SenPi - Smart Environment

Adrian Herrmann, Niclas Kristek, Iman Nouri-Namini, Thomas Tegethoff

28. September 2017

1 Einführung

Das Ziel des SenPi Smart-Environment-Projekts ist die Sammlung und Visualisierung verschiedener Sensordaten. Zu diesem Zweck wird ein Board mit integrierten oder extern angeschlossenen Sensoren mit einer RIOT-basierten Applikation bestückt, das die Daten der angeschlossenen Sensoren sammelt und an ein Raspberry Pi verschickt, welches die Daten in einer Datenbank speichert, die einem Webserver zur Verfügung steht. Zusätzlich können Sensordaten über an das Board angeschlossene LED-Streifen visualisiert werden. Es wird ein Phytec phyNODE KW22 benutzt, aber die Anwendung ist so geschrieben, dass beliebige Boards benutzt werden können, auf denen RIOT läuft. In diesem Bericht wird der Aufbau des Projekts beschrieben und Entscheidungen zur Architektur erläutert sowie auf Fehlschläge und noch anstehende Herausforderungen eingegangen.

2 SenPhy

Die SenPhy-Anwendung ist RIOT-basiert. Die Wahl von RIOT als IoT-Betriebssystem war naheliegend, da RIOT von vielen FU-Mitarbeitern mitentwickelt wird (inklusive der Betreuer des Softwareprojekts) und somit direkte Unterstützung vorhanden war. SenPhy liest die Werte der angeschlossenen Sensoren aus und überträgt sie auf Anfrage an den Server. Auch wenn die Anwendung theoretisch auf allen Boards, die von RIOT unterstützt werden, laufen sollte, wird primär das Phytec-phyNODE-KW22 verwendet, weil es bereits über eine Vielzahl vormontierter Sensoren verfügt. Bei diesen Sensoren handelt es sich um:

1. Thermometer
2. Luftfeuchtigkeitssensor
3. RGB-Lichtsensor
4. Magnetometer
5. Gyroskop
6. Drucksensor
7. kapazitiver Taster

Lediglich die Werte des kapazitiven Tasters werden nicht für Abfragen zur Verfügung gestellt, da es für ihn keinen Treiber gibt, der die SAUL-API implementiert und weil er streng genommen keine Umweltwerte darstellt. Die SAUL-API stellt eine einheitliche Schnittstelle für Gerätetreiber in RIOT dar und ermöglicht somit eine einfache Austauschbarkeit und Erweiterbarkeit von Sensoren.

Bei der Bereitstellung der Ressourcen trat ein Bug im 6LoWPAN-Treiber für die Phytec-Boards zutage. Übersteigt die Länge einer Nachricht die maximale Länge eines 6LoWPAN-Pakets, muss das Paket fragmentiert werden. Allerdings verschicken die Treiber nur das erste Fragment und alle restlichen nicht. Aufgrund dessen musste darauf geachtet werden, dass die Pakete ausreichend klein sind, dass eine Fragmentierung ausbleibt. Die Anzahl der bereitgestellten Ressourcen musste deswegen begrenzt werden, weil alle Ressourcen, die ein Board bereitstellt, mittels einer einzelnen Abfrage durch den Server ermittelt werden und die Antwort auf diese Abfrage nicht zu lang werden darf. Deswegen werden nur die Werte des Thermometers und der Luftfeuchtigkeitssensors bereitgestellt. Die Handler-Funktionen für die anderen Sensoren sind zwar implementiert, aber von außen nicht erreichbar.

Ursprünglich sollten auch noch CO_2-Werte mittels eines entsprechenden Sensors (MH-Z14A NDIR von iHaospace) bereitgestellt werden. Dieser Sensor sollte an ein anderes Board (Atmel Samr21-Xpro) angeschlossen werden, um die Vielseitigkeit von RIOT und die gute Hardware-Abstraktion durch RIOT zu verdeutlichen. Da es allerdings zu Lieferschwierigkeiten für diesen Sensor kam, wurde die Integration in das Projekt nicht mehr umgesetzt.

Damit die Anwendung auf dem Raspberry Pi die Boards finden kann, senden die Boards regelmäßig ein Announcement-Paket. Dieses Paket enthält nichts weiter als die App-ID „riot-swp-2017-se“. Server für diese Anwendung sind so in der Lage, Boards für die Anwendung zu erkennen. So wird verhindert, dass Anfragen an Boards geschickt werden, die in derselben räumlichen Umgebung arbeiten, aber keine Daten für die Anwendung dieses Projekts bereitstellen.

Die bereitgestellten Sensorwerte werden mittels *Constrained Application Protocol* (CoAP) für den Server als Ressource bereitgestellt. Konkrekt wurde dafür die CoAP-Implementierung *gcoap* aus der RIOT-Bibliothek genutzt. Wenn der Server einen CoAP-Request für eine dieser Ressourcen sendet (z.B. */se-app/temp*), wird:

1. mittels SAUL der aktuelle Wert des Sensors ausgelesen
2. der Wert in eine mit JSON formatierte SenML-Nachricht verpackt
3. die Nachricht mittels einer CoAP-Response an den Server übertragen.

An dieser Stelle kam es zu Verzögerungen in der Entwicklung. Aufgrund einer Undeutlichkeit in der Dokumentation zu gcoap wurden die CoAP-Requests nicht korrekt erstellt. Für den zweiten Paramter `len` der Funktion `gcoap_req_send2` ist als Beschreibung „length of the buffer“ angegeben. Diese Länge bezieht sich auf den ersten Parameter `buff`. Allerdings ist nicht die Länge des für den Buffer reservierten Speichers gemeint, sondern wie viel von diesem Speicher konkret belegt ist. Dies wurde falsch verstanden und die Gesamtlänge des Speichers für den Buffer angegeben. Die Funktion `gcoap_req_send2` füllte den nicht belegten Speicher mit '/' auf, wodurch der Request keine vorhandene Ressource mehr abfragte. Schwierig beim Auffinden dieses Fehlers war, dass gcoap auch bei eingeschalteter Debug-Funktionalität keine Fehlermeldung ausgibt, wenn es einen Request empfängt, den es nicht bearbeiten kann. Deswegen wurde zuerst vermutet, dass kein Response gesendet wird, bzw. gcoap falsch initialisiert wurde.

Aufgrund des oben bereits erwähnten Bugs in den 6LoWPAN Treibern der Phytec-Boards durfte die Größe einer Response-Nachricht die maximale Größe eines einzelnen 6LoWPAN-Pakets nicht überschreiten. Deshalb wurde darauf darauf verzichtet, die Seriennummer der Boards als `name`-Attribut in die SenML-Nachrichten einzufügen.

Zusätzlich sollten die Temperatur- und Luftfeuchtigkeitswerte noch über einen WS2812B-LED-Streifen dargestellt werden. Ein Streifen sollte 50 LEDs lang sein, sodass eine LED entweder:

- 2 % Luftfeuchtigkeit darstellen sollte

- 1 °C darstellen sollte, wobei die Skala von −10 °C bis 40 °C gehen sollte

Mittels eines einfachen Tasters sollte auf den jeweils andere Wert gewechselt werde

Die Entwicklungsarbeit für diese LED-Streifen war allerdings nicht erfolgreich. Die Probleme waren folgende:

1. Es gibt mindestens zwei Datenblätter für die WS2812B-LED-Streifen, die unterschiedliche Timings für das Ansteuern der LEDs angeben (siehe Anhang). Die Datenblätter hatten weder eine Versions- oder Revisionsnummer noch ein Datum. Auf Grund dieser fehlenden Informationen konnte nicht festgestellt werden, welches das korrekte Datenblatt ist.

2. Beide Timings wurden implementiert, jedoch funktionierte keines.

Dass die Verlötungen etc. alle korrekt sind, ist daran zu erkennen, dass die LEDs kurz aufleuchten, wenn sie an eine Stromquelle angeschlossen werden. Da keine zuverlässige Funktionalität hergestellt werden konnte, liegt der Treibercode für die WS2812B-LED-Streifen dem Projekt zwar bei, wurde aber nicht integriert.

3 SenPi

Der „Server“ auf dem Raspberry Pi empfängt die regelmäßigen Announcement-Pakete der SenPhy-Anwendung und speichert diese Geräte in der Datenbank, sie werden quasi „registriert“.

Mit einer Konfigurationsdatei werden die Zugangsdaten zur SQL-Datenbank, Polling-Intervall und weiteres festgelegt. In diesem definitierten Polling-Intervall werden alle in der Datenbank vorhandenen Geräte nach den mit CoAP freigegebenen Ressourcen abgefragt. Daraufhin werden die Sensordaten dieser Ressourcen abgefragt und in der Datenbank gespeichert.

Die Architektur der in Golang geschriebenen Anwendung ist in einzelne kontextbezogene Module aufgeteilt. Die einfache Testbarkeit, gute Lesbarkeit, Performance und leichtgewichtige Nebenläufigkeit macht Golang zu einer hervorragenden Sprache für eine Serveranwendung. Infolgedessen wurden Unit-Tests geschrieben und nebenläufig entwickelt. Es wurde versucht, die Referenzen der vier vorhandenen Komponenten untereinander zu minimieren.

1. Das Modul *main* ist der Startpunkt der Anwendung und startet die einzelnen Services.

2. Das Modul *db* ist für die Verbindung zur Datenbank zuständig.

3. Das Modul *config* ist dafür zuständig, die Konfigurationsdatei auszulesen und die angegebene Konfiguration als Datenstruktur bereitzustellen.

4. Das Modul *riot* ist für die Kommunikation mit SenPhy zuständig und referenziert dafür das Modul *db* um die erhaltenen Daten zu persistieren.

4 Webanwendung

Die Webanwendung liest die von SenPi in die Datenbank eingefügten Daten aus und plottet sie auf mehrere Charts. Die Charts werden beim Aufruf der Website dynamisch erstellt, abhängig davon, für welche Sensoren es Einträge in der Datenbank gibt. Es wird für jeden Sensor und für jede Dateneinheit ein Chart erstellt. Die Charts werden mit Google Charts erstellt. Andere Kandidaten waren Chart.js und Highchart. Chart.js wurde verworfen, weil es nicht in der Lage war, eine dynamische Anzahl Charts auf einer Seite zu plotten. Für jedes Chart musste bereits vor Durchlauf des Skripts ein Prototyp im Seitenquelltext vorhanden sein. Highchart wurde nur

versuchsweise benutzt, aber letztendlich nicht verwendet, weil es in der Vollversion kostenpflichtig ist und in alle Charts ein Wasserzeichen einfügt.

Aufgrund der Lizenzbestimmungen von Google Charts muss der Code extern eingebunden werden. Die Daten werden mit einem PHP-Skript aus einer MySQL-Datenbank gelesen und als JSON-String zurückgegeben. Dieser JSON-String wird dann von einem JS-Skript ausgelesen und die Daten mittels Google Charts visualisiert. Die Entscheidung für PHP fiel aufgrund bereits vorhandener Kenntniss dieser Programmiersprache. Die Konfiguration der Datenbank, d.h. Adresse, Datenbankname und Authentifizierungsinformationen, sind auf eine eigene PHP-Datei ausgelagert. Weiterhin wurde Bootstrap für die Website benutzt, um eine ansprechende Darstellung zu erreichen.

5 Zusammenfassung

SenPi fragt die Sensorwerte von SenPhy ab und speichert sie in einer Datenbank. Die Webanwedung liest diese Daten aus und stellt sie graphisch dar. Die groben Interaktionen sind in Abbildung 1 dargestellt.

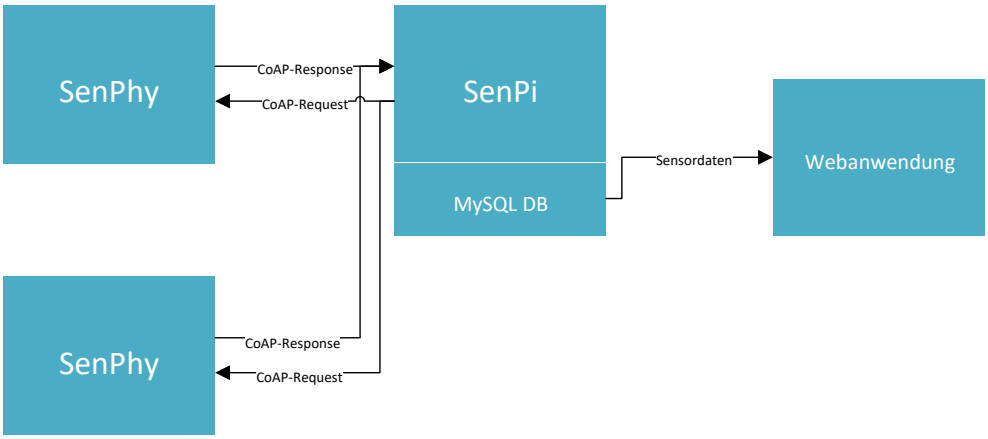

Abbildung 1: Interaktionen der Anwendungskomponenten

Der genaue Ablauf der Anfragen an die Boards, ist in Abbildung 2 dargestellt.

6 Weiterführende Arbeit

Sobald der in Abschnitt 2 erwähnte Bug im Funktreiber der phytec-Boards behoben ist, sollten die Nachrichten ausgebaut werden. Zunächst wäre es dann möglich, wirklich alle Sensorwerte zur Verfügung zu stellen, was bisher nur eingeschränkt möglich ist, weil die Nachrichten zur Ressourcenfindung nicht zu groß werden dürfen. Weiterhin sollte ein Basename in jede Nachricht eingefügt werden, damit Nachrichten von verschiedenen Boards unterschieden werden können. So wäre es möglich, Werte von verschiedenen Sensoren abhängig von ihrer Position unterschiedlich zu gewichten.

Eine solche Gewichtung ist sinnvoll für den eigentlichen nächsten Schritt im Rahmen von Smart-Environment: das Reagieren von Aktuatoren auf die Sensorwerte. Solche Aktuatoren könnten z.B. Motoren für das Öffnen und Schließen der Fenster in Abhängigkeit von Temperatur und Luftfeuchtigkeit sein oder auch das Steuern der Rollos in Abhängigkeit von Lichteinfall.

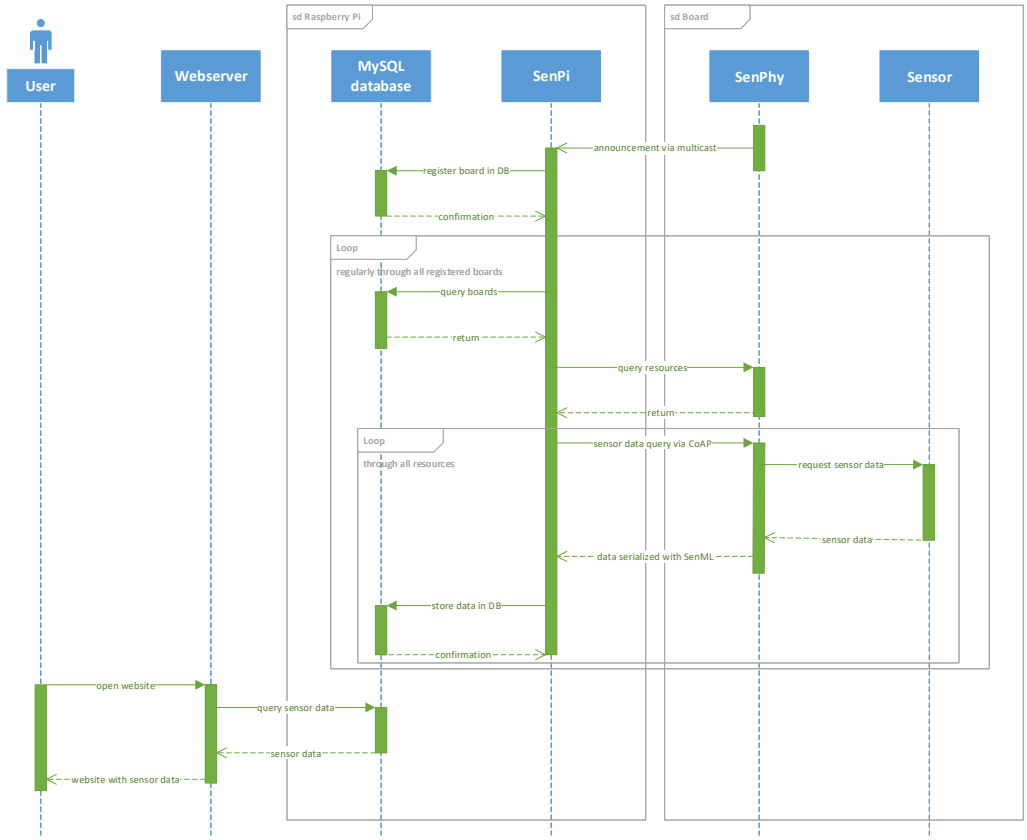

Abbildung 2: Sequenzdiagramm der Anwendung

Es könnte auch sinnvoll sein, das Auffinden der Ressourcen zu machen ohne, dass die Boards dafür aktiv werden müssen. Derzeit schicken die Boards regelmäßig Benachrichtigungen an die Link-Local-Multicast-Gruppe, um allen Servern in Reichweite zu signalisieren, dass sie Daten bereitstellen, siehe 2. Die Server könnten aber auch einfach über die Multicastgruppe */.well-known/core* anfragen und so die Bezeichnungen für alle bereitgestellten Ressourcen erhalten. Der Vorteil wäre, dass die Boards so Energie sparen, da sie nicht regelmäßig eine Nachricht verschicken müssen, sondern nur noch auf konkrete Anfragen antworten. Der Nachteil ist allerdings, dass die SenPi-Anwendung dann eine deutlich komplexere Logik benötigt, um unerwünschte Ressourcen (die z.B. von Anwendungen, die nicht zu diesem Projekt gehören bereitgestellt werden) auszusortieren. Da die Boards im aktuellen Aufbau an eine feste Energiequelle angeschlossen sind, wurde der Weg mit dem regelmäßigen Meldungen der Boards gewählt.

Ebenfalls sollte noch ein regelmäßiges Bereinigen der Datenbank des Servers implementiert werden. Derzeit werden alle jemals angefragten Daten dauerhaft gespeichert, was zwangsläufig (früher oder später) zum Überlaufen der Datenbank führen würde. Möglich wäre es z.B. alle Datenpunkte, die ein bestimmtes Alter erreicht haben, tageweise zusammenzufassen, sodass für jeden Tag nur noch ein Datenpunkt in der Datenbank liegt.