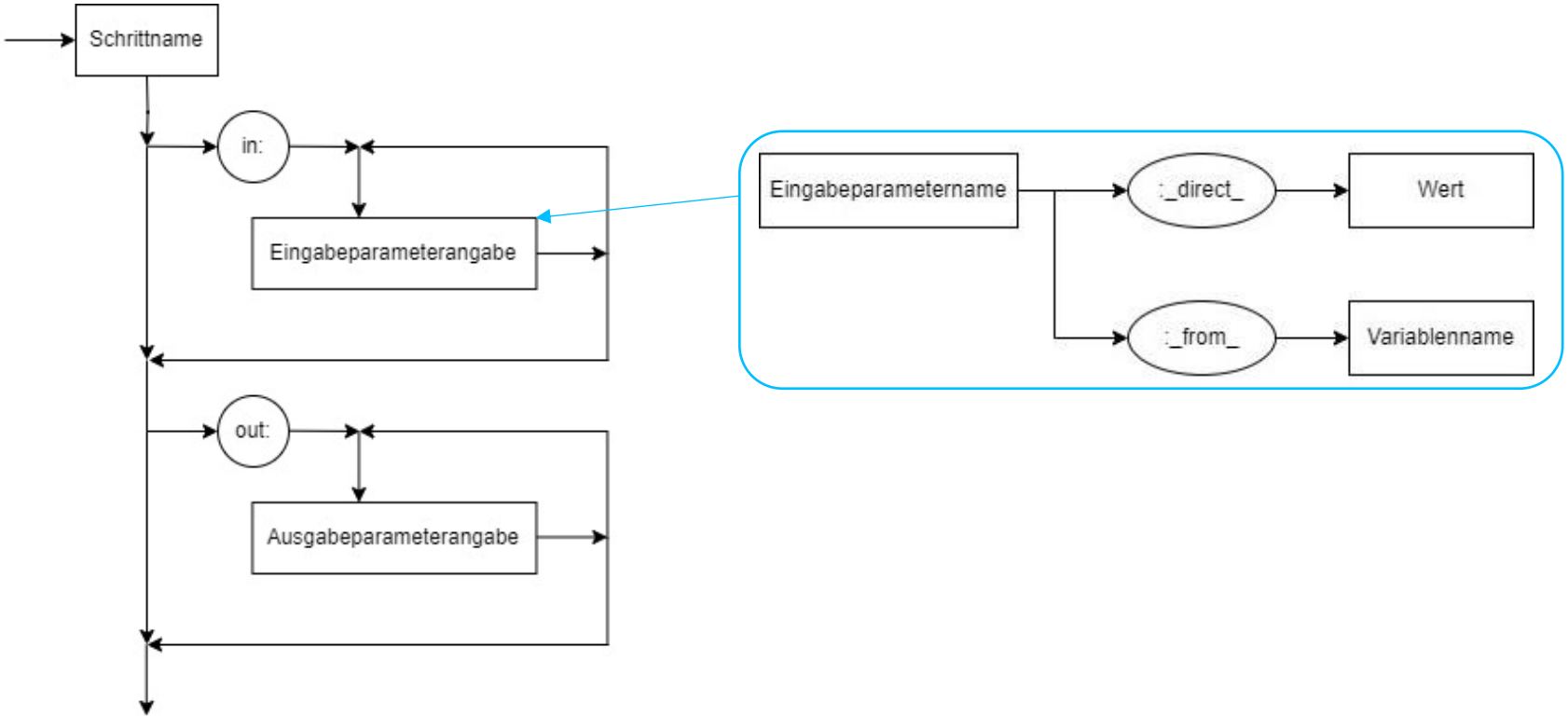
Entwurf (10/11) X

Übersicht Aufbau Pipelinekonfiguration (5/5)



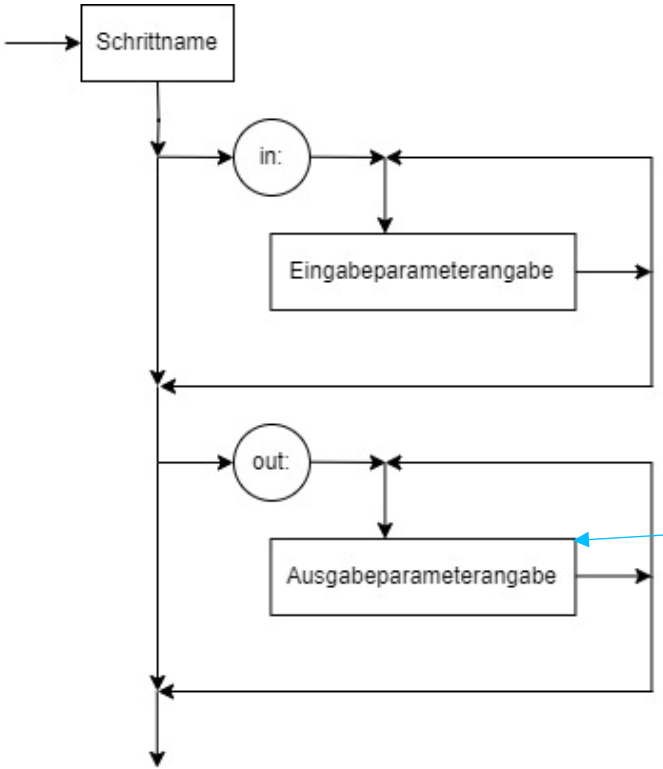
Entwurf (11/11) X

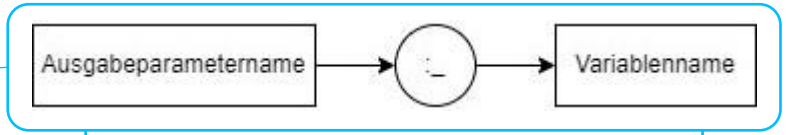
Aufbau Pipelineschritteinträge (6/6)



Entwurf (11/11) X

Aufbau Pipelineschritteinträge (6/6)





Automatisch: Dynamische Erstellung oder Überschreiben

Keine Typenänderung erlaubt!

Umsetzung 1: Volle Unterstützung von neuer oder geänderter Hardware, Software und Bibliotheken (1/4) X

Modelländerungen, wichtigstes: (1/2)

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 (2/4) X

Modelländerungen, wichtigstes: (2/2)

↓

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 (3/4) X

Änderungen am Post-Processing-Ablauf (1/2)

- 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 (4/4) X

Änderungen am Post-Processing-Ablauf (2/2)

↓

- 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 3 - Automatisierung per Pipeline (3/4) X

Ausführung (2/2)

- 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

Evaluation (1/15)

Vorgehensweisen (1/10)

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

Evaluation (2/15)

Vorgehensweisen (2/10)

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

Evaluation (3/15)

Vorgehensweisen (3/10)

- 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 (4/15)

Vorgehensweisen (4/10)

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

Evaluation (5/15)

Vorgehensweisen (5/10)

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

↓

Evaluation (6/15)

Vorgehensweisen (6/10)

- 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 (7/15)

Vorgehensweisen (7/10)

- Qualitätsbewertungen
  - Entwickelter Code: ISO25010-Standard

Evaluation (7/15)

Vorgehensweisen (7/10)

- Qualitätsbewertungen
  - Entwickelter Code: ISO25010-Standard

Evaluation (11/15)

Ergebnisse Qualitätsbewertungen

- 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 (12/15)

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 (13/15)

Ergebnisse CI/CD-Pipelinequalität

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

Evaluation
Diskussion

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

Schlussfolgerung

Vorteile (1/2)

- 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 (2/2)

↓

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

Zusammenfassung (1/2)

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

↓

Zusammenfassung (2/2)

- 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