TECHNISCHE UNIVERSITÄT ILMENAU
Fakultät für Informatik und Automatisierung
Fachgebiet Integrierte Kommunikationssysteme

Hauptseminar

Recherche und Implementierung einer webbasierten Entwicklungsumgebung zur Programmierung Mikrocontroller-basierter Steuerungen im GOLDi-Remotelab

Johannes Nau, B. Sc.
Matrikelnummer 53940
Matrikel 2014

Betreuer: Dipl.-Inf. René Hutschenreuter

Ilmenau, den 30. November 2018

Inhaltsverzeichnis

1 Einleitung und Motivation

Ein Gebiet der Lehre am Fachgebiet „Integrierte Kommunikationssysteme“ ist die Wechselwirkung zwischen realen elektromechanischen Systemen und den Algorithmen, die diese steuern und regeln. Um diese auch praktisch vertiefen zu können wurde ein Remote Laboratory namens GOLDi-Labs entwickelt. Mit diesem können die Studierenden über eine Webseite reale Modelle beobachten und über diverse Steuerungsmöglichkeiten kontrollieren.

Eine Form mit den Modellen zu interagieren liegt in der Programmierung eines Mikrocontrollers. Bisher musste auf der Webseite der GOLDi-Labs eine fertig kompilierte Firmware hochgeladen werden. Dies setzt voraus, dass der Nutzer bei sich lokal eine Entwicklungsumgebung inklusive Toolchain einrichten muss. Das Konzept der GOLDi-Labs ist jedoch, dass der Nutzer das System ohne Installation weitere Software lediglich mit einem modernen Browser nutzen kann. Das bisherige System kann dies offensichtlich nicht leisten. In dieser Arbeit sollen daher Möglichkeiten recherchiert werden, die es ermöglichen die lokale Installation bei dem Nutzer durch eine Webbasiertes System ersetzen zu können. Darauf aufbauend muss eine System ausgewählt und Implementiert werden.

2 Situations- und Anforderungsanalyse

In diesem Abschnitt soll ein kurzer Überblick über das bestehende System gegeben werden. Weiterhin soll hierauf aufbauend grob die Anforderungen an das neue System vorgestellt werden.

2.1 Mikrocontrollerboard

Das Steuerungssystem dass mithilfe der hier entwickelten Lösung programmiert werden soll ist ein Mikrocontroller des Typs „ATmega644“ des Herstellers „Microchip Technology Inc.“. Dieser Mikrocontroller bildet zusammen mit der weiter nötigen Beschaltung die Erweiterungsplatine microcontroller expansion unit(MCUX), welche in Abb. 2.1 zu sehen ist. Die MCUX erweitert im Betrieb die sogenannte Base Board Unit (BBU). Die BBU sorgt hierbei dafür dass der Mikrocontroller automatisch geflasht werden kann und stellt die Kommunikationsschnittstelle für den Austausch von Sensor- und Aktordaten mit dem restlichen System da. Die Kommunikation nutzt hierbei die serielle Schnittstelle des Mikrocontrollers.

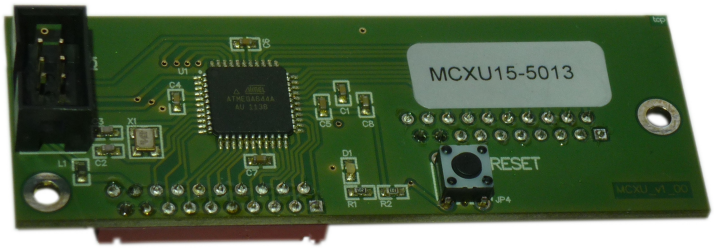

Abbildung 2.1: MCXU zum Aufstecken auf eine BBU

2.2 Programmierschnittstelle

Die Programmierschnittstelle die von der GOLDi-Webseite bereitgestellt wird funktioniert bisher wie folgt: Will der Nutzer eine Firmware hochladen, so wird diese zunächst mithilfe eines AJAX-Aufrufs an den Endpunkt „index.php?Function=UploadFile“ an die GOLDi-Webseite übertragen. Nach erfolgreicher Übertragung übermittelt der Browser des Nutzers über die bestehende Websocket-Verbindung mit dem GOLDi-Server das Steuerkommando „LoadDesign“. Dieses Kommando veranlasst den Server die zuvor vom Nutzer an die Website übermittelte Datei mittels des Endpunktes „index.php?Function=ServerGetProgramFile“ herunterzuladen und dann an die BBU weiterzuleiten, die diese Datei dann auf den Mikrocontroller lädt.

Die Zuordnung der vom Nutzer hochgeladenen Datei zu einem Experiment erfolgt dabei über die der HTTP-Anfrage enthaltenen Cookies. Der Server übermittelt in dem von ihm genutzten Endpunkt hingegen ebenfalls eine Experiment-ID die diese Zuordnung zulässt.

2.3 Anforderungen

Folgende obligatorischen Kriterien wurden für das zu entwicklende System wurden gefunden:

Req1: Das System bietet dem Nutzer die Möglichkeit der Quellcodeeingabe in einem webbasierten Editor.

Req2: Auf Wunsch des Nutzer soll der eingegebene Quellcode zur Übersetzung an einen Server zu übermittelt.

Req3: Nach dem Übersetzungsvorgang soll die Firmware an die GOLDi-Webseite übermittelt werden. Dies soll so erfolgen, dass der GOLDi-Server die Firmware auf dem selben Weg, der in Abschnitt 2.2 beschrieben ist, beschaffen kann.

Req4: Die Nutzerschnittstelle soll einfach gehalten sein und nur die nötigsten Funktionen enthalten, um insbesondere unerfahrene Nutzer nicht zu überfordern.

Req5: Es soll die Möglichkeit existieren bereits vorhanden Beispielprogramme in den Editor laden zu können.

Req6: Das Übersetzen soll von GCC-AVR-Toolchain ausgeführt werden.

Als optionale Kriterien wurde Folgendes identifiziert:

Req7: Der Editor und der dahinterstehende Server soll die eingabe von mehreren Dateien als Quellcode erlauben.

Req8: Der eingegebene Quellcode soll im Browser abgespeichert werden, sodass nach dem schließen des Editors und dem erneuten Öffnen der Quellcode noch vorhanden ist.

Req9: Der Editor soll den Nutzer mit Syntax-Highlighting und Autovervollständigung unterstützen.

3 Mögliche Lösungen

An dieser Stelle sollen verschiedene bereits existierende oder noch zu implementierende Lösungen vorgestellt und der anhand der oben genannten Kriterien bewertet werden.

3.1 Compiler Explorer

Das Projekt „Compiler Explorer“ bezeichnet sich selbst als interaktiver Compiler und besteht aus einem Web-Frontend, welches im Wesentlichen in Abb. 3.1 zu sehen ist, und einem zugehörigen Backend. Das Backend stellt über eine API diverse Compiler in unterschiedlichen Konfigurationen bereit und liefert den Asseblercode als Resultat des Compilerprozesses zurück. Das Projekt konzentriert sich hierbei auf den Vergleich der Ergebnisse verschiedene Compiler. Dies spiegelt sich in der Nuterschnittstelle wieder, die es dem Nutzer erlaubt diverse Compiler auszuwählen und deren Resultate parallel anzeigen sowie in einer Differenzansicht anzeigen zu lassen. Auf Seiten des Backends fehlt entsprechend eine Möglichkeit das Programm in der zur Ausführung benötigten maschienenlesbare Dateien.

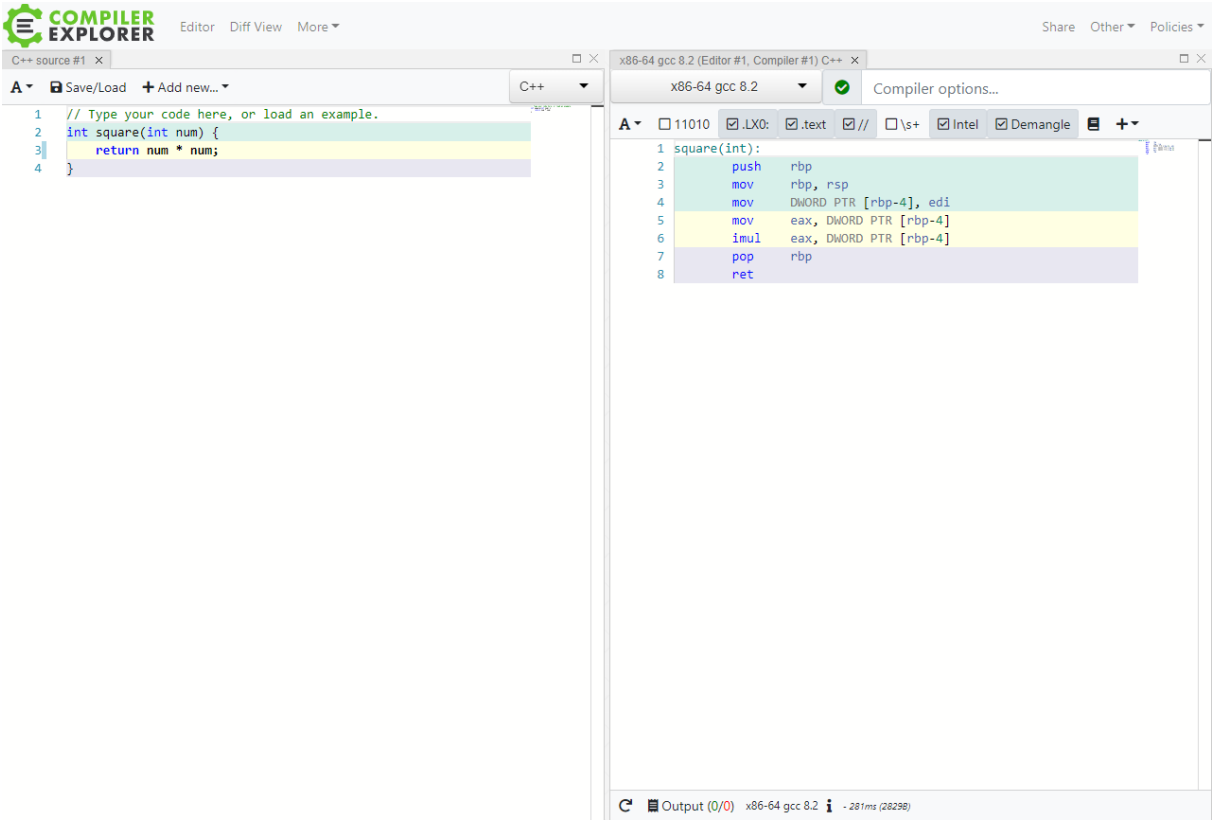

Abbildung 3.1: Frontend des Compiler Explorers

Das System erfüllt aktuell folgende Anforderungen:

Anforderung	Beschreibung	Erfüllung
Req1	Das System bietet dem Nutzer die Möglichkeit der Quellcodeeingabe in einem webbasierten Editor.	Ja
Req2	Auf Wunsch des Nutzer soll der eingegebene Quellcode zur Übersetzung an einen Server zu übermittelt.	Ja
Req3	Nach dem Übersetzungsvorgang soll die Firmware an die GOLDi-Webseite übermittelt werden. Dies soll so erfolgen, dass der GOLDi-Server die Firmware auf dem selben Weg, der in Abschnitt 2.2 beschrieben ist, beschaffen kann.	Nein
Req4	Die Nutzerschnittstelle soll einfach gehalten sein und nur die nötigsten Funktionen enthalten, um insbesondere unerfahrene Nutzer nicht zu überfordern.	Teilweise
Req5	Es soll die Möglichkeit existieren bereits vorhanden Beispielprogramme in den Editor laden zu können.	Ja
Req6	Das Übersetzen soll von GCC-AVR-Toolchain ausgeführt werden.	Nein
Req7	Der Editor und der dahinterstehende Server soll die eingabe von mehreren Dateien als Quellcode erlauben.	Nein
Req8	Der eingegebene Quellcode soll im Browser abgespeichert werden, sodass nach dem schließen des Editors und dem erneuten Öffnen der Quellcode noch vorhanden ist.	Ja
Req9	Der Editor soll den Nutzer mit Syntax-Highlighting und Autovervollständigung unterstützen.	Ja

Dass System erfüllt natürlich Req3 nicht, da es sich hierbei ja um eine dem GOLDi-Labs spezifische Schnittstelle handelt. Nichtsdestrotrotz lässt sich diese Funktionalität natürlich durch zusätzliche Entwicklungen implementieren. Die Abschätzung des Aufwands ist ohne eine Einarbeitung in des Projekt schwierig zu geben, es ist aber davon auszugehen, dass alle Softwareteile die der Extraktion des Compilat und der Kommunikation mit GOLDi neu implementiert werden müssen.

Die Benutzerfreundlichkeit (Req4) ist aufgrund der vielen Funktionen die die Webschnittstelle bietet eingeschränkt. Unter anderem ist es z.B. nicht vorgesehen mehrer Compiler zu verwenden, eine Auswahl des Compilers wäre dann überflüssig. Diese Funktionen sollten daher entfernt werden oder könnten den Nutzer verwirren. Es ist auch nötig weitere Benutzerelemente hinzuzufügen, unter anderem um das Uploaden des eingebenen Programms zu ermöglichen. Diese Punkte führen dazu, dass eine umfassende Überarbeitung des Frontends mit entsprechendem Arbeitsaufwand nötig ist.

Der Aufwand einen neuen Compiler (Req6) zu integrieren ist mit nicht mit großem Aufwand verbunden und es existiert eine Dokumentation, die das Vorgehen hierzu erläutert.

Von dem nachträglichen implementieren von Req7 sollte jedoch Abstand genommen werden, da hier tiefgehende Änderungen sowohl an Frontend, der API sowie dem Backend nötig sind.

3.2 Eclipse Che

Eclipse Che ist ein Ökosystem bestehend aus Server und Web-IDE. Dabei verwaltet Che sogenannte Workspaces, die beliebig viele Projekte und eine auf Docker basierende virtuelle Machiene beinhalten. Durch diesen Ansatz ist es im Prinzip möglich beliebige Programmiersprachen und Toolchains zu verwenden. Durch die Online-IDE benötigen die Nutzer keine zusätzliche Software mehr lokal zu installieren. Die bereitgestellte IDE, zu sehen in Abb. 3.2, bietet dabei eine vergleichbare Auswahl von Funktionen, die auch in klassischen IDEs zu finden sind.

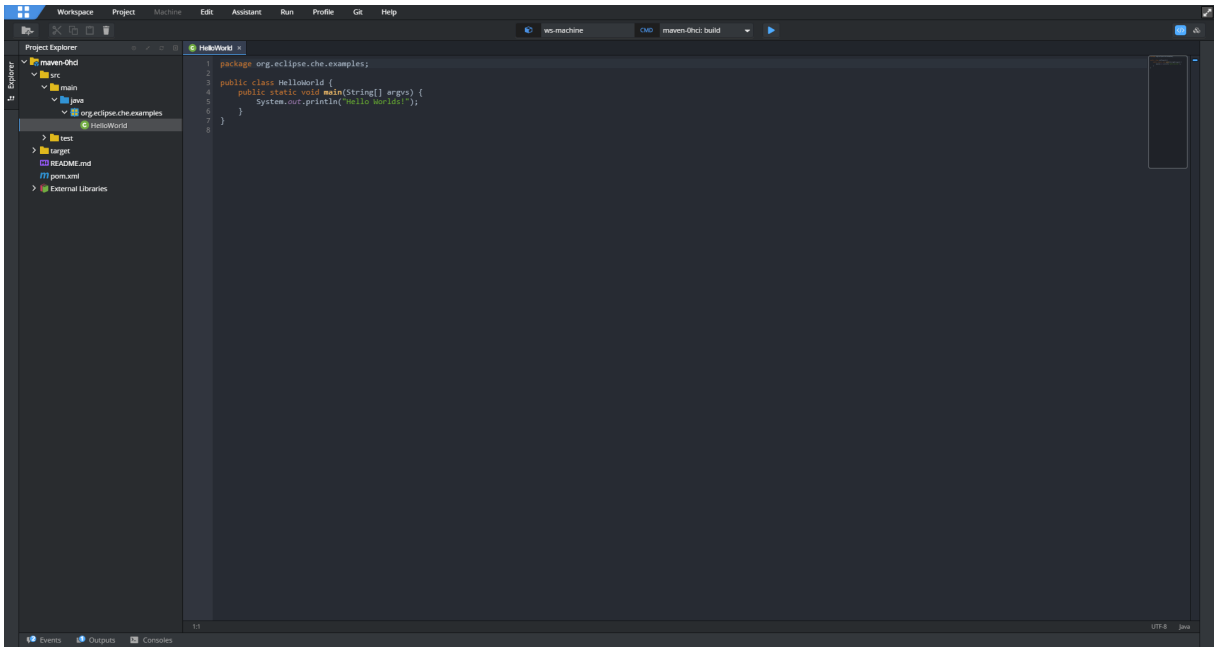

Abbildung 3.2: Frontend von Eclipse Che Lorisbachert `https://commons.wikimedia.org/wiki/File:Eclipse_Che_-_IDE_Screenshot_No_Markers.PNG`

Das System erfüllt aktuell folgende Anforderungen:

Anforderung	Beschreibung	Erfüllung
Req1	Das System bietet dem Nutzer die Möglichkeit der Quellcodeeingabe in einem webbasierten Editor.	Ja
Req2	Auf Wunsch des Nutzer soll der eingegebene Quellcode zur Übersetzung an einen Server zu übermittelt.	Ja
Req3	Nach dem Übersetzungsvorgang soll die Firmware an die GOLDi-Webseite übermittelt werden. Dies soll so erfolgen, dass der GOLDi-Server die Firmware auf dem selben Weg, der in Abschnitt 2.2 beschrieben ist, beschaffen kann.	Nein
Req4	Die Nutzerschnittstelle soll einfach gehalten sein und nur die nötigsten Funktionen enthalten, um insbesondere unerfahrene Nutzer nicht zu überfordern.	Nein
Req5	Es soll die Möglichkeit existieren bereits vorhanden Beispielprogramme in den Editor laden zu können.	Ja
Req6	Das Übersetzen soll von GCC-AVR-Toolchain ausgeführt werden.	Nein
Req7	Der Editor und der dahinterstehende Server soll die eingabe von mehreren Dateien als Quellcode erlauben.	Ja
Req8	Der eingegebene Quellcode soll im Browser abgespeichert werden, sodass nach dem schließen des Editors und dem erneuten Öffnen der Quellcode noch vorhanden ist.	Nein
Req9	Der Editor soll den Nutzer mit Syntax-Highlighting und Autovervollständigung unterstützen.	Ja

Req3 muss natürlich wieder selbst implementiert werden. Der Aufwand sollte bei der Implementierung der kompletten Toolchain nicht weiter ins Gewicht fallen.

Eclipse Che bietet versucht eine universelle und vollständige IDE zu implementieren. Dadurch werden viele Funktionen angeboten, die für den vorgesehen Zweck nicht benötigt werden und damit stören. Die Benutzerfreundlichkeit (Req4) ist aufgrund dessen sehr eingeschränkt. Eine Anpassung der IDE ist aufgrund des hiermit verbundem hohen Aufwand jedoch nicht möglich.

Req5 kann durch verschiedene Beispielprojekte implementieren werden.

Das Erstellen eines Workspaces entspricht im Grunde dem Erstellen eines Docker-Containers, der die Entwicklungsumgebung (Req6) inklusive Toolchain und Auslieferungsmöglichkeit bereitstellt. Der Aufwand ist daher nicht viel größer als die lokale Installation der Toolchain.

Von dem nachträglichen implementieren von Req7 sollte jedoch Abstand genommen werden, da hier tiefgehende Änderungen sowohl an Frontend, der API sowie dem Backend nötig sind.

Req8 wird zwar nicht erfüllt, der Quellcode wird jedoch Serverseitig verarbeitet und gespeichert was zu dem selben Ergebnis führt.

3.3 Vergleich zur Eigenimplementation

Neben dem Einsatz einer bereits existierenden Lösung bietet sich die Implementatierung einer eigenen Lösung an. Diese besteht dann aus zwei Komponenten: Die Weboberfläche und einen Server. Im Folgenden soll eine Aufwandsabschätzung gegenüber den beiden oben vorstellten Lösungen vorgenommen werden.

Die Implementatierung der Weboberfläche ist aufgrund der begrenzten Funktionalität nicht viel aufwändiger als das Anpassen der Oberfläche wie es für den Compiler Explorer nötig wäre. Dies kann vorallem dadurch erreicht werden, dass ein bereits vorhandener Editor mit Syntax-Highlighting und weiteren Vereinfachungen wie zum Beispiel „Monaco“ verwendet wird. Neben diesem Editor sind nur wenig weitere Benutzerelemente vonnöten. Lediglich ein Button zum Uploaden des Quellcodes und eine Ausgabe der eventuell auftretenen Fehler ist nötig. Die Unterstützung erfordert hingegen etwas mehr Entwicklungsarbeit, wäre aber beim Einsatz des Compiler Explorers auch noch Nachzuimplementieren.

Die Aufgabe des Servers beschränkt sich im Grunde auf das Herunterladen des Quellcodes vom Client, dem Ausführen der Toolchain, was z.B. durch ein Makefile gesteuert werden kann und dem Bereitstellen der Ergebnisse. Dies sind einfache Aufgaben, womit der Arbeitsaufwand als gering eingeschätzt werden kann. Der Aufwand das Backend des Compiler Explorers zu erweitern inklusive der nötigen Einarbeitung ist in etwas der selbe.

Aus dieser Aufwandsaufschätzung geht hervor, dass eine Eigenimplementierung wenig oder keinen Mehraufwand gegenüber dem Anpassen des Compiler Explorers darstellt. Hierdurch ist System am Ende jedoch deutlich besser in das GOLDi-Ökosystem einzubinden, womit die Eigenimplementierung vorzuziehen ist.

Der Vergleich zu Lösungen wie Exlipse Che und anderen Web-IDEs hat zwei Qualitäten: Zum einen der notwendige Aufwand zum Aufsetzen und Konfigurieren des Systems und dem Verlust einer einsteigerfreundlichen Benutzeroberfläche, welche zwangsläufig durch die komplexen und für verschiedenste Aufgaben einsetzbare Funktionen der IDEs entsteht. Das Aufsetzen eines solchen Systems kann unter Umständen auch leicht den Aufwand einer Eigenimplementierung übersteigen, was jedoch ohne genauere Einarbeitung schwer zu sagen ist.

Aus den genannten Gründen wird in diesem Hauptseminar eine Eigenimplementierung des gewünschten Systems durchgeführt.

4 Architektur und Implementierung

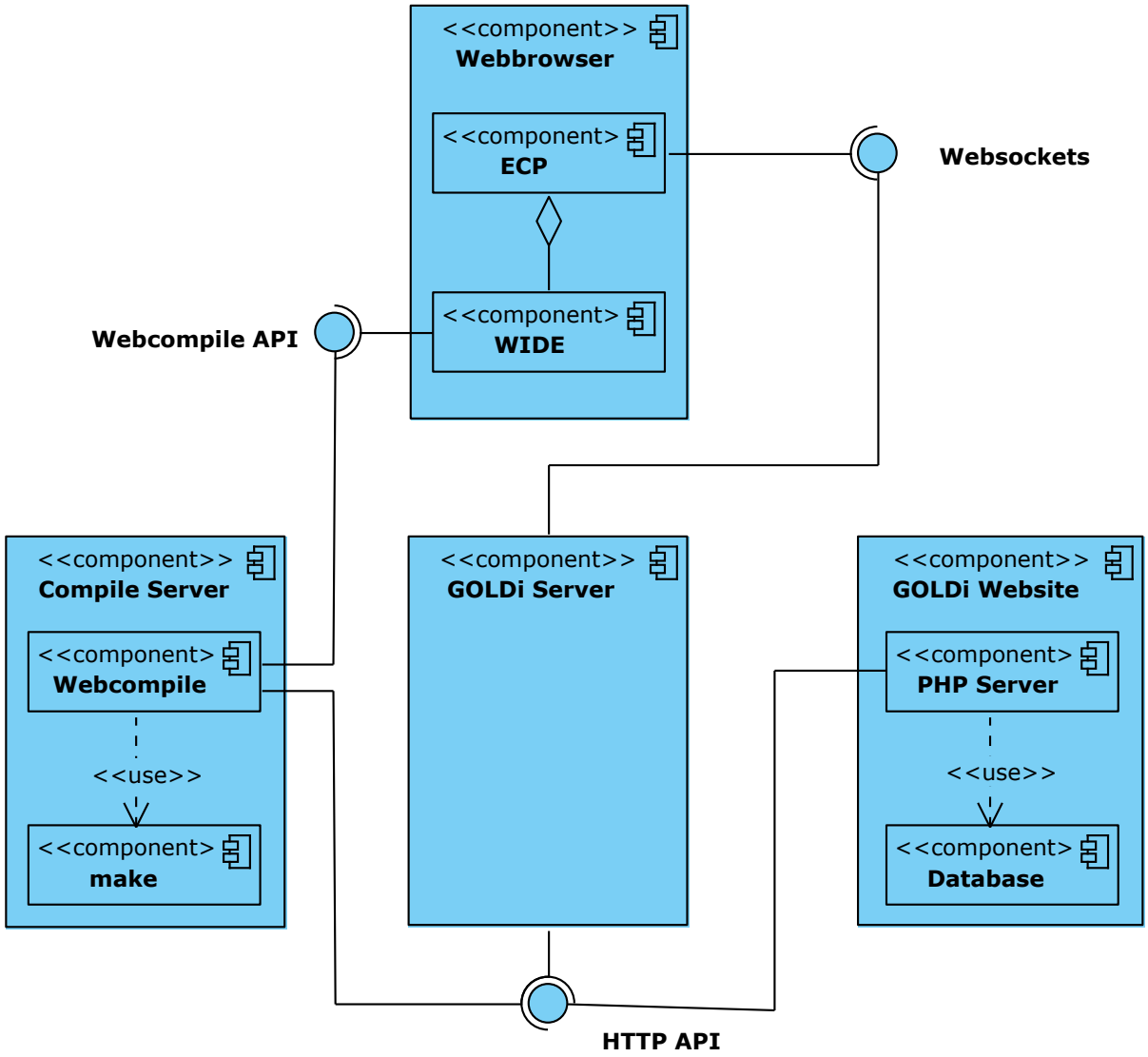

Abbildung 4.1: Verwendete Architektur

Die bestehende Architektur wird wie in Abb. 4.1 erweitert. Das ECP, welches im Webbrowser abläuft wird um die Web Integrated Development Environment (WIDE) erweitert. Diese ist zwar als Teil des ECP anzusehen soll aber bis auf die nötigsten Integratione unabhängig von der ECP entwickelt werden. WIDE greift mittels der Webcompile API auf den Compile Server zu um den eingegebenen Quellcode zu kompilieren. Sollte der Kompiliervorgang mittels des Tools make erfolgreich sein kann der Compile Server die Binärdatei auf die GOLDi Website übertragen. WIDE kann daraufhin als Teil des ECPs mittels der Websockets-Verbindung zum GOLDi-Server den Programmiervorgang auslösen, der die Datei von der GOLDi Website downloaded.

Für ein besseres Verständins des Ablaufs ist Abb. 4.2 zu betrachten: Im ersten Schritt wird vom Nutzer das Kompilieren und Hochladen des Quellcodes ausgelöst. Dies führt dazu, dass WIDE alle vorhanden Dateien als JSON-Objekt an Webcompile überträgt. Dieser speichert die Übertragenen Daten zunächst temporär lokal ab, und versucht ein vordefiniertes Makefile auszuführen (2.2). Wenn dies erfolgreich ist wird des Kompilierresultat an den PHP-Server übertragen (2.3), der diese in der GOLDi-Datenbank abspeichert. Webcompile informiert WIDE nun über den

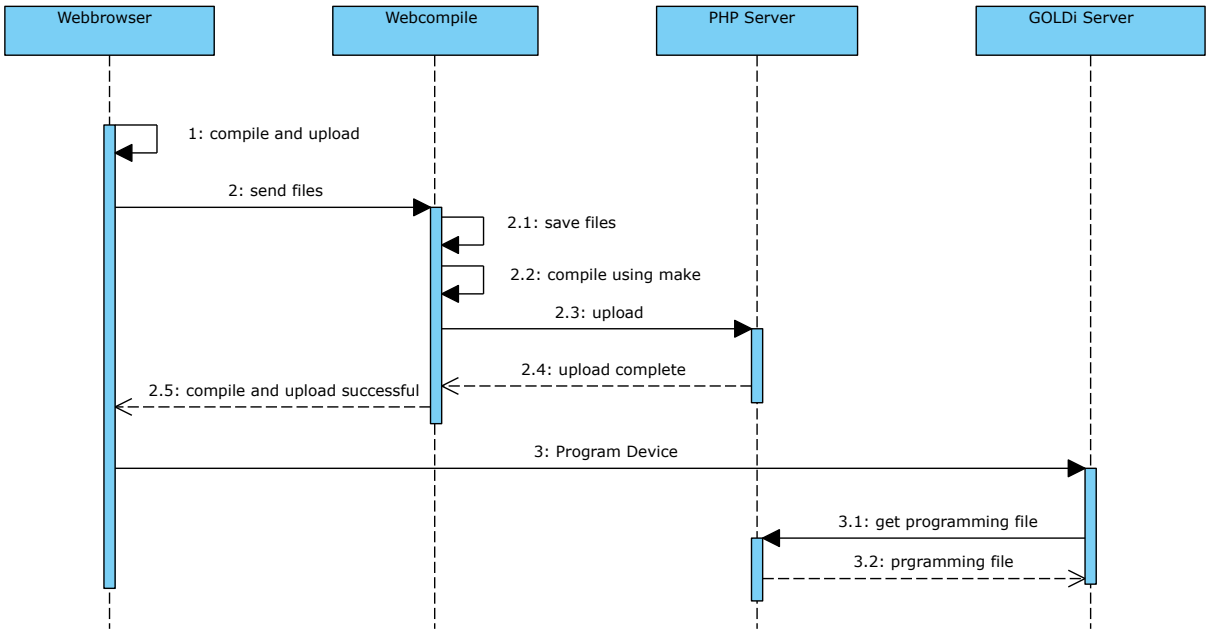

Abbildung 4.2: Ablaufdiagramm der Architektur

Erfolg der Anfrage (2.5), woraufhin WIDE direkt mit dem GOLDi-Server kommuniziert und ein Programmiervorgang auslöst (3). Der GOLDi-Server läd nun die zuvor hochgeladene Datei von dem PHP-Server herunter, um hiermit den Mikrocontroller programmieren zu können.

Die Implementierung von Webcompile ergibt sich direkt aus der obigen Beschreibung. Im Grunde wird der hier dargestellte Ablauf sequentiell abgearbeitet. Bei Auftreten von Fehlern wird die Verarbeitung abgebrochen und eine Fehlermeldung WIDE zurückgegeben.

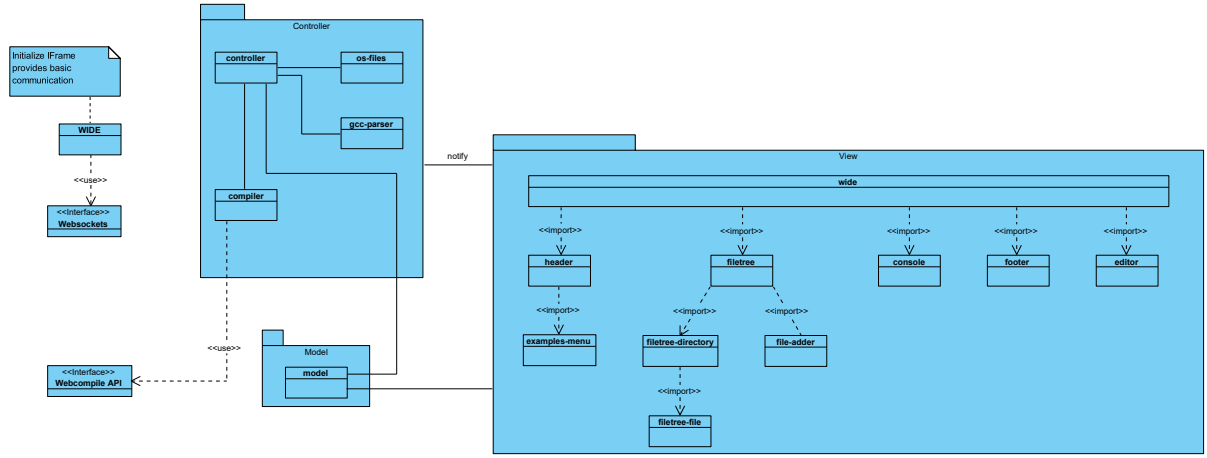

Abbildung 4.3: Klassendiagramm von WIDE

Die Implementierung von WIDE bedient sich der Model-View-Controller-Architektur wie in Abb. 4.3 zu sehen ist. Als Einsprungpunkt aus der ECP heraus wird zunächst in die Klasse „WIDE“ gesprungen. Diese Klasse initialisiert die nötigen HTML-Elemente um die tatsächliche Applikation anzeigen zu können. Da die Applikation mittels IFrame eingebunden wird, ist die Aufgabe dieser in der ECP ablaufenden Klasse ebenfalls die Kommunikation mit der Applikation zu übernehmen.

Die eigentliche Applikation besteht nun aus dem Controller, der Hauptsächlich aus der gleichnamigen Klasse besteht. Der Aufruf der Webcompile-API wird in der Klasse „compiler“ gekapselt. Das Resultat wird dann von der Klasse „controller“ interpretiert. Hierzu existiert noch die Hilfsklasse „gcc-parser“, die das Parsen der Fehlerausgabe von gcc übernimmmt. Eine weiter Hilfsklasse an dieser Stelle ist „os-files“. Diese Klasse stellt Methoden zur Verfügung um Datein auf dem Computer des Nutzer zu speichern, bzw. von diesem Dateien einzulesen.

Sämtliche Informationen, die durch den Controller verarbeitet werden werden in dem Datenmodell aus der Klasse „model“ gespeichert. Diese Information werden von den Klassen, die der View zugeordnet sind dargestellt. Sämtliche Aktionen, die der Nutzer auslöst werden an den Controller zurückübermittelt, der dann darauf reagiert. Eine Manipulation des Datenmodells aus den Views heraus findet nicht statt. Eine Ausnahme stellt hier lediglich die Klasse editor da. Da hier der Quellcode bearbeitet wird, werden Änderungen daran auch direkt in das Datenmodell syncronisert. Der Controller erhält dann nur die Nachricht, dass sich der Dateninhalt geändert hat.