Pyduino

[Kurzfassung 1](#_Toc123761466)

[Problem 1](#_Toc123761467)

[Lösungsidee 2](#_Toc123761468)

[Features 3](#_Toc123761469)

[Pyduino 3](#_Toc123761470)

[Programmstruktur 3](#_Toc123761471)

[Variablen 4](#_Toc123761472)

[Arrays 4](#_Toc123761473)

[Listen 5](#_Toc123761474)

[Funktionen 5](#_Toc123761475)

[Builtins 5](#_Toc123761476)

[Kontrollstrukturen 6](#_Toc123761477)

[if-Bedingung 6](#_Toc123761478)

[while-Schleife 7](#_Toc123761479)

[for-Schleife 7](#_Toc123761480)

[Umsetzung 7](#_Toc123761481)

[Transpiler 8](#_Toc123761482)

[Dank 12](#_Toc123761483)

[Quellen 12](#_Toc123761484)

# Kurzfassung

# Problem

Die Programmiersprache mit der der Arduino Microcontroller programmiert wird ist C/C++, mit einigen zusätzlichen Hilfsfunktionen, mit denen zum Beispiel die Ein- und Ausgänge des Arduinos angesteuert werden können *[Q1]*. Diese Programme können mit der Arduino Entwicklungsumgebung erstellt werden. Diese stellt Grundlegende Funktionen zur verfügung, wie zum Beispiel Syntax Highlightung und die Möglichkeit Programme zu kompilieren und auf den Arduino hochzuladen *[Q2]*. Die Arduino Entwicklungsumgebung basiert auf dem Arduino CLI, das zum Beispiel für das Kompilieruen und hochladen der Programme zuständig ist. Diese Funktionen sind aber relativ langsam, selbst das kompilieren und hochladen von einem lehren Program dauert:

|  |  |  |  |
| --- | --- | --- | --- |
| Platform *(siehe Quellen)* | Kompilieren | Hochladen | Gesamt |
| PC | 0.74s | 1.85s | 3.59s |
| Laptop 1 | 4.43s | 2.78s | 7.21s |
|  |  |  |  |
|  |  |  |  |

*(Die Zeiten wurden mit diesen Programmen ermittlet:* [*https://github.com/Bergschaf/Arduino-Benchmark*](https://github.com/Bergschaf/Arduino-Benchmark)*)*

Man kann zwar sagen, dass vor allem die Zeit, die für das Kompilieren benötigt wird, bei größeren Programmen nur leicht steigt. Die Zeit die fürs hochladen benötigt wird steigt dagegen etwas bei größeren Programmen. Diese Zeiten sind aber trotzdem sehr lang, da der Nutzer jedes mal bis zu 7s warten muss, nur um den Effekt einer kleinen Änderung im Programm zu sehen. Das liegt daran, dass die Arduino Programme kompiliert und auf den Arduino Microcontroller hochgeladen werden müssen, während zum Beispiel Python Programme von einem Interpreter ausgeführt werden, was fast ohne Zeitverzögerung funktioniert.

Die Arduino IDE (Integratet Development Environment: Entwicklungsumgebung) stellt einige Features zur Verfügung, die das Programmieren des Arduino Microcontrollers einfacher machen. Ein Beispiel dafür ist Syntax Highlighting, bestimmte syntaktische Elemente werden im Editor farbig markiert, was den Code deutlich lesbarer macht. Außerdem bietet die IDE Autovervollständigung an, die allerdings etwas umständlich ist.

Da der Arduino oft von Anfängern verwendet wird, um in die Programmierung einzusteigen ist auch die Programmiersprache ein Problem. C/C++ ist eine low-level Programmiersprache, bei der dem Nutzer weniger „abgenommen“ wird als bei high-level Programmiersprachen wie zum Beispiel Python. Das führt dazu, dass Programmieranfänger nicht nur grundlegende Programmierkonzepte, sondern auch spezielle Eigenheiten von C/C++ lernen müssen. Um zum Beispiel die länge eines Array zu ermitteln muss die Länge der Gesamten Datenstuktur in Bytes durch die Länge eines Arrayelements in Bytes geteilt werden. In Python dagegen kann einfach die len() Funktion benutzt werden. Arrays sind generell ein erschwerender Faktor bei der Programmierung des Arduinos. Solche Datenstrukturen sind in Python als verlinkte liste, bei der einfach Elemente mit verschiedenen Datentypen angehängt und entfernt werden können, implementiert. Beim Arduino dagegen können Arrays nur mit fester länge und einem Vorher festgelegten Datentypen definiert werden. Dies ist zwar deutlich schneller und Effizienter als das Python Gegenstück. Für Anfänger ist es dagegen deutlich intuitver, wenn Elemente aus Listen einfach entfernt und eingefügt werden können.

# Lösungsidee

Um diese Probleme zu lösen könnte man natürlich einfach mit einer high-level Programmiersprache wie zum Beispiel Python anfangen. Dabei bleibt aber ein großer Vorteil des Arduinos auf der Strecke. Er bietet die Möglichkeit, nicht nur theorethisch zu programmieren, sondern auch mit Elektronik zu expermientieren. Es können zum Beispiel LEDs und Sensoren angesteuert werden. Es gibt zwar Ansätze diese zwei Aspekte der Elektronik und der einfachen Programmierung zusammenzuführen. Da Python aber deutlich mehr Ressourcen benötigt als C/C++ sind solche Microcontroller wie der ESP oder sogar der Rasberry Pi deutlich teurer als der Arduino.

Deshalb ist meine Idee, eine Programmiersprache zu entwickeln, deren Syntax an Python insprierit, also möglichst einfach ist, die aber trotzdem so wenig Ressourcen verbraucht, dass sie auch auf dem Arduino ausgeführt werden kann. Um dies zu ermöglichen sollen zwar einige Vereinfachungen gegenüber Python unternommen werden, die die Programmiererfahrung aber nicht signifikant einschränken. Das soll ermöglichen, dass die Sprache von einem Transpiler in C/C++ Syntax übersetzt werden kann, der auf dem Arduino ausgeführt werden kann. Ein weiterer Vorteil der Sprache soll sein, dass sie nicht nur auf dem Arduino, sondern auch auf einem herkömlichen Computer ausgeführt werden kann. Dadurch können die Wartezeiten, bis das Program nach jeder kleinen Änderung auf den Arduino hochgeladen wird, verkürzt werden. Das wird auch dadurch ermöglicht, dass die Programme auf dem PC und auf dem Arduino parallel ausgeführt werden können. Der Programmteil, der auf dem PC ausgeführt wird hat dabei, sofern der Arduino verbunden ist, die Möglichkeit die Pins auf dem Arduino anzusteuern. Der Programmteil auf dem Arduino kann über die Verbindung dann zum Beispiel Ausgaben an den PC senden, die dann in der Konsole angezeigt werden.

Diese Verbindung gibt dem Programmierer die Möglichkeit, den Arduino nur als Steuerungseinheit, die Befehle von einem Programm auf dem PC erhält, für die Pins zu verwenden. Da so das Programm nicht jedes mal von dem Langsamen Arduino CLI kompiliert und hochgeladen werden muss können die Wartezeiten minimiert werden. Das Programm müsste nur von einem Transpiler in C/C++ übersetzt werden und dann von einem C/C++ Compiler für den PC kompiliert werden, was deutlich schneller als der Arduino Compiler ist.

Außerdem ist meine Idee eine Entwicklungsumgebung zu programmieren, die mit modernen Features wie Syntax Highlightung, Autovervollständigung und automatischer Fehlererkennung eine möglichst gute Programmiererfahrung bietet. Da es sehr schwierig ist eine eigene IDE zu entwicklen ist meine Idee, ein Plugin für die bekannte IDE Visual Studio Code von Microsoft zu entwicklen. Sie ist frei verfügbar und bietet eine gute API, die es ermöglicht mit geringem Aufwand neue Programmiersprachen zu entwickeln.

# Features

Welche Features bereits implementiert sind, kann über diesen Link eingesehen werden:

<https://trello.com/b/QFvFXErk/pyduino-features>

## Pyduino

Der Name Pyduino ist aus Python und Ardunio abgeleietet und bezeichnet die Progarmmiersprache, die ich entwickle.

### Programmstruktur

Pyduino Programme sind in zwei Teile aufgeteilt, den main-Teil und den board-Teil. Die Einzelnen Teile werden mit „#main“ und „#board“ eingeleitet.

#main

print("Hello World Pc")

#board

print("Hello World Arduino")

Ein Programm kann auch nur einen der Teile enthalten. Wenn die Funktionen des anderen Teils gebraucht werden, dann wird automatisch ein Programm auf der jeweiligen Platform gestartet, das zwar kein Pyduino Programm ausführt, aber trotzdem die Funktionen der jeweiligen Platform zur Verfügung stellt.

#main

analogWrite(11,255)

Hier wird trotzdem ein Program auf dem Arduino gestartet, dass auf Befehle vom PC (hier „analogWrite“, also „schalte LED an“) wartet.

#board

analogWrite(11,255)

Hier wird kein Program auf dem PC gestartet, da keine der Funktionen des PCs benötigt werden.

# Das ist ein Kommentar

Kommentare werden mit „#“ eingeleitet und werden vom Transpiler nicht berücksichtigt.

### Variablen

Variablen werden mit dem Datentyp und einem Namen definiert und ihnen kann ein Wert zugewiesen werden.

int x = 42

float y = 2.0

float z = x / y

string s = "Hello World"

Um Datentypen ineinander umzuwandeln können folgende Funktionen verwendet werden:

float y = 3.14

int z = int(y)

float w = float(z)

#### Arrays

Für alle dieser Datentypen können mit folgendem Syntax auch Arrays erstellt werden.

int[] a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

float[] b = [3.14,4.2]

string[] c = ["hello", "world"]

Arrays können auch nur mit der Länge ohne Werte intialisiert werden.

int[10] a  # integer array mit der Länge 10

float[4] b

string[2] c

Auf einzelne Elemente von Arrays kann mithilfe von Indices, die bei 0 begninnen, zugegriffen werden.

int[] array = [42, 15, 4, 8, 23, 16]

print(array[0]) # -> 42

print(array[3]) # -> 8

#### Listen

(Werden eventuell in Zukunft implementiert)

Neben arrays, die zwar eine schnelle Zugriffszeit (O(1)) haben, bei denen aber nicht einfach Elemente eingefügt oder entfernt werden können, gibt es Listen.

list<int> d = [1, 2, 3, 4, 5]

list<float> e = [1.0, 2.0, 3.0, 4.0, 5.0]

list<string> f = ["a", "b", "c", "d", "e"]

Listen bieten die Möglichkeit Elemente hinzuzufügen und zu entfernen.

list<int> liste = [1,2,3,4,5,6,7,8,9,10]

liste.append(42)  # 42 wird am Ende der liste angehängt

print(liste.pop())  # das letzte Element wird entfernt und ausgegeben

liste.insert(0, 42)  # 42 wird an der Stelle 0 eingefügt

liste.remove(42)  # 42 wird aus der Liste entfernt (die erste 42)

liste.index(42)  # gibt die Stelle, an der 42 in der Liste vorkommt, zurück

liste.count(42)  # gibt die Anzahl zurück, wie oft 42 in der Liste vorkommt

Auch bei Listen kann auf einzelne Elemente zugegriffen werden.

print(liste[0]) # 1

print(liste[8]) # 9

### Funktionen

Funktionen werden mit dem Datentyp, der zurückgegeben wird, dem Namen und den Parametern deffiniert. Der Programmteil, der nach der Funktionsdeffintion eingerückt wird gehört zur Funktion. Werte werden mit „return“ zurückgegeben.

int f(int x):

    return x + 1

print(f(1))

Funktionen werden mit Klammern, in denen die Paramter stehen, aufgerufen.

#### Builtins

##### print

Die print-Funktion gibt wie in Python Werte aus. Mehrere Werte können mit einem Komma getrennt werden. Bei Arrays und Listen werden die einzelenen Werte ausgegeben.

print("Hello World") # Hello World

print("Hello", "World") # Hello World

int[] a = [1, 2, 3]

print("Array", a) # Array [1, 2, 3]

Wenn diese Funktion auf dem Arduino ausgeführt wird und der Arduino mit dem PC verbunden ist, dann wird die Ausgabe in der Konsole auf dem PC angezeigt.

##### len

Die len-Funktion gibt die Länge von Arrays und Listen zurück.

int[] array = [42, 15, 4, 8, 23, 16]

print(len(array)) # 6

##### delay

Die delay Funktion hält das Program für eine bestimmte Zeit in Millisekunden an. Dabei wird aber trotzdem auf Anweisungen von jeweils Arduino oder PC gewartet, die dann während dieser Zeit auch ausgeführt werden könen.

delay(1000) # warte 1 Sekunde

##### analogWrite

Steuert die Pins am Arduino mit einem Wert von 0 - 255 an. Der Arduino, der eigentlich nur digitale Pins besitzt kann die Spannung bestimmte Pins (die mit einer Welle markiert sind) durch Pulsweitenmodulation (PWM) steuern. Dadurch kann zum Beispiel die Helligkeit von LEDs gesteuert werden.

for i in range(255):

    analogWrite(11,i)

    delay(10)

Die LED an Port 11 wird langsam heller, egal ob das Programm auf dem PC oder auf dem Arduino ausgeführt wird.

##### analogRead

Liest die Spannung an den analogen Ports des Arduino in einem Wertebereich von 0 – 1023 aus.

print(analogRead(0))

### Kontrollstrukturen

#### if-Bedingung

#### while-Schleife

Eine Schleife die ausgeführt wird, solange eine Bedingung wahr ist.

int x = 0

while x < 10:

    x = x + 1

    print(x)

Diese Schleife gibt die Zahlen von 1 bis 10 aus.

#### for-Schleife

Die for-Schleife iteriert mit einer Zählvariable über einen Bereich von Zahlen, der mit „range“ festgelegt wurde. Das Grundlegende Schema sieht folgendermaßen aus:

for Zählvariable in range(start,end,step):

    #code

Beispiele:

for i in range(10):

    print(i)

Diese for-Schleife gibt die Zahlen von 0 bis 9 aus.

for i in range(4,20,3):

    print(i)

Diese for-Schleife gibt die Zahlen von 4 bis 20 in 3er Schritten aus.

For-Schleifen können auch über Arrays und Listen iterieren.

int[] array = [42, 15, 314, 69, 20]

for i in array:

    print(i)

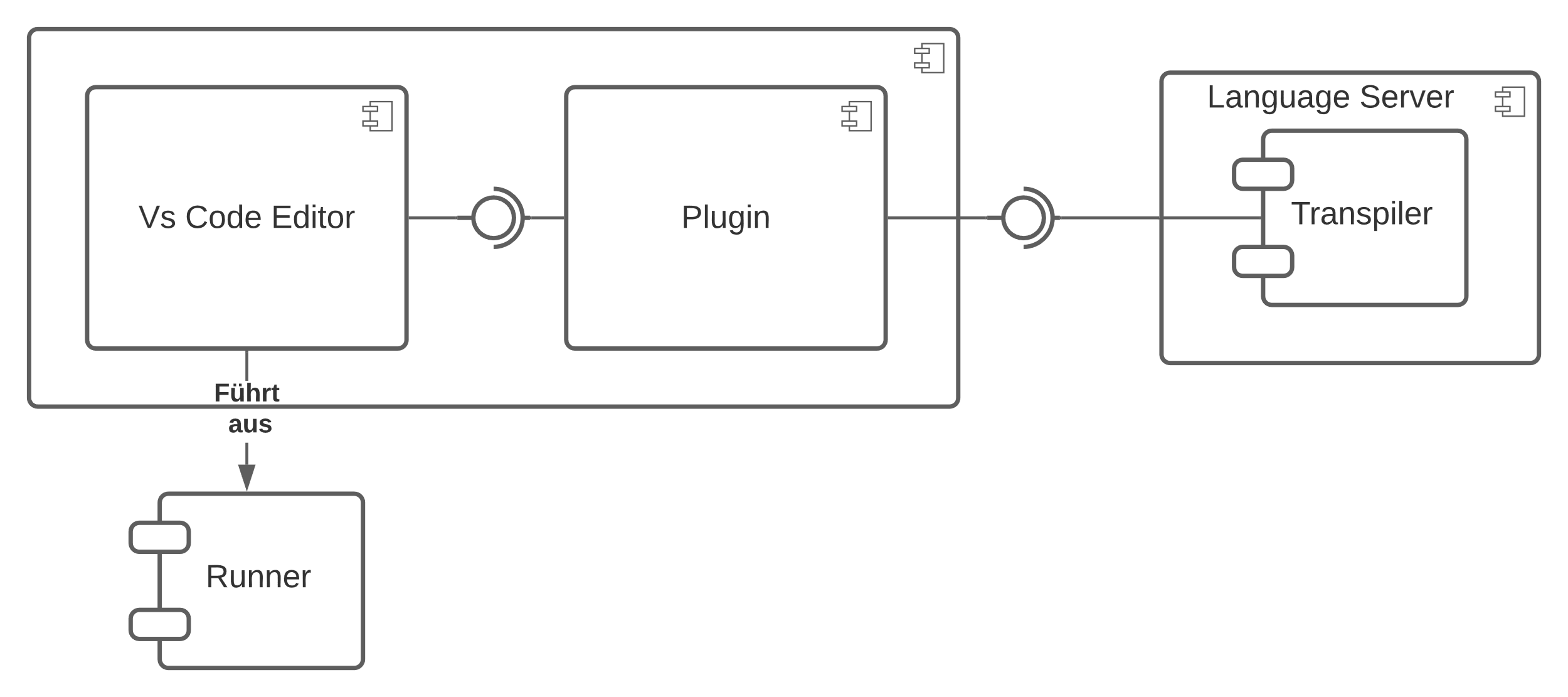
Diese for-Schleife gibt die Elemente aus dem Array einzeln aus.

# Umsetzung

Als Grundlage verwende ich die IDE Visual Studio Code. Sie stellt eine API für Erweiterungen zur Verfügung, mit der auch Unterstützung für neue Programmiersprachen implementiert werden kann. Um erweiterte Funktionen für diese Sprachen zur Verfügung zu stellen gibt es die Möglichkeit einen Language Server mithilfe von dem Language Server Protocol (LSP) zu verwenden [Qutemp1] . Die Idee hinter dem LSP ist, die Entwicklung von Features wie Autovervollständigung und Fehlererkennung für verschiedene Texteditoren so einfach wie möglich zu machen [Qutemp2]. Deshalb wird die Implementierung dieser Features von den Texteditoren und IDEs getrennt, indem nur ein Language Server für jede Programmiersprache entwickelt werden muss. Dieser Language Server kann dann über das Language Server Protocol mit einem Texteditor kommunizieren und so Features wie Autovervollständigung, Fehlererkennung und Dokumentation zur Verfügung stellen. Das bringt auch den Vorteil, dass der Language Server in einer beliebigen Programmiersprache entwickelt werden kann, unabhängig von der API des Texteditors.

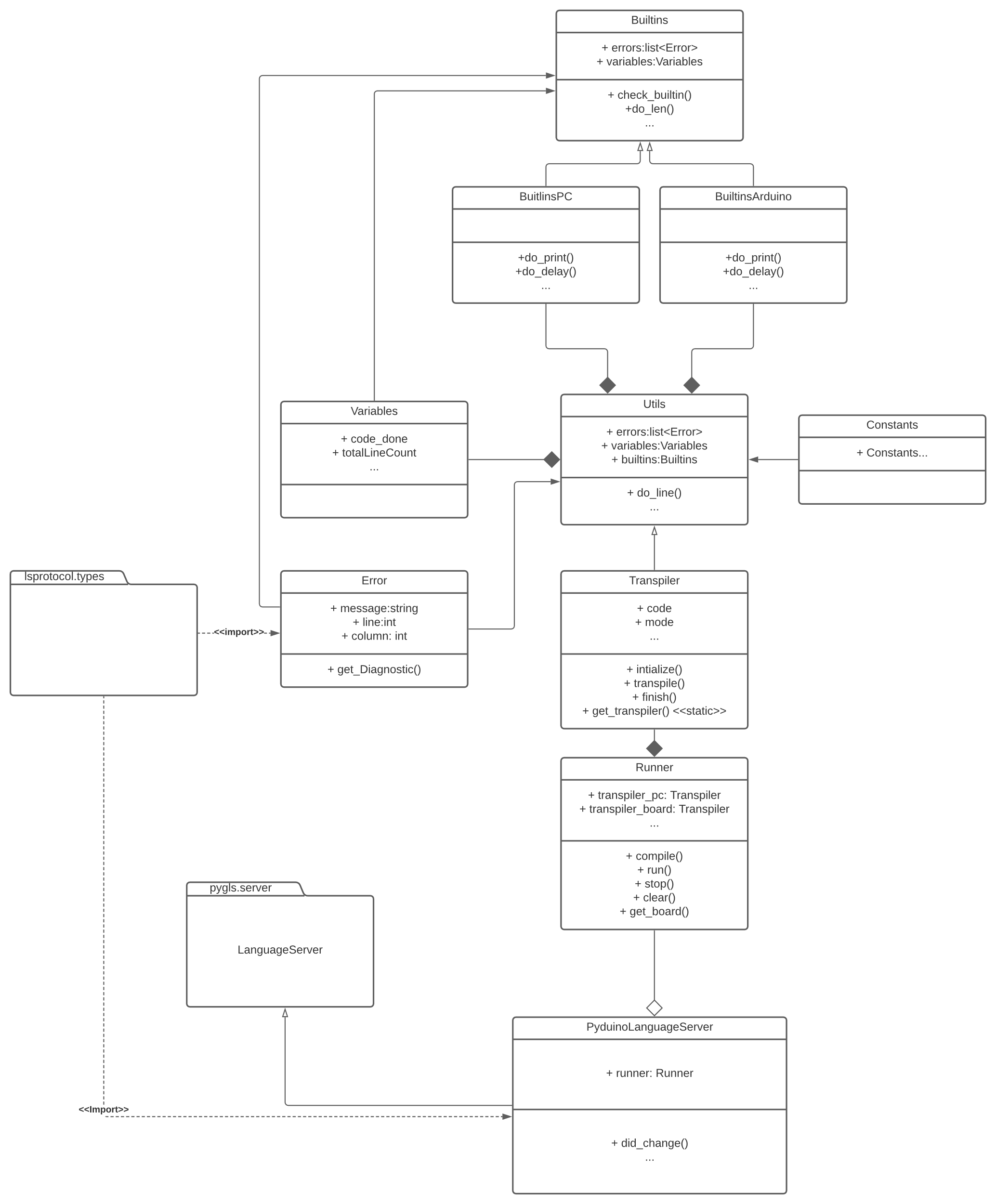
Daher ist meine Idee, mithilfe der Bibliothek pygls (<https://github.com/openlawlibrary/pygls>) einen Language Server für Pyduino in Python zu implementieren. Die Implementierten Features können dann mit einer Erweiterung in Visual Studio Code verwendet werden.

Um Pyduino Programme auszuführen wird aber auch ein Tranpiler benötigt, der den Pyduion Code in C/C++ übersetzt. Der Code muss beim Übersetzten in C/C++ auch auf Fehler überprüft werden, was eng mit den Aufgaben des Language Servers zusammenhängt. Deshalb habe ich mich entschieden, den Language Server und den Transpiler als ein Python Modul zu implementieren. Wenn ein Programm auf Fehler überprüft werden soll, wird es transpiliert und die Fehler werden gespeichert. Anhand von den Daten, die dabei über das Programm gesammelt werden, kann dann auch Autovervollständigung implementiert werden.

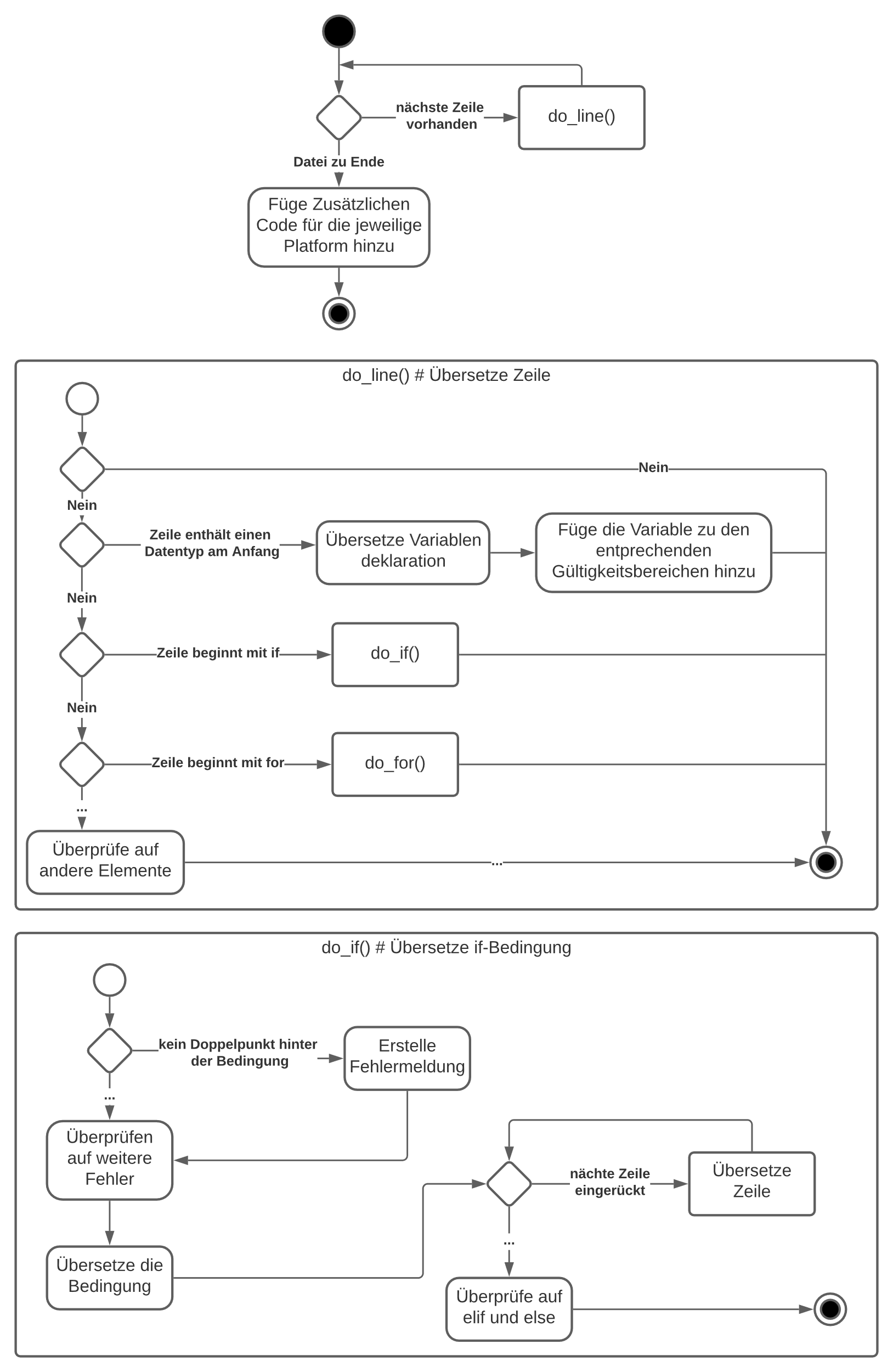


## Transpiler

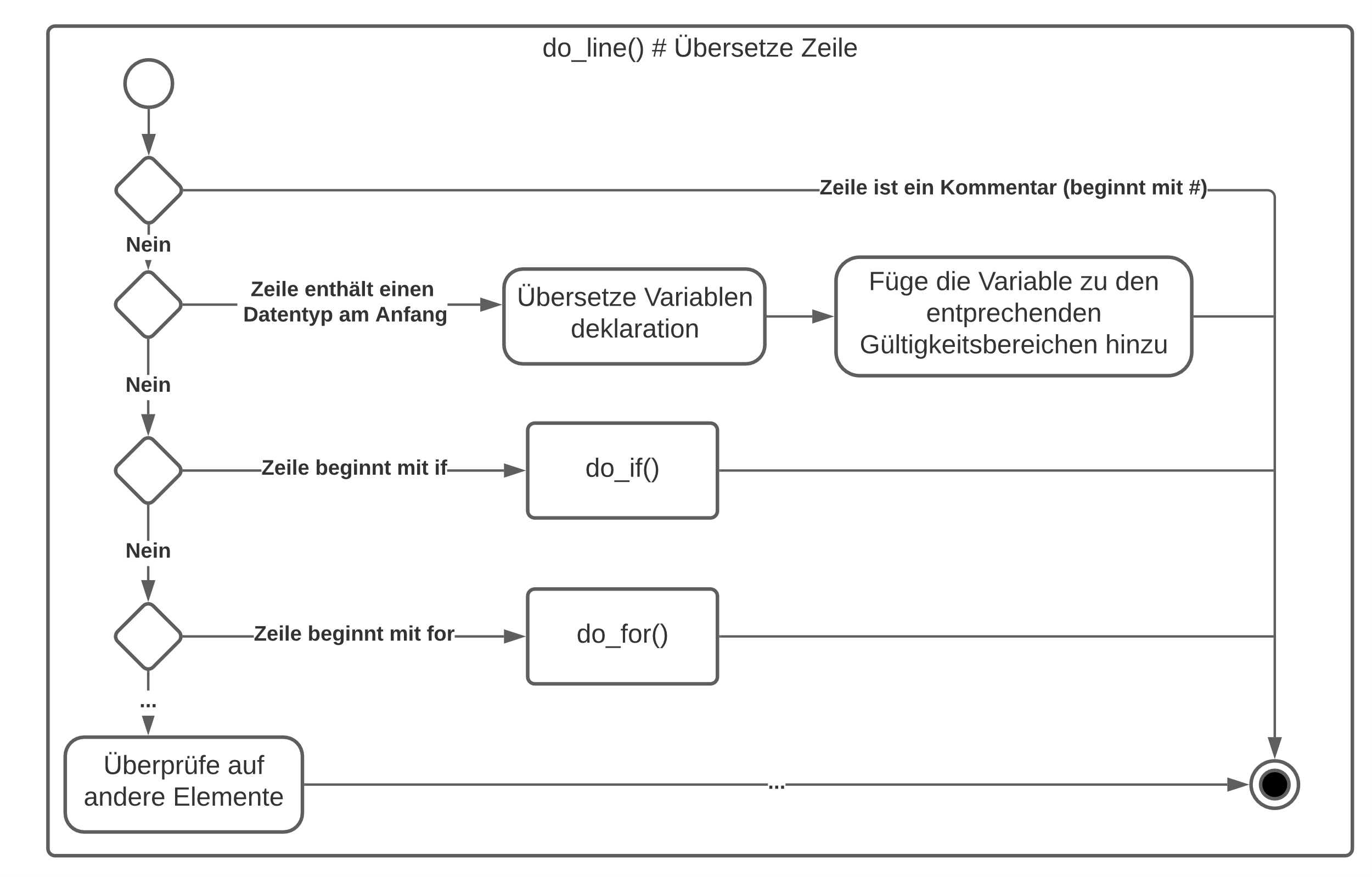
Ich habe mich dafür entschieden, den Transpiler in Python zu implementerien, da Python eine einfache und übersichtliche Programmiersprache ist, die es einfach macht zum Beispiel mit Strings zu arbeiten. Python ist zwar langsamer als Kompilierte Sprachen wie C++, die Performance reicht für den Language Server aber aus.

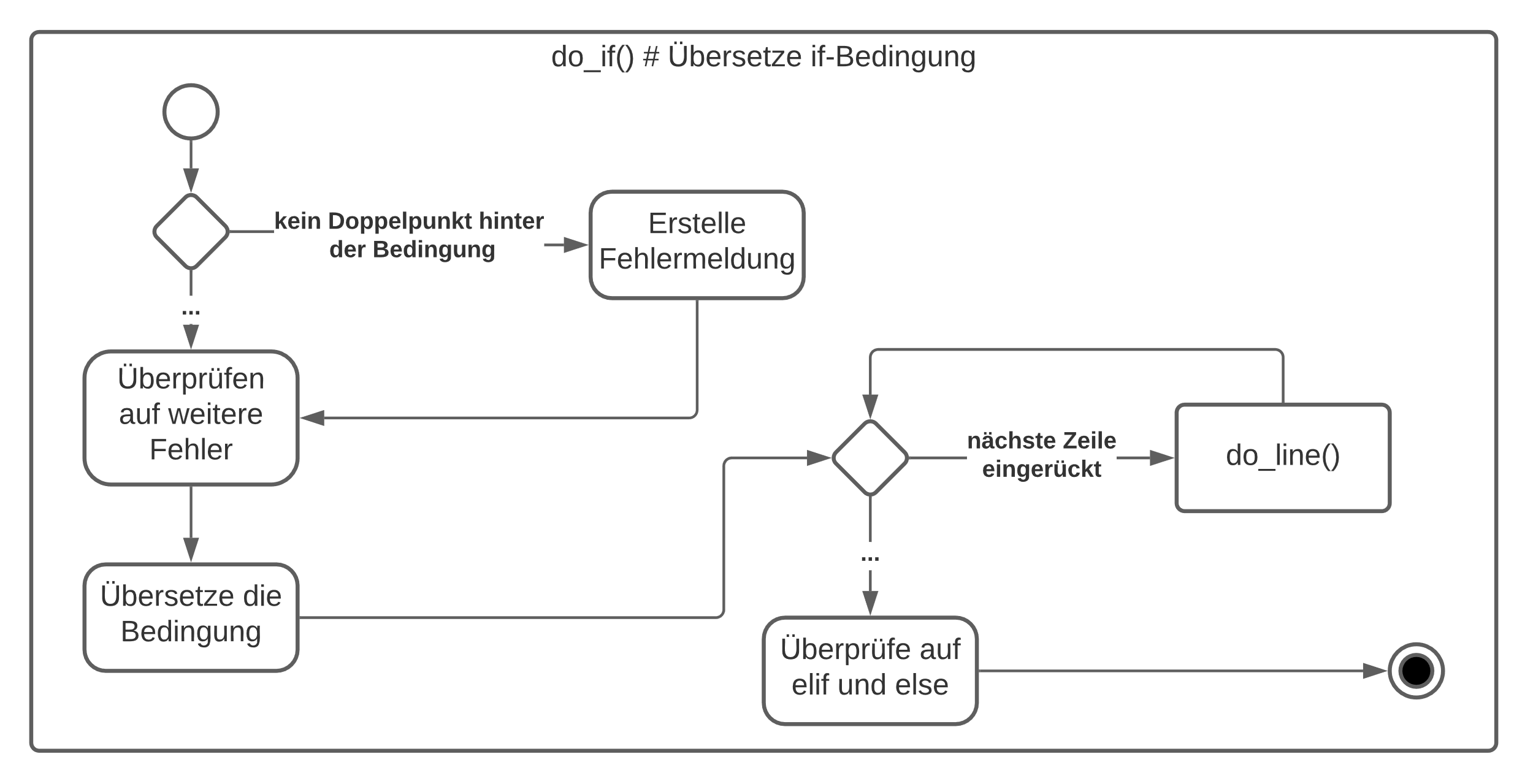


Für das Transpilieren von einem Pyduino Program zu C++ ist die Transpiler Klasse verantwortlich. Sie wird jeweils für den #main und #board Teil des Programs instanziiert. Dann erstellt jede Instanz der Klasse ein Variables Objekt, in dem wichtige Daten über das Program gespeichert sind. Es wird zum Beispiel eine Liste für den fertig übersetzten C++ Code angelegt. Außerdem wird in dem Variables Objekt ein iterator angelegt, mit dem der Code in der richtigen Reihenfolge von überall her Zeile für Zeile übersetzt werden kann. Das ist wichtig, da der Transpiler rekursiv arbeitet. Für das Übersetzten des Programs ist nach der initialisierung die transpile() Funktion zuständig. Sie funktioniert nach folgendem Prinzip:



Der Code wird mithilfe der do\_line() Funktion, die die Transpiler Klasse von der Utils Klasse erbt, zeilenweise übersetzt. Anschließend wird mit der finish() Funktion der Tranpiler Klasse der Platformspezifische Code, der zum Beispiel für die Verbindung zwischen PC und Arduino zuständig ist, hinugefügt. Dieses Diagram zeigt das Grundlegende Prinzip der do\_line() Funktion:



Die do\_line() Funktion überprüft, welchem Syntax Element oder welcher Art von Anweisung die Zeile entspricht. Dementsprechend wird dann die passende Funktion aus der Utils Klasse aufgerufen. Ein Beispiel dafür ist eine if-Bedingung. Wenn eine Zeile mit „if“ Beginnt, dann wird die do\_if() Funktion aufgerufen, die nach folgendem Prinzip arbeitet: 

Zuerst wird die Bedingung in C++ übersetzt. Anschließend werden alle Zeilen hinter der if-Bendingung, die eingerückt sind, mit der do\_line() Funktion übersetzt. Dannach wird überprüft, nach dem if-Block ein elif oder else Segment zu finden ist. Für diese Segemnte wird dann ggf. auch die Bedingung und der Eingerückte Teil übersetzt.

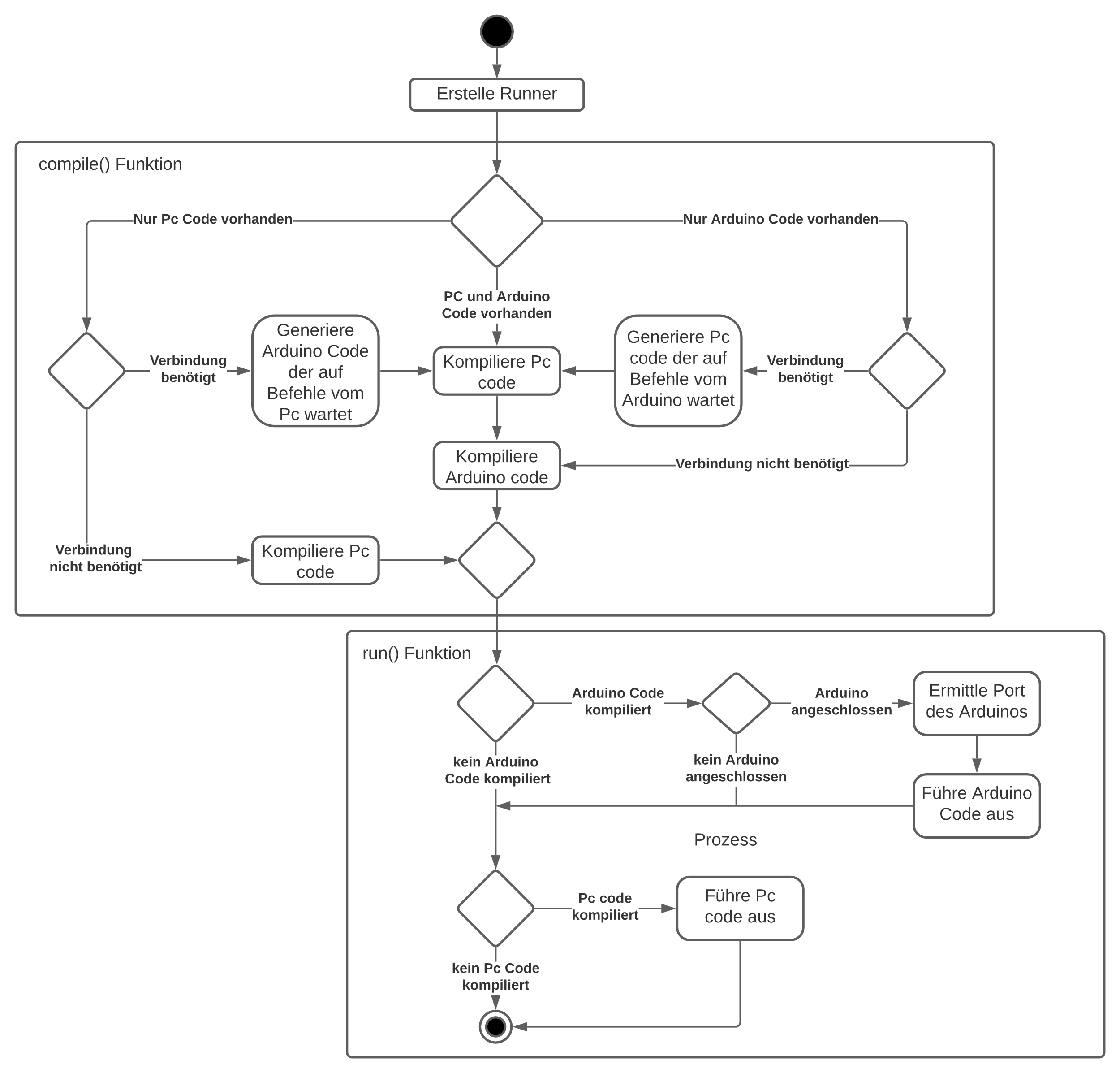
Falls beim Transpilieren Fehler entdeckt werden, wie zum Beispiel ein Fehlender Doppelpunkt nach einer if-Bedingung, werden diese gespeichert. Dafür wird ein Error Objekt erstellt, in dem die Fehlermeldung und die Zeile und Spalte des Fehlers gespeichert sind. Wenn ein Fehler entdeckt wird, wird das Programm nicht abgebrochen, sondern der Transpilierungsprozess wird fortgeführt. Das ist wichtig, da so alle Fehlermeldungenim Texteditor angezeigt werden können.

In der Utils Klasse sind Funktionen deffiniert, die für das Transpilieren der grundlegenden Programstruktur wichtig sind. Sie enthält ein Variables Objekt, in dem die Daten, die während dem Transpilieren wichtig sind, gespeichert sind und ein Builtins Objekt,

Die Builtins Klasse ist die Oberklasse für die Funktionen, die in Pyduino standartmäßig implementiert sind. Sie enthält dabei die Funktionen, deren C++ equivalent auf dem Pc und dem Arduino gleich sind. Ein Beispiel dafür ist die len-Funktion, die die Länge eines Arrays ermittelt.

Die BuiltinsPC und die BuiltinsArduino Klasse erben von der Builtins Klasse. Sie sind für die Funktionen zuständig, deren C++ Implementation sich zwischen Pc und Ardunio unterscheidet. Ein Beispiel dafür ist die print-Funktion. In der Pc-Version werden einfach die Argumente in der Konsole ausgegeben, in der Arduino version muss über die Serielle Verbindung eine Anfrage an den Pc geschickt werden. So kann dieser dann die Werte vom Arduino in der Konsole ausgeben.

Um ein Pyduino Program auszuführen wird die Runner Klasse instanziiert. Sie erhält für den #main und den #board Teil des Programs jeweils eine Instanz der Transpiler Klasse. Das Diagramm zeigt den Ablauf, um ein Pyduino Programm auszuführen:



# Dank

Benno und Matthias nicht vergessen

# Quellen

*[Q1]* <https://en.wikipedia.org/wiki/Arduino#Sketch>

*[Q2]* <https://github.com/arduino/arduino-ide>

[Qutemp1] <https://code.visualstudio.com/api/language-extensions/overview>

[Qutemp2] <https://microsoft.github.io/language-server-protocol/overviews/lsp/overview/>

PC: CPU: AMD Ryzen 5 5600X @3,7GHz 6-Cores 12-Threads; RAM: 16gb; OS: Windows 11

Laptop1: CPU: Intel Pentium N3710 @1,6GHz 4-Cores 4-Threads; RAM: 4gb; OS: Windows 10

Laptop2: