Eigenständige MUSCLE Device API

Studiengang: BSc in Mikro- und Medizintechnik | Vertiefung: Mechatronik
Betreuer: Prof. Andreas Habegger
Experte: Rico Zoss (Wabtec)

Die Software-Lösung MUSCLE (Muscle Unified System for Controlling Laboratory Equipment) ermöglicht es, mittels einer Webapplikation verschiedene handelsübliche Systeme zu steuern, Daten zu erfassen und zu visualisieren. Dass der Benutzer dieses Anwenderprogramm benutzen kann, muss er seine Implementierung gerätespezifisch im MUSCLE Framework einbinden. Ziel dieser Arbeit ist es, diese starre Kopplung aufzutrennen, um die Fehlerresistenz und Wartbarkeit zu erhöhen.

Ausgangslage

Mit MUSCLE wurde eine Grundlage geschaffen, eine einfache, aber dennoch leistungsfähige Umgebung zur Steuerung und Überwachung für nahezu jedes Gerät zu realisieren. Das MUSCLE-Framework enthält ein Frontend als Single Page Webapplikation, eine Backendkomponente und die Endgeräte. MUSCLE stellt ein Gerüst zur Verfügung, wo der Benutzer sein Programm im Endgerät direkt implementieren muss. Dies ist nicht nur aufwändig für den Benutzer, sondern auch anfällig auf Fehler wie auch umständlich bei Updates von MUSCLE. Diese starre Kopplung soll nun mittels einer API auf dem Endgerät aufgetrennt werden. Dem Entwickler wird so eine Schnittstelle geboten, welche das Benutzerprogramm vom MUSCLE-Framework trennt.

Methoden

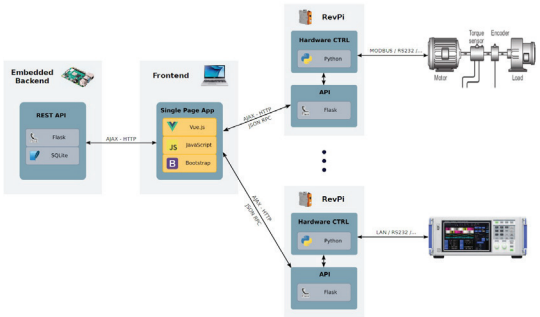
Die generelle Aufgabe der API ist es, Informationen zwischen der Flask API und dem Benutzerprogramm auszutauschen. Als zuverlässige Verbindung wird auf eine Netzwerkkommunikation via Sockets gesetzt. Diese Kommunikation besteht aus einer Server- und Clientkomponente. So kann das Master-Slave Prinzip angewendet werden, wobei der Server als Master agiert. Die Nachrichten werden als Strings im JSON-Format gesendet, da die Flask API mit derselben Kommunikation arbeitet. So können entsprechende Nachrichten direkt weitergeleitet werden. Für den Endbenutzer wird neu eine C-Library zur Verfügung gestellt, welche einen breiten Einsatzbereich aufweist, so dass diese auch z.B. in Python Anwendungen eingebunden werden kann. In dieser Bibliothek werden folgende Aufgaben umgesetzt:

- Bilden der Frontend-Option im JSON Format
- Parsen der angekommenen JSON-Nachrichten und weiterleiten der entsprechenden Daten an die dazugehörige Option mittels vom Endbenutzer programmierten Callback-Funktion
- Realisierung sämtlicher Socket Kommunikation

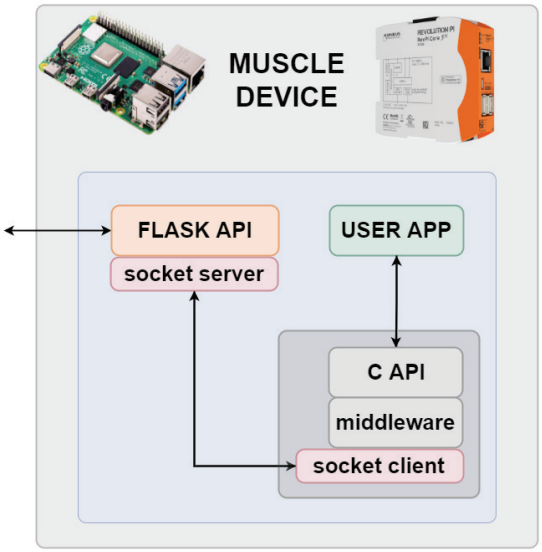
Resultate

Die Bibliothek ist in mehreren Softwareabstraktionsschichten aufgebaut. Diese haben das Ziel, mehr Flexibilität oder ein einfacheres Handling zu ermöglichen. Die vom Benutzer gewünschten Optionen im Frontend werden zuerst in Strukturen abgespeichert. Der Socket-Server wird direkt in die bestehende Flask API integriert. Zum Schutz des mehrmaligen aufstarten des Servers mit der selben Adresse und dem selben Port wird dieser als Singeltonklasse implementiert.

Adrian Steiner
adi.steiner@hotmail.ch



Übersichtsdiagramm MUSCLE Framework



Erweiterung MUSCLE Device zur Entkoppelung der USER APP