7.2 Integration der ArduinoCLI

Dieses Kapitel führt die Integration der ArduinoCLI in die Eclipse-IDE als ein Eclipse-Plug-In-Projekt ein. Hiermit soll die Nutzerschaft beim Arbeiten mit MUML und dem generierten Code nur noch selten von der Eclipse-IDE in das Dateisystem und in die Arduino-IDE wechseln müssen, indem die erforderlichen manuell durchgeführten Arbeitsschritte mittels Aufrufen von der Eclipse-IDE aus durchgeführt werden können. Zusätzlich soll es für die später beschriebene Umsetzung der Pipeline (siehe Kapitel 10 [TODO: Nach Notwendigkeit aktualisieren]) seine Funktionalität für andere Eclipse-Plug-In-Projekte bereitstellen.
Im Rahmen dieses Kapitels werden also die folgenden Forschungsziele beantwortet:

FZ3.1: Welche der manuellen Schritte mit der Arduino-IDE oder deren Teilen entsprechen welchen Befehlen der Arduino-CLI?
FZ3.2: Welche Aufrufe müssen bei der Arduino-CLI zusätzlich gemacht werden?
FZ3.3: Können durch die Arduino-CLI dieselben Ergebnisse wie mit der Arduino-IDE erzielt werden?
FZ3.4: Wie kann Sofdcar-HAL per Arduino-CLI installiert werden?

Die hierfür fehlenden Fähigkeiten der Eclipse-IDE werden durch die folgenden Anforderungen spezifiziert:

ANF7.2.1: Alle Schritte, die bisher über das Öffnen der .ino-Dateien mit der Arduino-IDE und dem Verwenden von dieser (die Arduino-IDE) erfolgten (siehe FZ3.1), müssen über die GUI der Eclipse-IDE durchgeführt oder ausgelöst werden können.
ANF7.2.2: Das resultierende Eclipse-Plug-In muss mögliche zusätzliche erforderliche Aufrufe durchführen können.
ANF7.2.3: Das Verhalten von Boards, auf die durch das resultierende Eclipse-Plug-In Arduino-Code installiert wurde, muss identisch sein zu dem Verhalten von Boards, auf die über die Arduino-IDE Arduino-Code installiert wurde.
ANF7.2.4: Es müssen Schnittstellen vorhanden sein, damit andere Eclipse-Plug-Ins auf diese Funktionen zugreifen können.

Um die Umsetzung zu beschreiben ist dieses Kapitel in die folgenden Abschnitte unterteilt:
In 7.2.1 [TODO: Verweis] werden die manuellen mit der Arduino-IDE durchgeführten Schritte den Befehlen der Arduino-CLI zugeordnet und mögliche zusätzliche Schritte erörtert.
In 7.2.2 [TODO: Verweis] wird der Test der Arduino-CLI beschrieben werden.
In 7.2.3 [TODO: Verweis] wird die Installation der Bibliothek Sofdcar-HAL mit der Arduino-CLI beschrieben werden.
In 7.2.4 [TODO: Verweis] wird die Implementierung beschrieben.
In 7.2.5 wird das Behandeln von Boards , die ihre eigenen Typinformationen nicht angeben, erklärt.

7.2.1 Manuelle Schritte mit der Arduino-IDE und entsprechende Schritte mit der Arduino-CLI

[TODO: Noch etwas mehr Text ausdenken]
Die in # [TODO: Figurennummer der Tabelle] aufgestellte Liste an manuellen Schritten und wie diese durchgeführt werden wird hier jeweils mit dem entsprechenden Befehl oder den entsprechenden Befehlen gegenübergestellt.

Schritt	Arduino-IDE	Arduino-CLI
Auflisten angeschlossener Boards	Werkzeuge/Port/[die Liste ablesen]	arduino-cli board list

	oder [in der horizontalen Leiste direkt über den Dokumenten die Auswahlliste ablesen]	
Auswählen eines Ports PORT	Werkzeuge/Port/[gewünschte Auswahl auswählen] oder [in der horizontalen Leiste direkt über den Dokumenten die gewünschte Auswahl treffen]	[Beim Hochladen durch „--port [PORT]“ (ohne Anführungszeichen)]
Boardauswahl	Werkzeuge/ Board: „*“/ [entsprechende Auswahl auswählen] oder [in der horizontalen Leiste direkt über den Dokumenten die gewünschte Auswahl treffen]	(Beim Kompilieren und beim Hochladen)
Kompillieren/Überprüfen der .ino-Datei [INOFILE]	Sketch/Überprüfen oder [in der horizontalen Leiste direkt über den Dokumenten den Haken anklicken]	arduino-cli compile [INOFILE]
Hochladen der .ino-Datei [INOFILE]	Sketch/Hochladen oder [in der horizontalen Leiste direkt über den Dokumenten den Pfeil anklicken]	arduino-cli upload --port [PORT] --fqbn [BOARD] [INOFILE] oder arduino-cli upload --port [PORT] --fqbn [BOARD] --input-dir [DIR]
Bibliotheksinstallation (Automatisch)	Werkzeuge/ Bibliotheken verwalten/[In der Liste bei der gewünschten Bibliothek den Installieren-Knopf klicken] oder Seitliche Leiste/Bibliotheksverwalter/[In der Liste bei der gewünschten Bibliothek den Installieren-Knopf klicken]	Empfohle Vorbereitung: arduino-cli lib update-index Befehl selbst: arduino-cli lib install LIBRARY[@VERSION_NUMBER]... [flags]
Bibliotheksinstallation (.zip-Datei)	Sketch/ Bibliothek einbinden/ .ZIP-Bibliothek hinzufügen...	Aufruf 1: arduino-cli config set library.enable_unsafe_install

	[Bibliotheksdatei heraussuchen] oder Inhalt der .zip-Datei bzw. entpackten Bibliotheksordner nach Dokumente/Arduino/libraries verschieben	true Aufruf 2: arduino-cli lib install --zip-path [LIBRARYZIPFILE] Aufruf 3: arduino-cli config set library.enable_unsafe_install false oder Inhalt der .zip-Datei bzw. entpackten Bibliotheksordner nach /home/Arduino15/libraries verschieben
Updaten von Bibliotheken	Werkzeuge/ Bibliotheken verwalten/[In der Liste bei der updatebaren Bibliothek den Update-Knopf klicken]	arduino-cli lib upgrade [Bibliothek]

Hier folgt für weitere wichtige Aktionen eine weitere Tabelle:

Schritt	Arduino-IDE	Arduino-CLI
Updaten der Kern-Daten	[Seitliche Leiste]/ Board-Verwaltung/[In der Liste bei den updatebaren Daten den Update-Knopf klicken]	Empfohle Vorbereitung: arduino-cli core update-index Befehl selbst: arduino-cli core update
Installieren von Kernen (Zu arduino:xyz)	[Seitliche Leiste]/ Board-Verwaltung/[In der Liste bei den updatebaren Daten den Installieren-Knopf klicken]	arduino-cli core install [arduino:xys]
Kompilieren der .ino-Datei [INOFILE] und speichern im Ordner [OUT]	[Nicht manuell möglich]	arduino-cli compile --fqbn [BOARD] [INOFILE] --output-dir [OUT]
Hochladen von bereits kompiliertem Code bzw. einer .hex-Datei [INOHEXFILE]	[Nicht manuell möglich]	arduino-cli upload --port [PORT] --fqbn [BOARD] --input-file [INOHEXFILE]

Diese beiden Tabellen [TODO: Verweise] stellen die Antwort auf die Forschungsfrage FZ3.1: „Welche der manuellen Schritte mit der Arduino-IDE oder deren Teilen entsprechen welchen Befehlen der Arduino-CLI?“ dar.

Je nach Form der Installation kann der Pfad zu dem Installationsort der Arduino-CLI nicht automatisch in den Systemdateien eingetragen worden sein. Wenn kein manuelles erfolgreiches Nachtragen durchgeführt wurde, muss als Vorbereitung für die Nutzung in der Konsole der Pfad temporär eingestellt werden. Dies erfolgt durch den Befehl „export PATH=[Pfad zu arduinocli-

Datei]:$PATH“. Sein Effekt beschränkt sich nur auf die Terminalinstanz, in der er durchgeführt wurde, und solange diese aktiv ist. Neu geöffnete Terminalinstanzen wären also nicht beeinflusst.

Bei dem Installieren von Bibliotheken ist der Befehl „arduino-cli lib update-index“ empfohlen, damit die Daten für das Herunterladen von den verschiedenen Adressen aktuell sind.

Dies beantwortet die Forschungsfrage FZ3.2 „Welche Aufrufe müssen bei der Arduino-CLI zusätzlich gemacht werden?“

7.2.2 Test der Arduino-CLI:

Um sicher zu gehen, dass die Arduino-CLI wirklich die gleichen Ergebnisse wie die Arduino-IDE liefert, wurden zu dem Beispiel „Blink“ Variationen mit unterschiedlichen Phasenlängen jeweils mit der Arduino-IDE und mit der Arduino-CLI auf einem Arduino-UNO-Board installiert und es traten keine Unterschiede auf. Andere Testprogramme wießen auch keine erkennbaren Unterschiede auf. Spätere Tests mit den Arduinocodes und Arduinoboards für das Anwendungsszenario ergaben auch keine Unterschiede.
Somit kann FZ3.3 mit „Ja“ beantwortet werden.

7.2.3 Installation der Bibliothek Sofdcar-HAL mit der Arduino-CLI

Das Installieren der Bibliothek Sofdcar-HAL aus dem Archiv „Sofdcar-HAL.zip“ erfolgt über eine Reihe an Anweisungen:

Aufruf 1: arduino-cli config set library.enable_unsafe_install true
Aufruf 2: arduino-cli lib install --zip-path [LIBRARYZIPFILE]
Aufruf 3: arduino-cli config set library.enable_unsafe_install false

Hierbei sind die Aufrufe „arduino-cli config set library.enable_unsafe_install true“ und „ arduino-cli config set library.enable_unsafe_install false“ notwendig, um nur für den Zeitraum der Installation zeitweilig die Installation von Bibliotheken in .zip-Archiven zu erlauben und dann wieder zu verbieten.
Die Anweisung „arduino-cli lib install --zip-path [LIBRARYZIPFILE]“ mit dem an der Stelle [LIBRARYZIPFILE] eingetragenen Pfad zu dem Archiv der heruntergeladenen Sofdcar-HAL-Bibliothek führt die Installation selbst aus.
Dies beantwortet FZ3.4: Wie kann Sofdcar-HAL per Arduino-CLI installiert werden?

7.2.4 Implementierung als Plug-In:

In diesem Abschnitt wird die Implementierung der oben genannten Aspekte als das Eclipse-Plug-In-Projekt „ArduinoCLIUtilizer“ beschrieben und es wird hierbei auf die wichtigsten der hierfür getroffenen Entscheidungen eingegangen werden. Für weitere Informationen und Diagramme siehe [TODO: Nummer] „Ergänzungsmaterial für die Integration der ArduinoCLI“.

Um ANF6.2.4 zu erfüllen, wurde beschlossen, dass die Klassen, die für die funktionalen Aspekte dieses Eclipse-Plug-In-Projekts verantwortlich sind, nur eine geringe Kopplung zu den Klassen und Teilen von Eclipse, die an der Interaktion mit dem Nutzer/ der Nutzerin beteiligt sind, aufweisen sollen. Zusätzlich soll das theoretische Loslösen aus der Verwendung mit Eclipse leicht und schnell umsetzbar erfolgen können. Deshalb wurden Klassen, die für die Interaktionen mit dem Nutzer/ der Nutzerin verwendet werden, in das Package „de.ust.arduinocliutilizer.popup.actions“ eingeplant.

Es wird eine Konfigurationsdatei namens „arduinoCLIUtilizerKonfig.yaml“ in dem Ordner „automatisationConfig“ in dem Projektordner des Eclipseprojekts, in dem gearbeitet wird, verwendet. In dieser ist unter der Einstellung „arduinoCLIPathSetInPathEnvironment“ eingetragen, ob der Installationspfad zu der Arduino-CLI in den Systemdateien vorhanden ist, und unter der Einstellung „arduinoCLIDirectory“ der volle Pfad des Ordners, in dem die Arduino-CLI installiert ist. So kann sich die Arduino-CLI, oder genauer gesagt, die Klasse ArduinoCLICommandLineHandler, auf für das System möglicherweise fehlende Pfaddaten zu der Arduino-CLI anpassen. Weitere Informationen zu dem Leseablauf folgen unten in diesem Kapitel [oder Unterkapitel oder Abschnitt?].
Die Erstellung der Konfigurationsdatei wird per Aufruf in dem Kontextmenü ([Rechtsklick auf eine .zip-, .ino- or .hex-Datei]/"ArduinoCLIUtilizer"/"GenerateArduinoCLIUtilizer config file") gestartet. Die Klasse ArduinoCLIUtilizerConfigGenerator ist u.a. für die Erstellung des Inhalts selbst verantwortlich. Sie prüft dabei den Installationszustand der Arduino-CLI und passt den Wert der Einstellung „arduinoCLIPathSetInPathEnvironment“ entsprechend an.

#
Klassendiagramm über die funktionalen Klassen

[Evtl. erst in der Ergänzung]
Klassendiagramm über die *Action-Klassen, die für die Interaktion über die GUI verwendet werden

Für die Interaktion zwischen dem Javacode für das Eclipse-Plug-in und der Arduino-CLI wird java.lang.ProcessBuilder verwendet. Die darin enthaltene Klasse ProcessBuilder ermöglicht von Javaprogrammen aus das Durchführen von Kommandozeilenbefehlen in Form von Prozessen. Für diese Ausarbeitung werden von diesen wiederum der Beendigungswert bzw. Exitcode, die aufgezeichneten normalen Ausgaben sowie die aufgezeichneten Fehlerausgaben verwendet und zusammen mit dem jeweiligen verwendeten Befehl in der Klasse CallAndResponses gespeichert. Für das leichtere Finden von Fehlern erstellt jede Klasse, die einen Arbeitsschritt mit der Arduinosoftware durchführt, automatisch in demselben Verzeichnis wie die verwendete Datei (z.B. Sofdcar-HAL.zip) einen Ordner namens „SavedResponses“ und speichert in diesem eine einfache Textdatei mit den folgenden Ausführungsinformationen:

- Command:
  [Der Konsolenbefehl der entsprechenden Klasse.]
- Results:
- Exit code: [Beendigungswert bzw. Exitcode]
- Normal response stream:
- [Aufgezeichnete normale Ausgaben]
- Error response stream:
- [Aufgezeichnete Fehlerausgaben]

Die Befehle für die Arduino-CLI werden intern von der Klasse ArduinoCLICommandLineHandler verwaltet, weil, wie bereits zuvor in Abschnitt [# TODO: Nummer] berichtet wurde, je nach Form der Installation der Pfad zu dem Installationsort der ArduinoCLI in den Systemdateien fehlen kann. Hierfür liest es die zuvor erwähnte Datei „arduinoCLIUtilizerKonfig.yaml“. Zunächst wird der Wert von „arduinoCLIPathSetInPathEnvironment“ gelesen und so intern festgelegt, ob dem System der Pfad zu der Arduino-CLI bekannt ist und somit direkt genutzt werden kann, oder ob er fehlt und bei den Aufrufen temporär eingestellt werden muss. Falls ersteres der Fall ist („arduinoCLIPathSetInPathEnvironment: true“), so werden die Befehle direkt für die Ausführung übernommen. Falls das zweite der Fall ist („arduinoCLIPathSetInPathEnvironment: false“), so wird der Wert von „arduinoCLIDirectory“ gelesen, „export PATH=[Pfad zu Arduino-CLI]:$PATH && “ wird vor den auszuführenden Befehl gesetzt und dann wird der so erweiterte Befehl genutzt.

Der Kompilierbefehl „arduino-cli compile ...“ erstellt zu einer kompilierten Datei oder einem kompiliertem Projekt immer eine Reihe an Dateien. Mit NAME stellvertretend für die kompilierte .ino-Datei ohne der Typenendung oder dem Projektnamen ist dies die Liste an erstellten Dateien wie folgt:

- NAME.ino.eep
- NAME.ino.elf
- NAME.ino.hex
- NAME.ino.with_bootloader.bin
- NAME.ino.with_bootloader.hex

CompilationCall nutzt diesen Befehl und stellt seine Optionen so ein, dass diese in demselben Verzeichnis wie die verwendete .ino-Datei in einem eigenen Ordner namens „CompiledFiles“ abgespeichert werden. Zusätzlich wird zum späteren Nachsehen und Vergleichen die dabei genutzte Boardtypenkennung in der Textdatei „fqbn.txt“ bei den Ergebnisdateien gespeichert.
UploadCall nutzt den Befehl „arduino-cli upload …“ und greift auch auf diese Datei (fqbn.txt) zu, um selber vor dem Upload der hochzuladenden .hex-Datei in den Speicher des zu programmierenden Mikrokontrollers die Kompatibilität zu prüfen und nur bei Übereinstimmung den Vorgang durchzuführen.
Die Klasse FQBNAndCoresHandler kann zu der bei ihrer Nutzung gegeben Boardtypenkennung prüfen, ob diese der lokalen Installation bekannt ist und, falls notwendig, eigenständig die Kern-Daten dazu aus den offiziellen Quellen heraussuchen und herunterladen. Sie verwendet hierbei die Befehle "arduino-cli board listall --format yaml", "arduino-cli core update-index" und "arduino-cli core search …". Das Herunterladen wird über InstallCoreForBoards durchgeführt.
InstallCoreForBoards verwendet den Befehl „arduino-cli core install ...“ um das zuvor erwähnte Herunterladen durchzuführen.
ConnectedBoardsfinder wird von den Klassen BoardAutoSelectionAndInstallation und ListAllConnectedBoardsAction, also zwei von den Klassen für die Interaktion mit dem Nutzer/ der Nutzerin, verwendet und nutzt den Befehl "arduino-cli board list --format yaml" für das Auflisten aller angeschlossenen und für die Arduino-CLI erkennbaren Boards.

Für die Interaktion zwischen dem Nutzer/ der Nutzerin und den funktionalen Teilen auf Basis der Eclipse-GUI wurde in der Datei „plugin.xml“ eingestellt, dass bei Rechtsklick auf eine Datei mit der Endung „.zip“, „.ino“ oder „.hex“ in dem Kontextmenü der Eintrag „ArduinoCLIUtilizer“ erscheint, der zu einem Untermenü führt, dessen Einträge von der Endung der erwähnten angeklickten Datei abhängen. So soll sichergestellt werden, dass das Plugin dem Nutzer/ der Nutzerin nur dann etwas anzeigt, wenn damit potenziell etwas gemacht werden kann:

- Bei „.zip“-Dateien "Generate ArduinoCLIUtilizer config file“ und"Install zipped arduino library".
- Bei „.ino“-Dateien "List connected Arduino boards", "Generate ArduinoCLIUtilizer config file", "Compile and upload Arduino project“, "Compile Arduino project" und "Verify Arduino project".
- Bei „.hex“-Dateien "List connected Arduino boards", "Generate ArduinoCLIUtilizer config file" und "Upload.hex file".

Die Beschriftungen der soeben genannten Einträge wurden so gewählt, dass sie zu dem jeweiligen Zweck passen bzw. kurz beschreiben. Unter der grafischen Benutzeroberfläche wird jeweils eine Klasse aus dem Paket „de.ust.arduinocliutilizer.popup.actions“ aufgerufen.
Für eine Übersicht über diese Klassen und den jeweils verwendeten funktionalen Klassen siehe Klassendiagramm [TODO: Einfügen].
In fast allen Fällen ist die Programmierung dieser Klassen (die Klassen aus dem Paket „de.ust.arduinocliutilizer.popup.actions“) auf Meldungen für den Nutzer und das aufrufen von funktionalen Klassen beschränkt.

Die Ausnahme ist die Klasse BoardAutoSelectionAndInstallation, die angeschlossene Boards sucht. Wegen der einfach gehaltenen automatischen Boardauswahl bricht sie bei anderen Anzahlen an erkannten Boards mit der entsprechenden Fehlermeldung ab. Je nach Notwendigkeit lädt sie fehlende Kern-Daten zu dem angeschlossenen Board automatisch herunter.
Sie stellt sowohl den Port als auch die Boardkennung zu dem angeschlossenen Board bereit. Prinzipiell könnte diese Klasse auch als funktionale Klasse gewertet werden, aber sie führt fallabhängig Meldungen aus und intern besteht sie fast nur aus Zugriffen auf vorhandene Funktionen bzw. sie bringt keine nennenswerte zusätzliche Funktionalität ein. Deshalb wurde sie nicht zu den funktionalen Klassen einsortiert.

Wegen der Konfigurationsdatei als Anforderung für das Arbeiten mit der Arduino-CLI wurde beschlossen, keine Aktionen in der Menüleiste einzutragen.

Durch die bisher umschriebenen Abläufe und Fähigkeiten werden ANF6.2.1 und ANF6.2.2 erfüllt. Tests von dieser Implementierung haben gezeigt, dass durch die Nutzung der Arduino-CLI als offizielle Software für das Arbeiten mit Arduinocode und kompatiblen Boards und Mikrokontrollern ANF6.2.3 erfüllt wird.

[(((Neu:))))]
7.2.5: Behandlung von Boards, die ihre eigenen Boardtyp bzw. eigene FQBN nicht angeben:
In manchen Fällen geben Boards keine Informationen über ihren jeweiligen Typ an. Ein Beispiel sind die Arduino Nano Exemplare, die zur Verfügung standen, obwohl es offizielle Produkte der Firma Arduino sind. Dies zeigt sich durch die leere Liste unter „matchingboards“ (siehe Figur/Text TODO als Beispiel).

```
- matchingboards:
    - name: Arduino Mega or Mega 2560
      fqbn: arduino:avr:mega
      ishidden: false
      platform: null
  port:
    address: /dev/ttyACM0
    label: /dev/ttyACM0
    protocol: serial
    protocollabel: Serial Port (USB)
    properties:
        pid: "0x0042"
        serialNumber: 8593731333735141A130
        vid: "0x2341"
    hardwareid: 8593731333735141A130
- matchingboards: []
  port:
    address: /dev/ttyUSB0
    label: /dev/ttyUSB0
    protocol: serial
    protocollabel: Serial Port (USB)
    properties:
        pid: "0x6001"
        serialNumber: AB0LRZAC
        vid: "0x0403"
```

hardwareid: AB0LRZAC

Deshalb wurde beschlossen, dass in solchen Fällen intern einfach ein konfigurierbarer Ersatztyp angenommen wird, um so die Informationslücke aufzufüllen. In der Konfigurationsdatei wird hierfür unter „fallbackBoardIdentifierFQBN“ ein Boardtyp bzw. eine FQBN eingetragen. Im Fall der in dieser Ausarbeitung verwendeten Roboterautos sind es, wie bereits berschrieben wurde, die vorhandenen Arduino Nano Exemplare, die das beschriebene Problem aufweisen, also ist standardmäßig „arduino:avr:nano“ eingetragen. Das Bereitstellen dieser Informationen auf Anfrage ist Aufgabe der Klasse „FallbackForBoardsWithoutInternalFQBNDataHander“, die hierfür auch die Konfigurationsdatei liest. Sobald die Klassen „BoardAutoSelectionAndInstallation“ und der Pipelineschritt „LookupBoardBySerialNumber“ bei der Durchführung ihrer jeweiligen Aufgabe auf ein Board stoßen, dass keine Informationen über seinen Typen angibt, fordern sie einen Ersatztyp bei „FallbackForBoardsWithoutInternalFQBNDataHander“ an. Dann kann der jeweilige Ablauf wie normal fortgeführt werden.
Dieser Lösungsansatz funktioniert nur dann nicht, wenn gleichzeitig unterschiedliche Boardtypen angeschlossen werden, deren Exemplare ihren jeweiligen Typ nicht angeben können. Dieser Fall tritt jedoch nur unter extrem selten Umständen auf, weil eine Großzahl der Mikrokontroller den jeweiligen Typen angeben kann.
Ein alternativer Ansatz wäre eine Liste gewesen, die zu einer Board-Seriennummer den jeweiligen Typen enthält, aber dies hätte bei der Nutzerschaft mehr Aufwand bedeutet und sich nur in bereits erwähnten extrem selten Umständen gelohnt. Also wurde dieser nicht implementiert.
Falls es doch passieren sollte, so kann als Workaround für solche Fälle ein kleines Skript-Programm aushelfen.