Gesamtdokumentation des FDBS im Projekt SensorCloud

[DBAP14]

Version 1.0 , 20.01.2015

Prof. Dr. Gregor Büchel
Henning Budde, M. Sc.
Thomas Partsch, M. Sc.

Gefördert durch:
Bundesministerium für Wirtschaft und Technologie
aufgrund eines Beschlusses des Deutschen Bundestages

Fachhochschule Köln
Cologne University of Applied Sciences

1. Vorhaben

Das System der SensorCloud hat einen komplexen Aufbau zur Erhebung, Verdichtung, Übermittlung und Steuerung von Sensordaten. Hardwareseitig besteht die Cloud aus einem System hochleistungsfähiger vernetzter Rechner, die via Internet mit Gebäuden, in denen Sensordaten erhoben werden, verbunden sind. In diesem Verbund von Rechnern werden mannigfache Arten von Sensordaten im Interesse der Kunden und des Betreibers der SensorCloud persistent verwaltet.

Sensor-/Aktornetzwerke besitzen zumeist proprietäre Zugangspunkte, über die eine Internet-Steuerung zwar möglich ist, aber die für die neuartigen Cloud-Anwendungen und die damit verbundenen Sicherheitsanforderungen nicht konstruiert wurden. Zudem sind diese Ansätze zumeist nicht service-basiert, sondern auf ein einfaches Protokoll beschränkt, was ihre Erweiterbarkeit entscheidend einschränkt. Diese Einschränkungen können mit einem wissensbasierten Ansatz zur Repräsentation von Sensoren/Aktoren überwunden werden, der gleichzeitig die Definition allgemeiner SensorCloud-Dienste ermöglicht.

Traditionell wird im industriellen Maßstab das Problem der Integration lokaler Datenbanken in ein globales Datenbanksystem durch den Einsatz eines verteilten Systems relationaler Datenbanken (RDB) gelöst. Erfahrungen der Betreiber großer Cloud Systeme (Amazon, Facebook, Google u.a.) zeigen aber, dass RDB Systeme in Hinsicht des Problems der horizontalen Skalierbarkeit großer und heterogen strukturierter Datenmengen nicht das Mittel der Wahl sind und betreiben hierfür die Entwicklung von NoSQL Systemen.

Die Kombination obiger Betrachtungen zum Aufbau eines geeigneten Persistenzmechanismus für die SensorCloud führt zum Problem der Integration heterogener Datenbankstrukturen in einem verteilten Datenbanksystem. Zur Lösung des Problems wurde das Konzept föderierter Datenbanksysteme (FDBS) betrachtet. Ein föderiertes Datenbanksystem ist ein schwach gekoppeltes verteiltes Datenbanksystem. Das Konzept föderierter Datenbanksysteme wird bei der Konstruktion von Data Warehouse Systemen angewandt. Ein FDBS kann als eine Vereinigung verschiedener Datenbanken, die jeweils ein eigenes lokales Schema haben, mit einem gemeinsamen globalen Schema angesehen werden. Der Ansatz zum Aufbau eines geeigneten FDBS für die SensorCloud lässt sich in drei Zielen zusammenfassen:

- Die Analyse, das Design und die Implementierung des konzeptionellen Schemas des FDBS als Komponente der Cloud. Dienste zur Evolution der Schemata der Cloud DB und der DB Systeme der Location Master, wie z. B. ein Schema Monitor, waren als Bestandteile dieser Komponente zu entwickeln und wurden in einer serviceorientierten Architektur, in Form von Web Services bereit gestellt.

- Die Analyse, das Design und die Implementierung von Segmenten der Cloud DB zur Verwaltung von Sensordaten in der Cloud, die von oder an Rechnern vom Typ des Location Masters übertragen werden. Hinsichtlich der Analyse der persistent zu verwaltenden Sensordaten wurden zur Entwicklung eines kontrollierten Vokabulars und eines Kategoriensystems, das zur Komplexitätsreduktion der Sensordaten eingesetzt wird, geeignete Ontologiekonzepte für Sensornetzwerke entwickelt. Für das Design der Cloud DB wurden Erfahrungen der Betreiber großer Cloud Systeme in Form von „structured storage“ Konzepten in Hinsicht auf Skalierbarkeit und Erweiterbarkeit untersucht, wie sie z. B. bei NoSQL Datenbanken benutzt werden. Da diese Konzepte jedoch im Unterschied zu RDB Konzepten keine vollständige Sicherheit von Transaktionen im Sinne der ACID Kriterien garantierten, wurden parallel Untersuchungen von Transaktionssicherheit, Gruppen- und Nutzerrechteverwaltung ausgeführt.

- Die Analyse, das Design und die Implementierung der lokalen DB des Location Master. Für die Analyse und das Design ist eine Zwischenspeicherungsstrategie bei Unterbrechung der Verbindung zur SensorCloud entwickelt worden. Weiterhin unterstützt die lokale DB die Transaktionskontrolle und die Überwachung der Gruppen- bzw. Nutzerzugriffsrechte auf Sensordaten, damit der Location Masters seine Rolle als Trustpoint ausführen kann. Bei der Analyse ist zu spezifizieren, welche Methoden der Vorverarbeitung von Sensordaten durch das DB System bereitgestellt werden sollen, um die Übertragungslast von Sensordaten in die Cloud zu minimieren.

1.1 Arbeitspakete des FDBS in der SensorCloud

Zum Erreichen der drei genannten Hauptziele wurden vor Projektstart die folgenden Arbeitspakete definiert:

- DBAP1:
  Anforderungsanalyse zur Verwaltung von Sensordaten mittels eines FDBS in der Cloud. Hierbei sind sowohl die Anforderungen der Installation unterschiedlicher Sensortypen in den durch die Cloud verwalteten Lokationen als auch die an die Messlinien gestellten Bedingungen, die aus den als Produktlinien angebotenen Services der Cloud entspringen, zu berücksichtigen.

- DBAP2:
  Deploymentanalyse: Bewertung vorliegender DBMS in Hinblick auf ihre Tauglichkeit für die Verwaltung von Sensordaten in Segmenten der Cloud DB als auch in Form leichtgewichtiger Datenbanken auf Rechnern vom Typ des Location Masters. Eine Festlegung auf einen DB Modelltyp (relational, NoSQL oder objektorientiert) soll Ergebnis der Bewertung sein. In Hinblick auf das Deployment bezüglich der Rechnertypen (leistungsstarke Cloud Hostsysteme vs. Rechner vom Typ des Location Masters mit stark eingeschränkten Betriebsmitteln) soll explizit keine Homogenitätsanforderung erhoben werden, d.h. ein FDBS bestehend aus DBMS unterschiedlicher Modelltypen wird als Implementierungsvorschlag a priori nicht ausgeschlossen.

- DBAP3:
  Mitwirkung an der semantische Analyse der persistent zu verwaltenden Sensordaten: Hierbei soll ein logisches Datenmodell mittels Ontologien für Sensoren auf der Grundlage bestehender Standards von Modellierungssprachen für Sensoren entwickelt werden.

- DBAP4:
  Auf Grundlage der Anforderungs-, Deployment- und semantischen Analyse soll das konzeptionelle Schema des FDBS der SensorCloud spezifiziert werden. Nach Abschluss der Analyse soll das konzeptionelle Schema als globales Schema des FDBS als maschinenlesbares Data Dictionary in einem Segment der Cloud DB implementiert werden.

- DBAP5:
  Entwicklung einer Familie von Schema Monitoring Diensten, die die wechselseitige Konsistenz der lokalen Schemata der Datenbanken der Location Master und des globalen Schemas des FDBS für die Sensordaten in der Cloud sicherstellen helfen sollen.

- DBAP6:
  Entwicklung einer Familie von Diensten, die der Übertragung von Sensordaten von Location Master in die Cloud DB als auch umgekehrt ausführen. Diese Dienste sind transaktionssicher und die Integrität der Sensordaten prüfend zu implementieren.

- DBAP7:
  Spezifikation und Implementierung eines Konzepts, das die Zwischenspeicherung von Sensordaten, die in die Cloud übertragen werden sollen, auf dem Location Master sicherstellen soll, wenn dessen

Internetverbindung zur Cloud DB unterbrochen ist.

- DBAP8:
  Entwicklung eines datenbanktechnischen Konzepts zur Unterstützung von Sensordiensten auf dem Location Master. In der Implementierung des Konzepts sollen mindestens Sensoren der Typen Raumsensor, Energieerfassungssensor, Sensornetzwerk Homematic und Vision Sensor für SensorCloud unterstützt werden.

- DBAP9:
  Um die Sicherheit und das Vertrauen der Übertragung von Sensordaten in die Cloud bzw. von der Cloud in die Lokation zu unterstützen, sollen seitens der Datenbanktechnik Methoden und Dienste bereitgestellt werden, um die Einrichtung des Location Master als Trustpoint zu unterstützen. Hierbei sind datenbankgestützt geeignete Strukturen für die gesicherte Übertragung und Speicherung von Sensordaten mit dem Bezug zum Location Master zu implementieren, die von den Konsortionalpartnern entwickelt werden.

- DBAP10:
  Koordinierung der Arbeiten der Datenbankarbeitsgruppe FH mit den Arbeiten der anderen Konsortionalpartnern und mit den FH Arbeitsgruppen Eingebettete Systeme und Vision Sensor. Aufgrund der zentralen Rolle der Speicherung der Sensordaten in der Cloud DB gibt es erhöhten Koordinationsbedarf mit den anderen Konsortionalpartnern, die benachbarte Segmente in der Cloud DB aufbauen bzw. die die Sensordaten der Cloud DB nutzen.

- DBAP11:
  DB Lasttest auf dem Location Master, Zwischenpufferung von Bildern des Vision Sensors auf der lokalen DB und ihre Übertragung in die Cloud DB.

- DBAP12:
  DB Systemintegration auf dem Location Master und in der Cloud, DB gestützter Lasttest des Location Master als Trustpoint.

- DBAP13:
  Gesamttest des FDBS für die Sensordaten in Verbindung mit dem vollständigen Test der SensorCloud. Fehlerbehebung. Durchführung des Gesamttests mit einem Massenaufkommen von Sensordaten. Eventuell Redesign und Neuprogrammierung einzelner Komponenten.

- DBAP14:
  Vollständige Dokumentation des FDBS für die Sensordaten.

2. Entwickelte Module

2.1 Konzeptionelles Schema und Ontologie

2.1.1 Konzeptionelles Schema

Das föderierte Datenbanksystem der SensorCloud besteht aus einem Zusammenschluss verschiedener Datenbankmanagementsysteme, die über unterschiedliche Datenmodelle (relationale, objektorientierte, NoSQL Datenmodelle) verfügen. Dieser Zusammenschluss wird über ein konzeptionelles globales Datenbank Schema gesteuert und beinhaltet jeweils für Cloud und Gateways verschiedene Fragmentierungsschemata.

Die Entitäten des globalen Schemas werden in sechs verschiedene Gruppen gegliedert umso semantisch zusammengehörige Entitäten enger aneinander zu binden. [DBAP4]

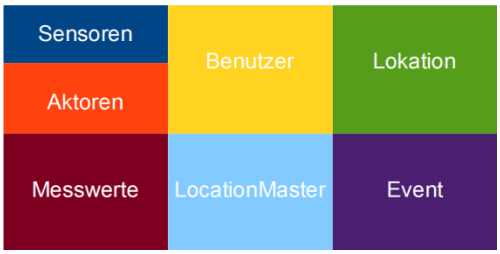

Abbildung 1: Entitätsgruppen des FDBS der SensorCloud

Derzeit beinhaltet das konzeptionelle Schema der SensorCloud 69 Entitäten und ist als generisches Schema so konzipiert, dass durch neue Anwendungsfälle und zukünftige Erweiterungen weitere Entitäten entstehen können und dem konzeptionellen globalen Schema einfach hinzugefügt werden können.

Entitätsname	Entitätsname	Entitätsname
Admin	EventMitglieder	NavigationsKoordinaten
AdminMitglieder	Flur	NetzwerkManager
AdminRechte	Gebaeude	NutzerEmail
Adresse	Geraet	NutzerSicherheit
Aktor	GeraetTyp	NutzerStammdaten
AktorAnforderung	Gruppen	NutzerTelefon
AktorKommunikation	GruppenMitglieder	Raum
AktorKomponente	GruppenRechte	Sensor
AktorKonfiguration	LocationMaster	SensorEvent
AktorProdukt	LocationMasterKomponente	SensorKommunikation
AktorProduktKonfiguration	LocationMasterKomponentenKonfiguration	SensorKomponente
AktorService	LocationMasterKonfiguration	SensorKonfiguration
AktorServiceFunktion	LocationMasterTopologie	SensorProdukt
AktorServiceFunktionMitglied	LocationMasterTopologieKomponente	SensorProduktKonfiguration
AktorServiceMitglied	LocationMasterTreiber	SensorService
AktorTyp	Lokation	SensorServiceFunktion
AktorVerbund	Mandanten	SensorServiceFunktionMitglied
AktorVerbundMitglieder	MandantenMitglieder	SensorServiceMitglied
Anlage	MessLinie	SensorTyp

Aufzug	MessLinieTyp	SensorVerbund
Etage	MessTyp	SensorVerbundMitglieder
Event	MessTypMitglieder	ServiceLinien
EventAktion	Messwert	ServiceLinienMitglieder

Abbildung 2: Entitäten der SensorCloud

Trotz der Konstruktion der SensorCloud als föderiertes Datenbanksystem mit relationalen und NoSQL DBMS wird das konzeptionelle globale Schema als relationales Entitätenmodel beschrieben, dies bedeutet, dass die Entitäten untereinander mathematisch beschreibare Relationen bilden können und voneinander Abhängigkeiten besitzen. Diese Abhängigkeiten können je nach DBMS physikalisch (relat. DBMS) implementiert sein oder auch nur als rein virtuelle (NoSQL DBMS) Abhängigkeit im konzeptionellen Schema vorhanden sein. Eine Übersicht der Entitäten der gruppenübergreifenden Abhängigkeiten ist in Abbildung 3 zu sehen.

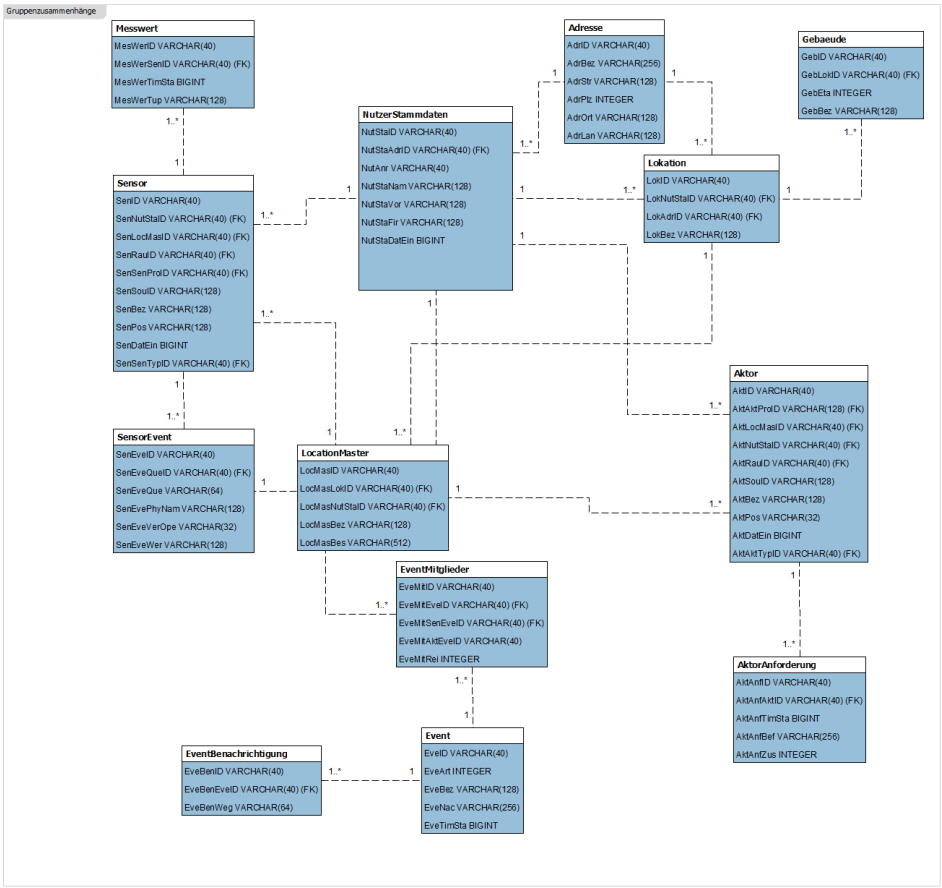

Abbildung 3: Ausschnitt aus dem ERD des konzept. globalen Schemas der SensorCloud

Einen Spezialfall im Konzeptionellen Schema der SensorCloud stellt die Entität Messwert dar. Da diese als Senke für Massendaten gedacht ist, muss hier speziell für das jeweils verwendete DBMS Anpassungen gemacht werden. Als Referenzimplementierung wurde Apache Cassandra als DBMS gewählt. Die Anpassungen der Messwert Entität sind in 2.3.1 nachzulesen.

MySQL Workbench

Das Konzeptionelle Schema der SensorCloud wird als Basis in der MySQL Workbench[1] abgebildet. In diesem Tool lassen sich, wie vom Konzeptionellen Schema vorgesehen, einzelne Entitäten zu Gruppen zusammenfassen und deren Abhängigkeiten (sowohl innerhalb als auch außerhalb einer Gruppe) definieren. Die MySQL Workbench bringt zwei essentielle Funktionen mit, die für einen zeitgemäßen evolutionären Workflow entscheidend sind.

Eine Funktion ist die grafische Dokumentation, d.h. die Erstellung eines Entity-Relationship-Diagrams (ERD). Automatisch können hier komplett ERDs, Gruppen-ERDs oder auch individuelle Teil-ERDs erstellt werden. Für eine schnelle Dokumentation von Änderungen, die durch die beständige Weiterentwicklung der SensorCloud nötig ist, lassen sich diese Diagramme erzeugen und entsprechend in die Entwicklerdokumentation einpflegen.

Die zweite wichtige Funktion ist die Generierung von SQL Scripten, auf denen in weiteren Arbeitsschritten das JSON Globale Schema (vgl. 2.1.2) basiert. Die SQL Scripte sind maschinell verständlich und beinhalten Entitätsnamen, Entitätsgruppen, Entitätsattribute mit Datentypen und deren Integritätsbedingungen. Auch hier gilt wieder, dass diese schnell bei Änderungen am konzeptionellen Schema neu generiert werden können und so ein geregelter Arbeitsablauf für Schemaänderungen aufgestellt werden kann.

JSON Schema Creater

Der JSON Schema Creater [JSCHECRE] ist ein eigen in Java entwickeltes Tool, welches den Übergang von den MySQL Workbench Skripten hinzu zu dem JSON Globalen Schema vollzieht. Es durchläuft die aus der MySQL Workbench erstellten SQL Skript Dateien und extrahiert dort sämtliche für das JSON Globale Schema nötigen Informationen. Der Schema Creater kennt dabei die Struktur des JSON Globalen Schemas und füllt mit den extrahierten Werten die relevanten Felder des JSON Globalen Schemas.

2.1.2 Ontologie und Regelwerk

JSON Globales Schema

Innerhalb der SensorCloud greifen viele verschiedene Dienste auf Daten der Datenbank zu. Da im Laufe der Zeit sich Änderungen an den Entitäten ergeben und auch neue Entitäten entstehen, muss eine maschinell lesbare Form des konzeptionellen Schemas der SensorCloud existieren. Diese ist das JSON Globale Schema. Somit können alle Anwendungen gegen die maschinell lesbare Schnittstelle programmiert werden (vgl. 2.2.1).

Das Schema der SensorCloud wird als JSON globales Schema erstellt und enthält sämtliche vorhandene Entitäten mit Namen, Datentypen, Abhängigkeiten (Constraints) und weiteren Beschreibungen (Entitätengruppe, zusätz. Beschreibungen, …) im JSON Format. Als Beispiel ist ein Ausschnitt aus dem JSON globalen Schema Eintrag für die Entität Sensor angegeben.

[1] http://www.mysql.de/products/workbench/

```
{
"global": {
        "entityGroup":[
        {
                "nameOfGroup":"Sensor/Aktor",
                "entity":[
                {
                        "nameOfEntity":"Sensor",
                        "attributes": [
                        {
                                "name":"SenID",
                                "friendlyName":"Sensor ID",
                                "datatype":"text",
                                "constraints":["PRIK","UNIQUE", "UUID","notNull"],
                                "description":"Eindeutige ID für einen Sensor"
                        } ,
                        {
                                "name":"SenBez",
                                "friendlyName":"Sensor Bezeichnung",
                                "datatype":"text",
                                "constraints":[],
                                "description":"Bezeichnung für einen Sensor"
                        } ,…

                        ]
                }]
        }]
}}
```

Abbildung 4: Ausschnitt aus JSON Globalen Schema für die Entität Sensor

Das JSON Globale Schema folgt der Idee einer Schema-Management-Komponente aus [KLE2014] und wird als zusätzliche Schicht außerhalb des Datenbanksystems genutzt. Nicht jede Applikation ist gezwungen diese Schema-Management-Komponente zu nutzen, für eine Applikation ist dies aber empfehlenswert um jede Evolutionsstufe der SensorCloud mitgehen zu können.

Regeln

Zur Steuerung von Aktoren wird innerhalb der SensorCloud auf ein Regelwerk gesetzt, welches aus der Datenbank generiert wird. Eingabewerte für eine Regel können eine oder auch mehrere sensorische Eigenschaften (Messwerte) sein, die zu einer oder auch mehreren Aktionen führen können.

Das Eventverhalten kann anstelle von einzelnen Messwerten als Voraussetzung für ein Event (if <Voraussetzung>) auch als Verbund definiert werden: Ein Event kann dann aufgrund eines Sensorverbund-„Messwerts“ ausgelöst werden. Dieser Sensorverbund-„Messwert“ kann als Aggregat (Summe, Durchschnitt, etc.) der Messwerte der am Verbund beteiligten Sensoren ermittelt werden. Ein Aktorverbund kann durch einen gemeinsamen Event gesteuert werden (z.B. alle Lampen in Raum R1 einschalten, wenn der zugehörige Verbund von Bewegungsmeldern das Ereignis „Bewegung“ meldet).

Durch die Zusammenfassung von Sensoren und Aktoren zu Verbünden ist es somit möglich, mehrere Aktoren zusammengefasst zu einem Verbund durch ein Ereignis eventbasiert zu steuern. Genauso können mehrere Messwerte von verschiedenen Sensoren zusammengefasst zu einem Event als eine Voraussetzung (z.B. als Summe) für eine Steuerung eines Aktorverbunds genutzt werden.

Ein weiterer Vorteil durch den Einsatz von Verbünden ist, wie folgt, zu sehen: Wird ein neuer Sensor/Aktor zu einem Gateway hinzugefügt, kann dieser sich automatisch (durch zusätzliche Software) zu einem Verbund hinzufügen. Durch das Hinzufügen zu diesem Verbund ist dieser neue Sensor/Aktor sofort Teil einer Regel bzw. eines Events, ohne dass hierfür eine Regel verändert oder hinzugefügt werden muss.

Regeln und Events können von der Cloud heraus gesteuert werden, ohne dass die Cloud dabei jeden

Sensor/Aktor, die mit den Regeln/Events in Verbindung stehen, „kennen“ muss. Hierbei ist es ausreichend, wenn die Cloud den Verbund (ID) „kennt“. Dadurch wird eine gewisse Kapselung erreicht. Ein Sensor muss so nicht der Cloud direkt bekannt sein, sondern kann durch einen Verbund verschleiert werden.

Regelaufbau

Eine Regel R für ein Event hat die Form: $V_1 \wedge V_2 \wedge V_3 \ldots \wedge V_m \rightarrow A_1 \wedge A_2 \wedge \ldots \wedge A_n$. Hierbei sind V_i ($1 <= i <= m$) sensoriell ermittelte Voraussetzungen und A_j ($1 <= j <= n$) sind Aktivitäten (aktorische Aktivitäten oder Benachrichtigungen des Systems an Nutzer oder andere Beteiligte (z.B. Feuerwehr)), die vom System SensorCloud ausgefüllt werden, wenn $V_1 \wedge V_2 \wedge \ldots \wedge V_m$ wahr ist. Da es sich bei den V_i um Ereignisse handelt, nenne ich R: $V_1 \wedge V_2 \wedge V \ldots \wedge V_m \rightarrow A_1 \wedge A_2 \wedge \ldots \wedge A_n$ eine EVENTART. Diese Bezeichnung dient auch der Einordnung in das Schema des FDBS. Der Regelaufbau sieht demnach wie folgt aus:

IF

{ V1 AND V2 AND … AND Vm }

THEN

{ A1 AND A2 AND … AND An }

Für V „Element aus“ { V_1 , V_2 , V … , V_m } und A „Element aus“ { A_1 , A_2 , … , A_n } werden nachfolgend Beschreibungen in einem Pseudocode angegeben, um ihre Abbildbarkeit in ihren Stammdaten und ihren Parametern, die durch den Benutzer eingegeben werden, darzustellen:

V := iTypV(S, SP) ∧ hat(S, mw) ∧ vglA(mw, g1)

Hierbei ist:

b1) iTypV(S, SP) := „Sensor S ist vom Typ des Sensorprodukts SP“ (Ein Sensorprodukt ist eine konkrete Realisierung eines Sensortyps).

b2) hat (S, mw) := „Sensor S hat Einzelwert mw aktuell gemessen bzw. zeigt Zustand mw an.“

BSP: SP = Thermometer → mw = 29° C

SP = Tür/Auf/Zu-Sensor → mw = ZU

SP = Brandmelder → mw = ALARM_AN

b3) vglA(mw, g1): Vergleichsausdruck (zulässig sind die üblichen Vergleichsoperatoren: <, >, =, >=, <=) zwischen mw und einem vom Benutzer einzustellenden Grenzwert g1 oder Zustand g1, ab dem eine Aktion auszulösen ist. Bei boolschen Zuständen können Defaults eingestellt sein.

BSP für Voraussetzungen V:

V = iTypV(4711, Thermometer) ∧ hat(4711, 29) ∧ mw < 24

V = iTypV(4812, Tür/Auf/Zu) ∧ hat(4812, ALARM_AN) ∧ mw = ALARM_AN

V = iTypV(4910, Tür/Auf/Zu) ∧ hat(4910, Zu) ∧ mw = ZU

Definition von Regeln/Events:

Aktoren:

- A1: FensterAufZu: FAZ oeffnen(), schliessen()
- A2: Jalousie: JAL hoch(), runter()

- N: Nachricht

Sensoren:

- S1
- S2
- S3

Raum R:

- Sensoren: TS → t
- HelligkeitsSensor: HS → h
- VisionSensor: VS → Pers_da (boolean = 1)

Nutzer (Eig.):

- E

IF

((iTypV(S1, TS) Λ hat(S1, t) Λ gT(t, 25°))

Λ (iTypV(S2, HS Λ hat(S2, h) Λ gT(h, 10 Lux))

Λ (iTypV(S3, VS Λ hat(S3, Per_da) Λ gL(Pers_da, 0))

)

THEN

((iTypV(A1, FAZ) Λ A1.oeffnen())

Λ (iTyp(A2, JAL) Λ A2.runter()

Λ N(E, „Fenster auf in R“))

)

Regelerstellung

Die Regeln der SensorCloud werden zunächst im föderiertem Datenbanksystem der SensorCloud abgelegt.

Für die Abbildung der Regeln sind im konzeptionellen Schema mehrere Entitäten vorgesehen, die im folgenden Abschnitt erläutert werden.

Entität SensorEvent:

SenEveID	SenEveQueID	SenEveQue	SenEvePhyNam	SenEveVop	SenEveWer
4711	S1	Sensor	Temperatur	>	25°
4712	S2	Sensor	Helligkeit	>	10 Lux
4713	S3	Sensor	Bewegung	=	0

In der Entität SensorEvent werden Events von Sensoren ausgehend hinterlegt, z.B. ein Sensor S1 hat einen Temperaturwert > 25.

Entität EventMitglieder:

EveMitID	EveMitEveID	EveMitSenEveID	EveMitRei
103	2030	4711	1
104	2030	4712	2
105	2030	4713	3

In dieser Entität werden SensorEvents bestimmten Events zugeordnet. Zum Beispiel werden die SensorEvents 4711, 4712 und 4713 dem Event 2030 zugeordnet.

Entität Event:

EveID	EveArt	EveBez	EveNac	EveTimSta
2030	400	Temp.- u. Helligkeitssteuerung	„Fenster auf in Raum R“	1234567891234

In der Entität Event wird das Event mit einer Nachricht, einer Bezeichnung, einem Zeitstempel und einer Eventart hinterlegt. Die EventArt kann wie eine Priorität interpretiert werden: EveArt > 1000: Alarm (zeitkritisch); EveArt < 500: Regeln (nicht zeitkritisch).

Entität EventBenachrichtigung:

EveBenID	EveBenEveID	EveBenWeg
7001	2030	EMail: E@gmx.de

Für Events wird in EventBenachrichtigung der Benachrichtigungsweg hinterlegt, z.B. Benachrichtigung an eine bestimmte E-Mail-Adresse.

Entität EventAktion:

EveAktiID	EveAktiEveID	EveAktiBez	EveAktiZieID	EveAktZie	EveAktiZiePar	EveAktiZieWer	EveAktiPrio
8002	2030	FensterAufZu	502	Aktor	AufFunktion	100	1
8003	2030	Jalousie	503	Aktor	HochFunktion	50	2

In EventAktion ist hinterlegt, welche Aktion(en) durch ein Event gestartet werden. Dazu wird das Ziel (z.B. Ziel ID 502) in der Entität Aktor abgelegt.

AktorServiceFunktion:

AktSerFunID	AktSerFunNam

9001	AufFunktion
9002	HochFunktion

In der Entität AktorServiceFunktion ist hinterlegt, welche Funktionen existieren. Über die Entitäten AktorServiceFunktionMitglied, AktorService und AktorServiceMitglied kann erfragt werden, welche Services und damit auch welche ServiceFunktionen ein AktorProdukt anbietet.

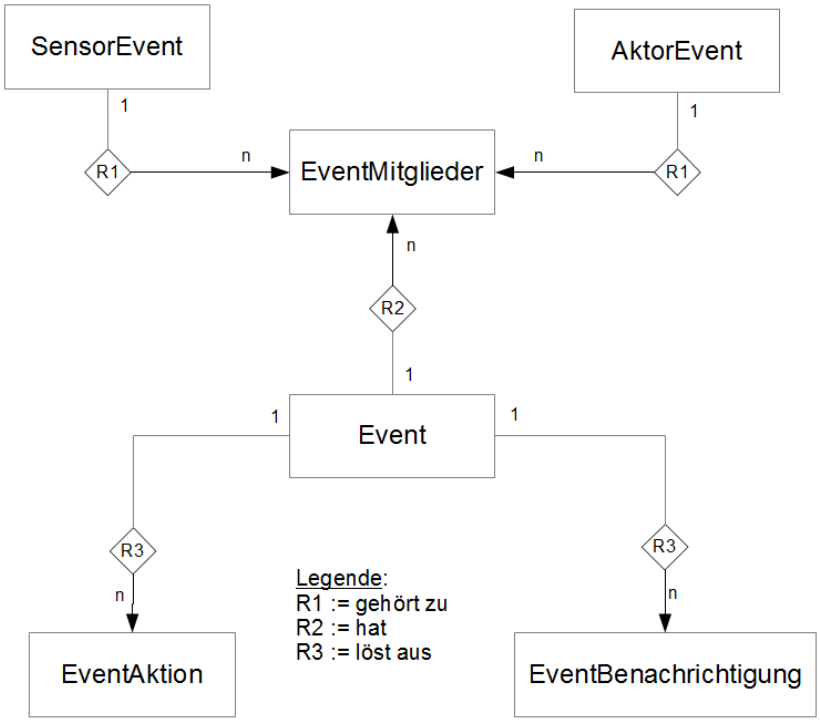

Abbildung 5: Darstellung der Entitäten im Zusammenhang mit Events als ERD

Zur Erstellung von Regeln wurden verschiedene prototypische Applikationen entwickelt, die es einem Nutzer/Entwickler erlauben neue Regeln in die SensorCloud einzupflegen. Besonders zu erwähnen sind dabei eine Android App [BA-AS], die neben der Darstellung von Messwerten eben auch neue Regeln erstellen lässt. Neben der Android App wurde eine Java Applikation geschrieben [REGU], die den Benutzer GUI-geführt Regeln erstellen lässt, d.h. der Benutzer kann nur seine eigenen Sensoren und Aktoren mittels Regeln verknüpfen und nur Funktionen, die ein Sensor oder Aktor anbietet (gestützt durch die Produkt Semantik für Sensoren und Aktoren) können für eine Regel auch genutzt werden. Eine weitere Option zur Regelerstellung bietet der SensorCloud DAL, der ein Web Interface zur Verfügung stellt [SCEDITEV]. Diese Variante ist aber derzeit nur für Entwickler gedacht.

Regel- und Konfig-Extraktor

Der Regel- und Konfig-Extraktor ist das Zwischenstück zwischen den in der Datenbank gespeicherten Regeln und dem Regelwerk der SensorCloud. Zusätzlich zu den Regeln werden auch Konfigurationen für Sensoren und Aktoren mit Hilfe diese Tools extrahiert. Hauptzweck des Tools ist die Generierung eines Regelwerks für einen LocationMaster, das in Form einer JSON Regeldatei zur Verfügung gestellt wird. Da für bestimmte Regeln auch die Lokationen, an denen sich Sensoren oder Aktoren befinden relevant sind, müssen auch diese verfügbar gemacht werden.

Der Normalfall sieht vor, dass Regeln, Konfigurationen und Lokationen aus der Cloud Datenbank (Cassandra) extrahiert werden und als JSON Regel-, Konfigurations- und Lokations-Datei abgelegt werden.

```
{
  "rules": [
    {
      "description": {
        "eventId": "<value>",
        "eventType": "<value>",
        "eventName": "<value>",
        "eventMessage": "<value>",
        "eventTimestamp": "<value>"
      },
      "conditions": [
        {
          "condSensorGroup": "<value>",
          "condSensorId": "<value>",
          "condPhyNam": "<value>",
          "condCompOperator": "<value>",
          "condValue": "<value>"
        },
        ...
      ],
      "actions": [
        {
          "actionActorGroup": "<value>",
          "actionActorId": "<value>",
          "actionFunction": "<value>",
          "actionParameter": [
            {
              "n": "<value>",
              "sv": "<value>"
            },
```

Abbildung 6: Beispiel für eine JSON Regel-Datei

Der Regelinterpreter kann die JSON Regel-Datei importieren und gültige Regeln überwachen. In zyklischen Abständen werden die Regeln mit Hilfe des Extraktors aus der Cloud Datenbank extrahiert und zum LocationMaster transferiert. Regeländerungen werden derzeit nicht sofort sondern erst nach periodischem Abfragen der Cloud synchronisiert. [PRAX-OE]

SensorProduktSemantik und AktorProduktSemantik

Das Projekt SensorCloud lebt von der Heterogenität der Sensoren und Aktoren. Dies führt dazu, dass jeder Sensor eine unterschiedliche Art der Darstellung seines gemessenen Wertes verwendet. So kann es beispielsweise sein, dass eine Temperatur von einem Sensor mal in Grad Celsius und von einem anderen

Sensor mal in Fahrenheit angegeben wird. Auch die Darstellung der Werte in unterschiedlichen Zahlensystemen (Dezimalsystem, Binärsystem, Hexadezimalsystem) ist dabei durchaus geläufig. Das Beispiel des Multi-Raum-Sensors zeigt, dass auch ein Sensor gleichzeitig mehrere Messwerte in einem Messwert (ein Messwert Tupel) erzeugen kann. Diese Verschiedenartigkeit der Messwerte führt dazu, dass eine maschinelle Lesbarkeit eines Messwerts nicht direkt gegeben ist. Die Heterogenität der Aktoren ist durch die unterschiedlichen Inventarien an Funktionen, die Aktoren eines Aktorprodukts ausführen können, gegeben.

Die Analyse einiger Sensorprodukte führte zu einer Folge von Deskriptoren, die für eine SensorProduktSemantik von Messwerten erforderlich sind. Ein Messwert kann dabei aus mehreren Komponenten bestehen [DBAP3]. Dabei ist es wichtig zu wissen, wie viele Komponenten ein Messwerttupel (NR: Anzahl der Messwerte eines Tupels) besitzt.

Jede Komponente eines Messwerttupels muss einzeln und explizit beschrieben werden. Dabei sind in der Entität SensorProdukt zu folgenden Deskriptoren Einträge zu hinterlegen:

- nr: Nummer/Stelle des Messwertes; dient für einen direkten Zugriff auf Deskriptoren des n. Messwerts in einer Produkt Semantik.
- pn: physikalischer Name des gemessenen Phänomens (:= ein Wort eines kontrollierten Vokabulars (z.B. Temperatur, Geschwindigkeit, Luftfeuchtigkeit,...)).
- pns: Physikalischer Name des Sensors; die Bezeichnung kann bei Sensoren verschiedener Hersteller/Produkte unterschiedlich sein (z.B. wird die Luftfeuchtigkeit mal als „Luftfeuchte“, mal als „Luftfeuchtigkeit“ und mal als „Humidtiy“ bezeichnet).
- w: Wertebereich des Messwerts (Intervall, Aufzählung, Wahrheitswert(An/Aus)).
- eh: physikalische Einheit des Messwerts. Im Moment werden drei Arten von Einheiten unterschieden: Physikalische Einheiten[2] (z.B. Temperatur in °C), Messwerte ohne Einheiten (<ohne Einheit>) und Wahrheitswerte.
- dt(w): Datentyp von w (z.B. int, float, …)[3].
- erlz (w): Erläuterung der Zustände, die durch einen Aufzählungswert w repräsentiert .werden
- Erläuterungen zu phynam (optional): Beschreibender Text, der die physikalische Größe genauer beschreibt.
- aggfkt: Gibt an, welche Aggregationsfunktionen verwendet werden kann (z.B. *avg* oder *sum*).
- ufkt: Beschreibt die Umrechnung eines Messwerts in ein anderes, leicht für den Nutzer lesbares Format.
- erl für weitere Erläuterungen.

Die definierte SensorProduktSemantik wird im Attribut SenProSem der Entität SensorProdukt als JSON-Format abgelegt. Durch die für (fast) alle Programmiersprachen vorhandenen Parser stellt sich diese Darstellung als leicht zu verarbeiten dar und bringt zusätzlich den Aspekt der menschlichen Lesbarkeit mit. Da die SensorProduktSemantik nicht dauerhaft ausgetauscht werden muss, ist der durch die umfangreichere Beschreibung anfallende „Overhead“ zu vernachlässigen.

Beispiel: SensorProduktSemantik in JSON

[2] SI-Einheiten
[3] Siehe Kontrolliertes Vokabular in Kapitel 1.3 in dem Bericht „DBAP3 Semantische Analyse mittels Ontologien“

```
{
        "n": <Anzahl der Werte>,
        "parvor": [
                {
                        "nr": <Nummer des Messwerts>,
                        "pn": <Physikalischer Name>,
                        "pns":<Physikalischer sensorseitiger Name>,
                        "w":{"min": <minWert> , "max": <maxWert>, "enum":[<Wertebereich>]},
                        "eh":<Einheit>,
                        "dt":<Datentyp>,
                        "erlz": {<Erläuterung Zustand>,<Erläuterung Zustand>,…},
                        "aggfkt":[<Aggregatsfunktion>,<Aggregatsfunktion>,…],
                        "ufkt": <Umrechnungsfunktion>,
                        "erl":<Erläuterung>
                },
                {
                        …
                }
        ]
}
```

Das Projekt SensorCloud lebt neben der Heterogenität der Sensoren auch von der Heterogenität der Aktoren. Dies führt dazu, dass jedes AktorProdukt eine unterschiedliche Menge an Funktionen anbieten kann, die die physikalischen Aktivitäten des Aktors steuern. Die Übergabeparameter jeder dieser Funktionen können demnach auch verschieden sein. Um wie bei den Sensoren diese Vielfalt gewährleisten zu können, wird auch hier eine geeignete Produkt Semantik für Aktoren definiert.

Aktoren sind im Datenmodell zum größten Teil ähnlich zu Sensoren abgebildet. In der folgenden Abbildung sind die Entitäten Aktor, AktorProdukt und AktorTyp abgebildet, die im gleichen Zusammenhang wie Entitäten Sensor, SensorProdukt und SensorProdukt stehen.

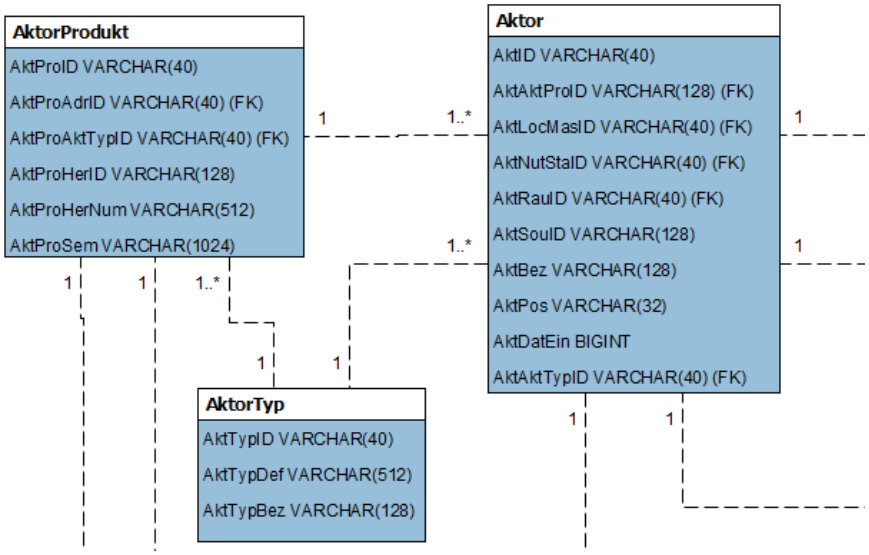

Abbildung 7: Aktor, AktorProdukt und AktorTyp im konzeptionellen Schema

In dem Attribut AktorProduktSemantik (AktProSem) wird die Produkt Semantik für die Beschreibung der Funktionsmenge eines Aktors im JSON-Format hinterlegt.

Die folgende Produkt Semantik bezieht sich auf die Funktionen, die ein Aktorprodukt anbietet. Um eine Produkt Semantik für AktorProdukte definieren zu können, sind die folgenden Eigenschaften zu beachten[4]:

- fnr: Nummer der Funktion; dient für einen direkten Zugriff auf die Semantik der n. Funktion in einer Produkt Semantik
- fnam: Name der Funktion (z.B. forward)
- fart: Funktionsart (ist im kontrollierten Vokabular des zugehörigen Aktortyps hinterlegt)
- fpanz: Anzahl der Parameter der Funktion
- pnr: Nummer des Parameters in der Folge der Übergabeparameter der Funktion
- pnam: Parameternamen in der Folge der Übergabeparameter der Funktion
- part: Parameterart gemäß kontrolliertem Vokabular des Aktortyps
- pw: Wertebereich (Intervall, Aufzählung, Wahrheitswert(An/Aus))
- peh: physikalische Einheit (z.B. Grad Celsius für Temperaturen, Wert in %, <ohne Einheit>, Wahrheitswert)
- pdt(pw): Datentyp von pw (z.B. int, float)
- perlz (pw): Erläuterungen für pw
- franz: Anzahl der Rückgabewerte der Funktion
- rnr: Nummer des Rückgabewerts
- rnam: Rückgabewertname der Funktion
- rart: Art des Rückgabewerts

[4] Die Spezifikation dieser Eigenschaften folgt der Beschreibung von Funktionen in einer RPC-ähnlichen Struktur (vgl. a. die Beschreibung von Funktionen in einer RTDL [MA-TE]

- rw: Wertebereich (Intervall, Aufzählung, Wahrheitswert(An/Aus)) des Rückgabewerts
- reh: physikalische Einheit (z.B. Wert in %, <ohne Einheit>, Wahrheitswert) des Rückgabewerts
- rdt(rw): Datentyp von rw (z.B. int, float)
- rerlz (rw): Erläuterungen für rw
- ferl: Erläuterungen der Funktion (optional): Beschreibender Text

Die definierte Produkt Semantik wird in einem einheitlichen Format im Attribut AktProSem der Entität AktorProdukt abgelegt. Das gewählte Format der Produkt Semantik ist auch hier die Java Script Object Notation (JSON).

Beispiel: Aktor Produkt Semantik in JSON

```
{
      "n": <Anzahl der Funktionen>,
      "parvor": [
            {
                  "fnr": <Nummer der Funktion>,
                  "fnam": <Name der Funktion>,
                  "fart": <Art der Funktion>,
                  "fpanz": <Anzahl der Parameter>,
                  "p": <Parameter> [
                        {
                              "pnr": <Nummer des Parameters>,
                              "pnam": <Name des Parameters>,
                              "part": <Parameterart>,
                              "pw": {
                                    "min":<minWert> , "max": <maxWert>
                                    "enum": [<Wertebereich>]
                              },
                              "peh": <Einheit des Parameters>,
                              "pdt": <Datentyp des Parameters>,
                              "perlz": {<Erläuterung Zustand>,
                                    <Erläuterung Zustand>,… }
                        },
                        {
                              …
                        }
                  ]
                  "franz": <Anzahl der Rückgabewerte>,
                  "r": <Rückgabewert> [
                        {
                              "rnr": <Nummer des Parameters>,
                              "rnam": <Name des Parameters>,
                              "rart": <Parameterart>,
                              "rw": {
                                    "min":<minWert> , "max": <maxWert>
                                    "enum": [<Wertebereich>]
                              },
                              "reh": <Einheit der Rückgabe>,
                              "rdt": <Datentyp der Rückgabe>,
                              "rerlz": {<Erläuterung Zustand>,
                                    <Erläuterung Zustand>,… }
                        },
```

```
                        {
                                ...
                        }
                ]
        }
    ],
    "fkterl": <Erläuterung>
}
```

2.1.3 SensorCloud Protokoll

Das SensorCloud Protokoll [SCPROT] wurde speziell für die Bedürfnisse der SensorCloud entwickelt. Es wird eingesetzt um zwischen Gateway und Cloud zu kommunizieren, aber auch interne Komponenten tauschen über verschiedene Typen des SensorCloud Protokolls Nachrichten aus. Das SensorCloud Protokoll, das von allen Konsortialpartnern gemeinsam entwickelt wurde, ist eine wesentliche Voraussetzung für die Durchführung der Verschlüsselungsverfahren, die vom Konsortialpartner RWTH/Comsys entwickelt wurden. Das Protokoll wird im JSON Format definiert und dient insbesondere auch dem Nachrichtenaustausch zwischen dem FDBS und benachbarten Systemen.

SensorCloud Protokoll Header

Der Kopf einer jeden SensorCloud Protokoll Nachricht sieht für alle Typen gleich aus und muss zwingend vorhanden sein.

```
{
    "ver":"number", /* Versionsnummer */
    "seq":"number", /* Sequenznummer zur eindeutigen Identifikation eines Pakets */
    "pl":"[payload]" /* eigentliche Nachricht */
}
```

Das "ver" Feld definiert die Versionsnummer des Protokoll Kopfes (Header) und der beinhalteten Nachricht (Payload). Sollten sich Änderungen an der Struktur des Headers oder des Payloads ergeben, muss die Versions Nummer erhöht werden. Der Empfänger des jeweiligen Paketes muss die aktuelle Versionsnummer unterstützen oder den Sender informieren, dass das Paket nicht akzeptiert werden konnte. Die Versionsnummer ist eine (natürliche) Zahl und ist derzeit als 1 definiert ("ver": "1").

Die Sequenznummer "seq" ermöglicht Ende-zu-Ende Bestätigungen (acknowledgments := ack) auf Applikationsschicht. Die Sequenznummer ist eine positive Integer Zahl, die bei jeder Nachricht um 1 inkrementiert werden muss.

Der Payload "pl" beinhaltet ein Feld von JSON Objekten mit Nachrichten, bspw. einen Messwert. Derzeit sind folgende Payloads definiert:

Typ	Beschreibung
1	Sensor Data
2	Configuration
4	Actuator Command
5	Actuator Response

Zusätzliche Protokoll Typen können zukünftig, wenn sie benötigt werden, erstellt werden. Payloads müssen

das "typ" Feld beinhalten um ihren Payload Typ zu spezifizieren. Zusätzlich müssen sie ein "src" (:= Source) Feld beinhalten um die Quelle der Nachricht anzuzeigen. Zusätzliche Informationen werden benötigt, wenn Sicherheitsmechanismen auf die zu übertragende Nachricht angewandt werden sollen.

Zu beachten ist, dass eine Nachricht mehrere unterschiedliche Typen von Payloads übertragen kann umso den Protokoll Header Overhead möglichst klein zu halten.

Typ 1 : Sensor Data

Um Sensor Daten (Messwerte) zu übertragen wird eine moderate Erweiterung des SENML[5] Standard eingesetzt.

```
{
        "typ":"1",              /* Protokoll Typ Nummer, hier 1 für Sensor Data */
        "src":"string",         /* Gateway ID */
        "bn":"string",          /* Sensor ID */
        "bt":"string",          /* Zeitstempel eines Sensor Data Pakets */
        "e":[{
                "n":"string",   /* physikalischer Name des Messwerts */
                "t":"number",   /* relativer Zeitstempel zu bt eines Messwerts */
                "sv":"string"   /* eigentlicher Messwert */
        }]
}
```

Jede Sensor Data Nachricht beinhaltet ein Feld "typ" mit dem Wert 1, welches den Sensor Data Typ wiederspiegelt. Das "src" und "bn" Feld beinhaltet die jeweilige einzigartige ID eines Gateways bzw. eines Sensors. Um für jeden Messwert den genauen Messzeitpunkt abzubilden existiert das Feld "t", das zu einem Wert "bt" den relativen Wert angibt.

Ein einzelner (virtueller) Sensor kann gleichzeitig mehrere Werte senden. Jeder Wert wird als eigenes Objekt im "e" Feld verschickt. Bei einem Multiraumsensor, der gleichzeitig Temperatur, Luftfeuchtigkeit, Luftqualität und Bewegung misst, entstehen vier verschiedene Messwerte zum gleichen Messzeitpunkt und somit werden auch vier Objekte im "e" Feld versendet. Innerhalb der "e" Objekte werden die "n" Felder mit den entsprechenden physikalischen Phänomenen beschrieben. Für den Multiraumsensor also z.B. "n":"Temperatur" oder "n":"Luftfeuchtigkeit". Das "e" Feld darf dabei aber keine zwei gleichen "n" und "t" Felder beinhalten.

Das "sv" Feld beinhaltet den wirklichen Messwert. Durch SENML definiert muss ein "e" Objekt nicht zwangsläufig ein "sv" Feld beinhalten. "sv" steht für „string value“. Sollte es sich beim gemessenen Wert um einen boolean Wert handeln, ist es optional möglich "bv" als Feldnamen zu verwenden.

Typ 2: Configuration

Die „Configuration“ Nachricht beschreibt den SensorCloud Protokoll Typ um Konfigurationen zu verschicken.

```
{
        "typ":"2",              /* Protokoll Typ Nummer, hier 1 für Sensor Data */
        "src":"string",         /* Gateway ID */
        "bn":"string",          /* Sensor ID */
```

[5] https://datatracker.ietf.org/doc/draft-jennings-senml/

```
        "js":"string"           /* JSON Konfigurationsbeschreibung */
}
```

Das "typ" Feld hat den Wert 2 und beschreibt damit immer eine Konfigurationsnachricht. Das "src" und "bn" Feld beinhaltet die jeweilige einzigartige ID eines Gateways bzw. eines Sensors. Das "js" Feld beinhaltet eine in JSON beschriebene Konfiguration eines Sensors, eines Messwerts, eines Gateways oder ähnlichem.

Typ 4: Actuator Command

Um Aktoren zu steuern wird der Typ „Actuator Command“ benötigt. Hiermit lassen sich direkt Aktor Funktionen aufrufen und benutzen. Der Aufbau der „Actuator Command“ Nachricht basiert auf dem „Sensor Data“ Nachricht Aufbau.

```
{
        "typ":"4",              /* Protokoll Typ Nummer, hier 4 für Actuator Command */
        "src":"string",         /* ID des aufrufenden Services*/
        "dst":"string",         /* Gateway ID */
        "bn":"string",          /* Aktor ID */
        "seq":"string",         /* Optionale Sequenznummer */
        "fn":"string",          /* Funktionsname des Aktors */
        "e":[{
                "n":"string",   /* Funktionsparametername */
                "sv":"string"   /* Parameterwert */
        }]
}
```

Jedes Nachrichten Paket für „Actuator Command“s hat den Wert 4 im "typ" Feld. Das "src" Feld beinhaltet die ID des aufrufenden Services um ihn für Antworten adressierbar machen zu können. Das "dst" Feld beinhaltet die ID des Gateways, an das der anzusprechende Aktor angeschlossen ist. "bn" Feld beinhaltet die ID des Aktors der angesprochen werden soll. Das Feld "seq" ist optional und kann zur Identifikation von Antworten (Actuator Response, Type 5) genutzt werden. Das "fn" Feld beinhaltet den aufzurufenden Funktionsnamen des Aktors wie es von RPC[6] ähnlichen Interaktionen bekannt ist. Die für einen Aktor eines bestimmten Aktorprodukts zulässigen Funktionen sind in der Aktorproduktsemantik definiert (vgl. [DBAP3]). Im "e" Feld können Parameter für eine Funktion übergeben werden, das "n" Feld ist für den Parameternamen vorgesehen, das "sv" Feld für den Wert des Parameters. Mehrere Parameter können im gleichen "e" Feld angegeben werden, wohingegen mehrere Funktionsaufrufe nur über mehrere Payload Felder angegeben werden können.

Typ 5: Actuator Response

Die „Actuator Response“ Nachricht wird benutzt um Antworten von „Actuator Command“ Nachrichten zu verschicken. Für diese ist es zwingend notwendig, dass in der vorangegangen „Actuator Command“ Nachricht die Sequenznummer (Feld "seq“) vergeben wurde.

```
{
        "typ":"5",              /* Protokoll Typ Nummer, hier 4 für Actuator Response */
```

[6] Remote Procedure Call

```
        "src":"string",          /* Gateway ID */
        "dst":"string",          /* ID des aufrufenden Services */
        "bn":"string",           /* Aktor ID */
        "seq":"string",          /* Sequenznummer */
        "fn":"string",           /* Optionaler Funktionsname des Aktors */
        "e":[{
                "n":"string",    /* Funktionsparametername */
                "sv":"string"    /* Parameterwert */
        }]
}
```

Der Wert 5 im "typ" Feld beschreibt eine „Actuator Response“ Nachricht. In "src" und "dst" stehen die gleichen Werte wie in der vorrausgehenden „Actuator Command“ Nachricht mit getauschten "src" und "dst" Werten. Die Werte für "bn", "seq" und "fn" werden aus der „Actuator Command“ Nachricht übernommen. Das "n" Feld wird für eine Beschreibung der Antwort benutzt und das "sv" Feld für die Antwort selbst. Auch Fehlerwerte können so über Fehlercodes und Fehlertexte übertragen werden.

Sind mehrere Antworten von einer „Actuator Command“ Nachricht zu versenden, so müssen mehrere Pakete vom Typ 5 versendet werden, dies kann aber im gleichen SensorCloud Nachricht Paket geschehen.

Sicherheitserweiterung des SensorCloud Protokolls

Um die Sicherheit der SensorCloud Protokoll Pakete sicherzustellen wird JOSE (JSON Object Security) benutzt. JOSE ist ein sich bei der IETF in der Standardisierung befindender Vorschlag um Objektsicherheit in JSON Dokumenten zu gewährleisten.

```
{
        "typ":"1",               /* Protokoll Typ Nummer, hier 1 für Sensor Data */
        "src":"string",          /* Gateway ID */
        "bn":"string",           /* Sensor ID */
        "bt":"string",           /* Zeitstempel eines Sensor Data Pakets */
        "e":[{
                "n":"string",    /* physikalischer Name des Messwerts */
                "ev":[{
                        "unprotected":{                 /* unverschlüsselter Teil */
                                "alg":"dir",            /* direkter Modus */
                                "enc":"AESGCM256",      /* Verschlüsselungsalgorithmus */
                                "kid":"string",         /* Schlüsselhinweis */
                                "typ":"string"          /* Name des zu verschl. Feldes */
                        },
                        "iv":"base64string",            /* Initialisierungsvektor */
                        "ciphertext":"base64string",    /* verschlüsselter Wert */
                        "tag":"base64string"            /* Authentication Tag */
                ]}
        }]
}
```

Der Kopf einer verschlüsselten Nachricht bleibt bestehen. Veränderungen finden nur im "e" Feld statt. Anstatt, dass im unverschlüsseltem Feld "sv" direkt der Messwert abgelegt wird, wird im zu verschlüsselnden Fall ein Feld "ev" (encrypted value) erzeugt und mit weiteren Feldner versehen. Im Feld "unprotected" werden die öffentlichen Verschlüsselungsparameter beschrieben. Im Feld "iv" wird der Initialisierungsvektor vermerkt. Im Feld "ciphertext" steht die eigentliche verschlüsselte Nachricht/Messwert. Das Feld "tag" beschreibt den Authentification Tag.

2.2 Datenbank Management und Monitoring

2.2.1 Data Abstraction Layer (DAL) – SensorCloud RESTFull Webservices

Der Einsatz von Datenbanksystemen mit relationalen und nicht-relationalen Ansätzen bedingt unterschiedliche Zugriffsmöglichkeiten auf die Daten. So differieren die Datenbanksprachen und die Datenhaltung (normalisiert oder denormalisiert) je nach dem Datenmodell des eingesetzten Datenbanksystems (z. B. relationales Datenmodell vs. Datenmodell eines Wide Column Stores). Dies führt für die Sensordienste zu einer notwendigen Kenntnis über die eingesetzten Systeme und deren Datenhaltung. Zudem sind Änderungen an den eingesetzten Datenbanksystemen und implementierten Datenbankschemata nicht ohne gleichzeitige Änderung der Sensordienste umsetzbar.

Der Zugriff auf die Datenbanken wird daher für Sensordienste über eine Middleware (Database Abstraction Layer (DAL)) abstrahiert. Hierzu stellt der DAL geeignete Dienste für den Zugriff auf die Entitäten des konzeptionellen globalen SensorCloud-Datenbankschemas [1] bereit. Der DAL vereinfacht die Sicht auf die Datenbanken und ermöglicht eine Änderung an den eingesetzten Datenbanken und den implementierten Datenbankschemata ohne Änderungen an den Sensordiensten.

Der Database Abstraction Layer wird im Projekt SensorCloud über RESTful-Services realisiert [DBAP8]. Diese unterstützen eine horizontale Skalierung [FIE2000], wie sie aufgrund der Big-Data Problematik gefordert ist [MA-TP] und sichern gleichzeitig eine Abgeschlossenheit aller verteilten Datenbanktransaktionen des aufgerufenen Services und damit die Konsistenz der Daten. Denn im Unterschied zu einem RDBMS verfügt ein Wide Column Store DBMS über keinen inhärenten Transaktionsmechanismus. Dieser muss über geeignete Datenbankdienste, hier sind es RESTful-Services, bereitgestellt werden. Transaktionsmechanismen sind für Datenbankknoten einer TrustedCloud unerlässlich [MA-AS].

RESTful-Services verwenden das HTTP-Protokoll und sind von den LocationMastern nutzbar, sofern bei der Lokation des Kunden keine Sperrung von HTTP-Traffic in dessen Firewall eingerichtet ist. Dies muss im Projekt SensorCloud organisatorisch ebenso sichergestellt werden wie die Erreichbarkeit des SensorCloud-XMPP-Servers zur Übertragung von Messwerten und Aktornachrichten gemäß SensorCloud Protokoll. Weiterhin erben RESTful-Services alle bei HTTP eingesetzten Sicherheitsmechanismen. Dazu zählen die Verschlüsselung (vgl. [HUM2012]) und Authentifizierung über Transport Layer Security (TLS) [DIER2008] und Zertifikate. Eine ausführliche Beschreibung der Authentifizierungsaspekte im Zusammenhang mit dem DAL sind in [DBAP9] gegeben.

Ein wesentliches Mittel der Authentifizierung bei der Anmeldung an die DAL-Dienste der Cloud sind Zertifikate, die für jeden LocationMaster individuell ausgestellt und in dessen Software hinterlegt sind (LocationMaster-Zertifikat). Ebenso können Services der SensorCloud-Serviceplattform [Häuß2011] die DAL-Dienste der Cloud über deren individuellen Zertifikate verwenden (Service-Zertifikat). Zur Kontrolle der Authentizität des DALs verwenden dessen Dienste ebenso Zertifikate (Serverzertifikat).

Die RESTful-API des DALs wird zur Laufzeit über das JSON globale Schema generiert. Die API dient als Ersatz für lesende, schreibende, ändernde und löschende Zugriffe auf die SensorCloud-Datenbanken mit SQL-Statements aus der Data Manipulation Language (DML). Hierdurch kann für das Föderierte Datenbanksystem der SensorCloud sichergestellt werden, dass - im Sinne eines synchronen Datenbestandes zwischen den entfernten LocationMaster-Datenbanken und der zentralen SensorCloud-Datenbank - bei Änderungen an dem Cloud-Datenbestand die betroffenen LocationMaster über eine Tiefensuche identifiziert und an diese Änderungsnachrichten für deren lokalen Datenbestand verschickt werden können.

Ressourcen werden bei dem als RESTful-Service implementierten generischen DAL über die URI-Schreibweise identifiziert. Eine Ressource beschreibt ein oder mehrere Tupel einer Entität (ein oder mehrere Datensätze einer Tabelle). Die HTTP-Request-Methoden unterscheiden analog der verschiedenen DML-Schlüsselwörter die Art des Zugriffes auf eine Ressource. Der DAL nutzt für lesende Zugriffe *GET*- (analog dem SELECT-Statement in DML), für löschende Zugriffe *DELETE*- (analog dem DELETE-Statement in DML), für die Anlage oder Änderung neuer Ressourcen *PUT*- (analog dem INSERT- und dem UPDATE-

Statement in DML) und für die Verknüpfung von Ressourcen über deren Fremdschlüssel *POST-Requests*.

2.2.2 CopyCass und DatastoreSync

CopyCass

Das CopyCass [COPCA] Tool ist im Verlaufe des Projekts als Hilfsmittel für die Datenmigration von Cassandra Datenbanken unterschiedlicher Versionen entstanden. Die Notwendigkeit für dieses Tool entstand als Cassandra den Versionssprung von der Version 1.x zu 2.x vollzogen hat. Zusätzlich war das Datenbank Schema noch als relationales Datenbankschema angelegt und sorgte somit zu großen Performanceeinbrüchen bis hin zu Systemabstürzen, die durch die Überlast an Daten entstanden ist. Das größte Problem war der komplett Export der ca. 10 Millionen Messwerte mit einer Größe von ca. 2GB an Daten, der für die herkömmliche CQL Schnittstelle der Cassandra Datenbank eine zu große Last darstellte. Die Lösung des Problems konnte durch Gruppierung der Daten erreicht werden. Hierzu wurden nur die Messwerte, die innerhalb einer Stunde eines Sensors aufgelaufen sind, selektiert und exportiert. Um die Datenmenge immer weiter zu reduzieren wurden nach dem Export der Daten diese auch gleich aus der Datenbank gelöscht.

Als Exportformat werden herkömmliche SQL Insert-Statements erzeugt, die nicht nur von einem Cassandra 2.x System sondern auch von anderen Datenbankmanagementsystemen (PostgreSQL, MySQL, Derby, ...) gelesen werden können.

DatastoreSync

Das Tool DatastoreSync [SWP-DS] wurde zu Beginn des Projekts als Tool zur Datensynchronisierung entwickelt. Es hat die Aufgabe zwischen den in der Deploymentanalyse untersuchten Datenbanken Daten zu synchronisieren. Die zu synchronisierenden Datenbanken kommen aus dem Bereichen der relationalen Datenbanken (PostgreSQL, SQLite, MySQL), den objektorientierten (db4O) und den No-SQL Datenbanken (Cassandra, BerkeleyDB). Ursprünglich sollte das Tool in Richtung von LocationMaster zu Cloud als auch in Richtung Cloud zu LocationMaster synchronisieren, da die Entwicklung aber auf Grund der voranschreitenden Konzepte zur Synchronisierung in der SensorCloud eingestellt wurden, kann der DatastorSync nur von LocationMaster in Richtung Cloud synchronisieren.

Das Tool kann als JAVA Gui Programm genutzt werden oder aber auch Kommandozeilen basiert. Dies ist nötig um es dauerhaft als Hintergrundprozess laufen zu lassen um geänderte Datensätze sofort zu synchronisieren. Das Tool arbeitet nach dem Polling Prinzip, d.h. die Datenbank wird in kurzen Abständen zyklisch angefragt, ob neue Datensätze vorhanden sind. Im Positiven Fall werden diese ausgelesen und an die entsprechende Zieldatenbank geschickt.

Das Voranschreiten der SensorCloud stellte die Anforderung Daten von der Cloud aus an mehrere verschiedene LocationMaster zu pushen. Dies war mit Hilfe des Konzepts des DatastoreSync nicht zu erreichen, weswegen der DatastoreSync zu Gunsten des neuen Syncom/Commloss (vgl. 2.5.3) Konzepts nicht weiter entwickelt wurde.

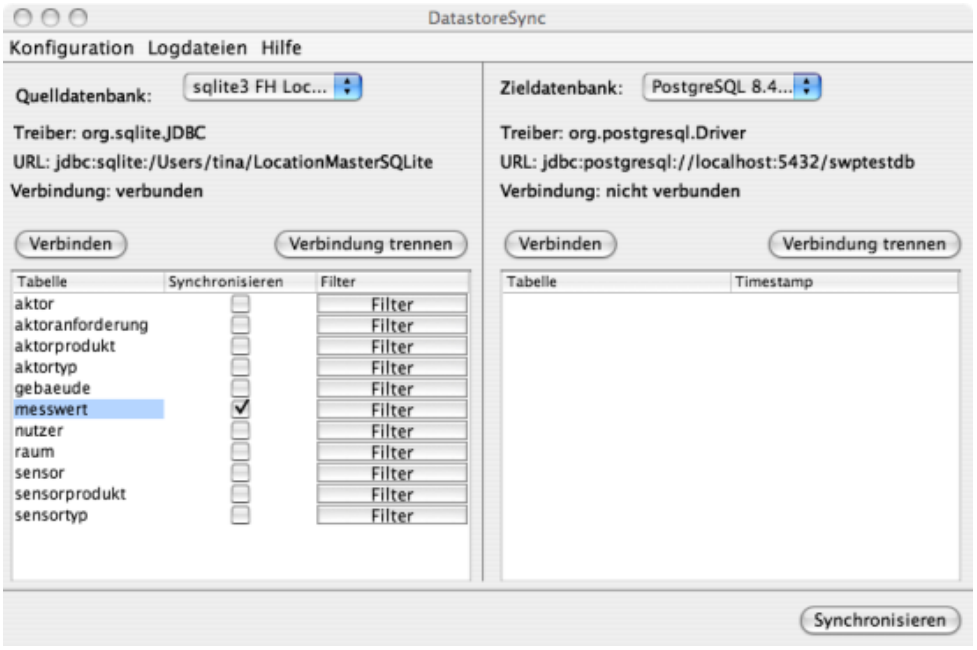

Abbildung 8: Ausschnitt DatastoreSync

2.2.3 Schema Monitor

Der Schema Monitor [DBAP5,PRAX-AK] ist ein weiteres Werkzeug um auf Veränderungen im konzeptionellen Schema und auch der real existierenden Schemata auf Gateway Seite und Cloud Seite reagieren zu können. Der Schema Monitor arbeitet dabei mit dem zentralen Abbild des konzeptionellen Schema, dem JSON Globalen Schema (vgl. 2.1.2).

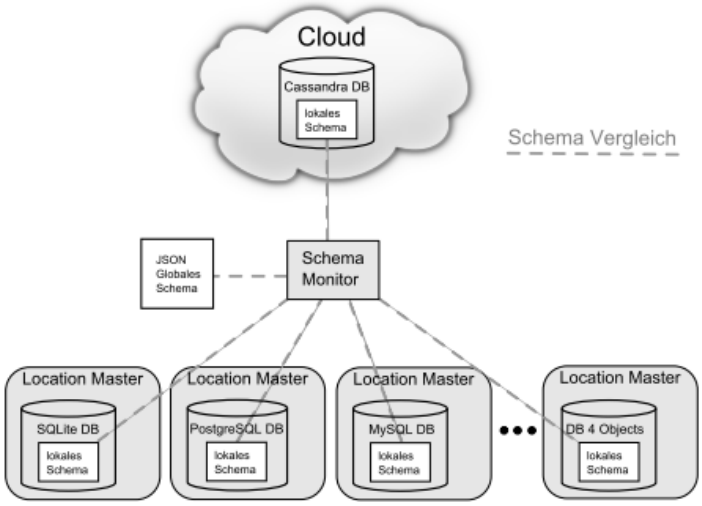

Abbildung 9: Schema Monitor Übersicht

Werden Änderungen an dem JSON Globalen Schema vorgenommen, so sollen diese durch den Schema Monitor in den lokalen Schemata aller Location Master Datenbanken sowie der Cloud Datenbank übernommen werden. Zusätzlich soll es die Möglichkeit geben bestimmte Änderungen der lokalen Schemata der Cloud oder Location Master Datenbanken, sowohl in das JSON Globale Schema, als auch auf die lokalen Schemata der restlichen Datenbanken, zu übernehmen.

Der Schema Monitor unterstürzt einige der in der Deploymentanalyse (vgl. 2.5.1) analysierten Datenbanken:

- Cassandra
- PostgreSQL
- MySQL
- SQLite
- DB4Objects

Zusätzlich stehen folgende Dienste vom Schema Monitor zur Verfügung:

- Vergleich von Schemata
- Backup eines lokalen Schemata in die Cloud
- Wiederherstellen eines Backups aus der Cloud
- Fenstermodus
- Hintergrunddienstmodus

Um unterschiedliche Datenbank Schemata vergleichen zu können müssen diese zunächst in eine vergleichbare Form gebracht werden. Relevante Informationen sind dabei Tabellennamen sowie Name und Datentyp der einzelnen Spalten. Die Form zum Vergleich der verschiedenen zu überwachenden Schemata hat folgenden Aufbau:

Index	Inhalt	Bedeutung
0	$$$	Trennzeichen
1	Sensor	Start der Tabelle Sensor
2	SenID	Name der ersten Spalte der Tabelle Sensor
3	text	Datentyp der ersten Spalte der Tabelle Sensor
4	SenDatEin	Name der zweiten Spalte der Tabelle Sensor
5	timestmp	Datentyp der zweiten Spalte der Tabelle Sensor
6	$$$	Trennzeichen
7	Aktor	Start der Tabelle Aktor
8	AktBez	Name der ersten Spalte der Tabelle Aktor
9	text	Datentyp der ersten Spalte der Tabelle Aktor
10	AktDatEin	Name der zweiten Spalte der Tabelle Aktor
11	timestmp	Datentyp der zweiten Spalte der Tabelle Aktor

Abbildung 10: Vergleichsformat des Schema Monitors

Datentypen-Bezeichnungen müssen normalisiert werden, da unterschiedliche DBMS unterschiedliche Datenbanktyp-Bezeichnungen benutzen. Um festzustellen in welche Richtung gelöscht, erzeugt oder bearbeitet werden darf, muss eine Berechtigungsmatrix erstellt werden. Diese kennt drei zu überwachende Einheiten: JSON Globales Schema, Cloud DB und die lokalen DBs der LocationMaster.

	Tabelle erzeugen	Tabelle löschen	Spalte erzeugen	Spalte löschen	Datentyp ändern
JSON Schema	x	x	x	x	x
Cloud DB	x		x		
Location Master DB					

Abbildung 11: Schema Monitor Berechtigungsmatrix

Stellt der Schema Monitor Unterschiede in den Schemata der verschiedenen DBMS der SensorCloud fest, kann er auf unterschiedliche Weise mit den Änderungen umgehen. Läuft der Schema Monitor als Hintergrunddienst, so orientiert er sich an Konfiguration und Berechtigungsmatrix umso zu entscheiden, ob

Änderungen (erzeugen/löschen/bearbeiten) direkt ausgeführt werden oder nur Änderungsskripte erzeugt werden sollen. Änderungsskripte können zu späteren Zeitpunkten manuell von einem Administrator ausgeführt werden oder lokale Änderungen zurück genommen werden.

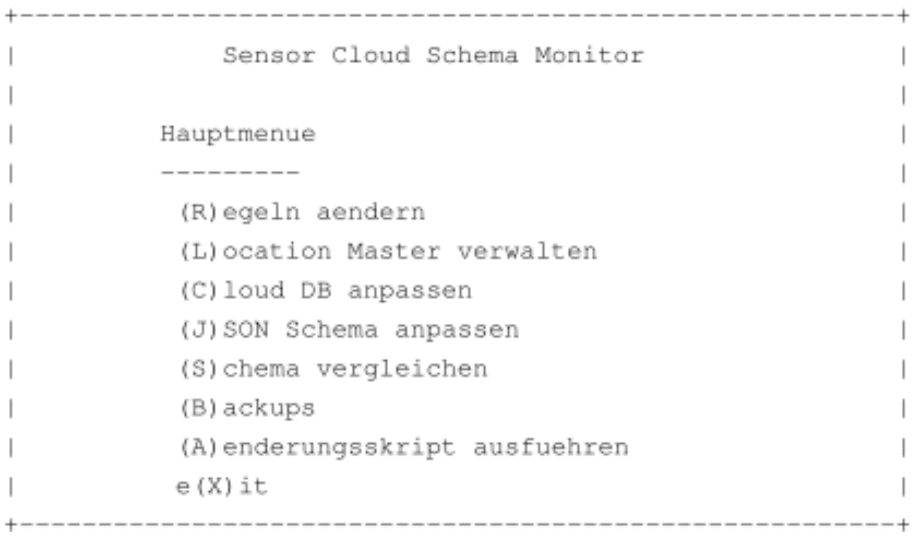

Abbildung 12: Hauptmenü des Schema Monitors

2.2.4 Invertiertes Dateisystem

Für die Speicherung anfallender Dateien in der SensorCloud wurde neben Datenbanksystemen ein invertiertes Dateisystem für Text und Binärdateien realisiert [PRAX-TP].

Die Vorteile eines dateisystembasierten Datenspeichers auf Cloud-Seite sind u. a. die einfache Replizierbarkeit auf mehrere Knoten, die transparente Verschlüsselung von Dateien auf Dateisystemebene, der Redundanzaufbau mit lokalen oder auf mehreren physikalischen Knoten verteilten RAID-Systemen (wie z. B. DRBD[7]) , die Lastverteilung über mehrere Knoten, die einfache Datensicherung, und die für einen Datenbankersatz unterstützenden Mechanismen eines POSIX-konformen Dateisystems (Konsistenz des Dateisystems, Zugriffskontrolle).

In einem Dateisystem wird ein Wurzelverzeichnis für die Dateispeicherung gewählt. Ein Daemon überwacht die Änderungen in dem gewählten Verzeichnis und seinen Unterverzeichnissen (Watch Daemon). Bei Änderungen werden die Metadatenscanner informiert, die für die Invertierung des Dateityps der geänderten Datei zuständig sind. Durch die Klassifizierung von Dateien über Medientypen lassen sich dateityp-spezifische Algorithmen zur Gewinnung der Metadaten-Informationen schreiben. Zum Beispiel für die Gewinnung der Exif-Informationen aus den Bilddateien eines Bildsensors.

Die Implementierung nutzt einen geclusterten Aufbau mit einer Socketkommunikation über TCP/IP (Abbildung 13). Die Trennung des Watch Daemons und der Metadatenscanner ermöglicht eine horizontale Skalierbarkeit durch Hinzufügen weiterer Rechnerknoten. Der Watch Daemon überwacht die Änderungen eines lokalen Dateisystems, welches als Network Attached Storage (NAS) an die Metadatenscanner-Knoten angebunden ist. Auf den angebundenen Knoten laufen 1-N Metadatenscanner, die für das Scannen verschiedener Dateitypen in verschiedenen Verzeichnissen zuständig sind. Die Metadatenscanner registrieren sich an dem Watch Daemon mit der Angabe über ihre Zuständigkeit für Medientypen und

[7] http://www.drbd.org

Verzeichnisse. Der Watch Daemon informiert bei Änderungen im Dateisystem die zuständigen Metadatenscanner anhand dieser Informationen. Die Worker Threads der Metadatenscanner indizieren anschließend die Datei und speichern die extrahierten Informationen in einer dem Metadatenscanner dedizierten lokalen Datenbank.

Werden identische Metadatenscanner auf mehreren Knoten gestartet, so ist deren Datenbestand identisch. Durch die Möglichkeit der Implementierung einer Abfrage-API in die Metadatenscanner kann aufgrund der dedizierten Datenbanken mithilfe von Lastverteilern ein hohes Abfrageaufkommen mit hoher Zuverlässigkeit und Ausfalltoleranz verarbeitet werden. Zusätzlich können die Metadatenscanner durch die Angabe von einer Maximalzahl von parallel laufenden Worker Threads für eine optimale Ressourcen-Ausnutzung weiter skaliert werden.

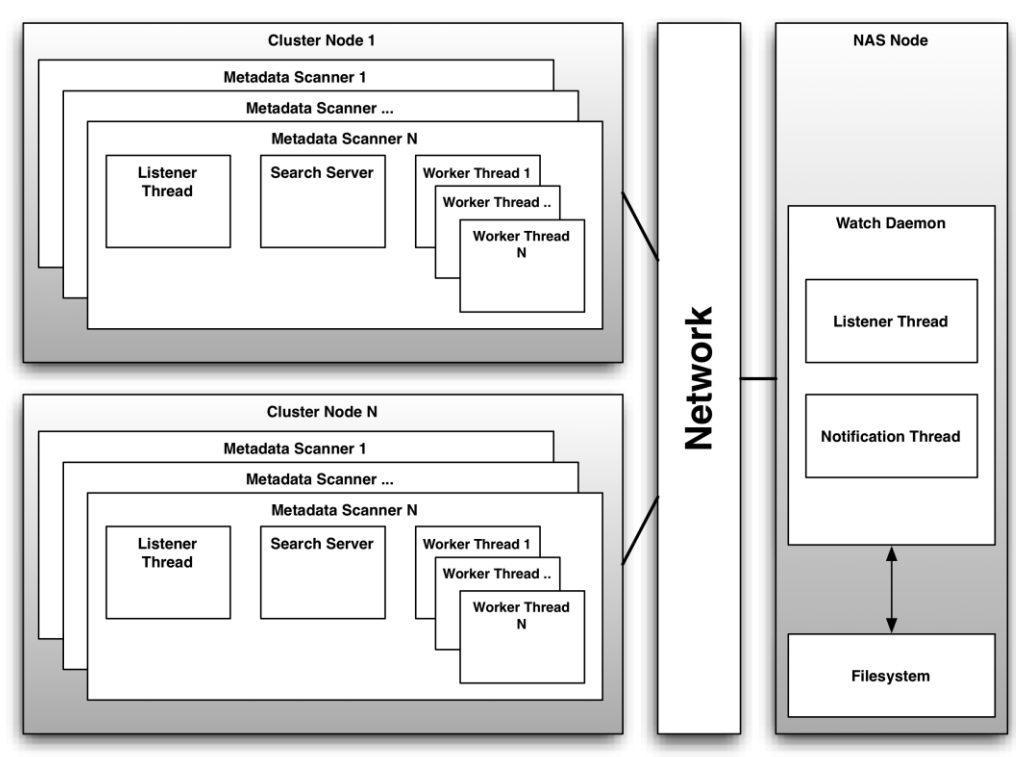

Abbildung 13: Horizontale Skalierbarkeit durch den Einsatz mehrerer Knoten

2.2.5 Früher Prototyp der Software des LocationMaster

Um in der frühen Phase innerhalb der ersten zwölf Monate neben den theoretischen Konzepten auch praktische Erfahrung einfließen lassen zu können wurde eine rudimentäre Teststrecke entwickelt. Diese Teststrecke enthielt von Beginn an die typischen Komponenten der jetzigen SensorCloud. Dies beinhaltet Sensoren und Aktoren, eine Datenbank mit entsprechenden Zugriffsmöglichkeiten und eine frühe Version eines LocationMasters mit entsprechender Software.

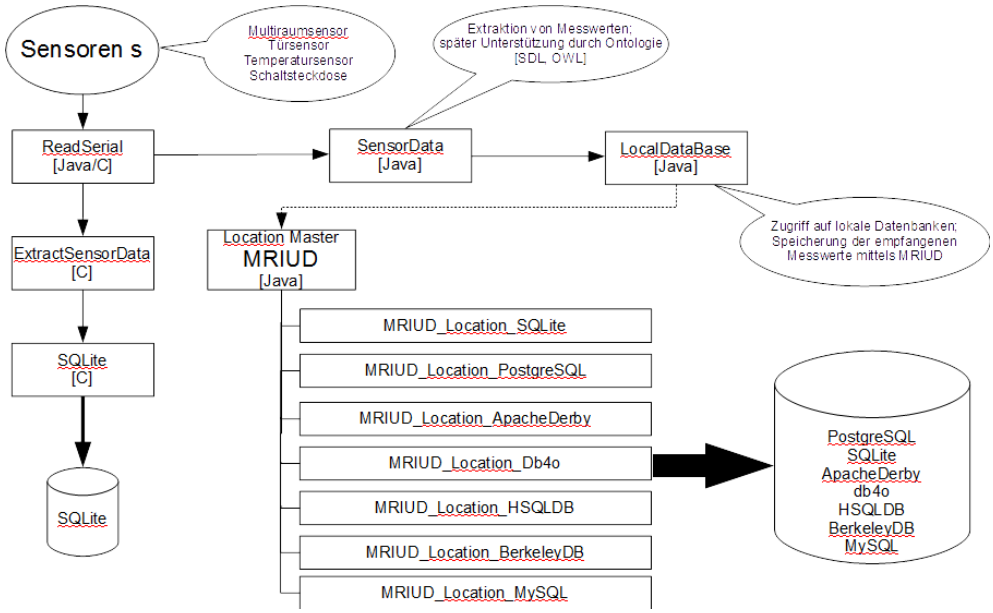

Abbildung 14: früher Prototyp SensorCloud

SensorDataToLocationMaster

Die SensorDataToLocationMaster Komponente auf dem Prototyp LocationMaster hat mehrere Dienste innerhalb eines Tools vereinigt:

- Auslesen der Schnittstellen zu Sensoren
- Schreiben in Schnittstelle zu Aktoren
- Weiterleiten der gelesenen Werte in föderiertes DBMS

Das Auslesen der verschiedenen Schnittstellen (Bluetooth, CC1101, Ethernet, …) erfolgt im Read Serial Modul, welches sowohl als C wie auch als Java Bibliothek zur Verfügung steht. Über das Read Serial Modul werden nicht nur Messwerte ausgelesen sondern auch Aktor Kommandos an die Aktoren geschickt.

Nachgelagert an das ReadSerial Modul existiert ein Modul, welches zur Vorverarbeitung von gelesenen Messwerten dient. Das Modul SensorData weiß dabei über den Aufbau und die Beschaffenheit eines gesendeten Messwerts Bescheid und konvertiert diesen Messwert in eine allgemeingültige lesbare Form für die SensorCloud. Das SensorData Modul hat dabei fest implementierte Informationen über Sensoren, die auch in der SensorProduktSemantik (vgl. 2.1.2) eines Sensors vorliegt.

Die extrahierten und vorformatierten Messwerte werden über das Modul MRIUD (**M**anage **R**ead **I**nsert **U**pdate **D**elete) in das föderierte Datenbanksystem der SensorCloud geschrieben. Das MRIUD Modul ist eine frühe Version einer Datenabstrahierungsschicht. Sie bietet die Möglichkeit Messwerte persistent zu speichern ohne zu wissen, welches DBMS aktuell für die Speicherung von Messwerten zuständig ist. Dies erlaubte eine frühe Syncom Commloss Strategie, bei der ohne Internet Verbindung Messwerte in eine lokale Gateway Datenbank geschrieben wurde oder bei bestehender Verbindung zur Cloud Messwerte direkt in die Cloud Komponente des FDBMS geschrieben wurde.

SensorCloud Protokoll über XMPP Strecke

Nachrichten der SensorCloud, die zwischen dem LocationMaster und den Cloud Komponenten ausgetauscht werden sollen, können mittels XMPP (E**x**tensible **M**essaging and **P**resence **P**rotocoll) bidirektional verschickt werden.

2.2.6 Android Manager

Die unter Android entwickelte Applikation Android Manager [BA-AS] dient der Bearbeitung von Stammdaten, der Erstellung von Regeln und Verbünden und der Anzeige von Messwerten der SensorCloud. Der Android Manager ist eine native Android Applikation, die mit Hilfe von Rest Servicen (Vorversion des DAL, vgl. 2.2.1) mit der Cloud Datenbank kommuniziert. Die App wurde so entwickelt, dass ein Nutzer der SensorCloud sich mit seinen Zugangsdaten an der SensorCloud anmeldet und seine Sensoren und Aktoren, seine Regeln und zugehörige Daten angezeigt bekommt. Der Nutzer ist beschränkt auf die ihm gehörigen Daten und kann auch nur dort Veränderungen durchführen.

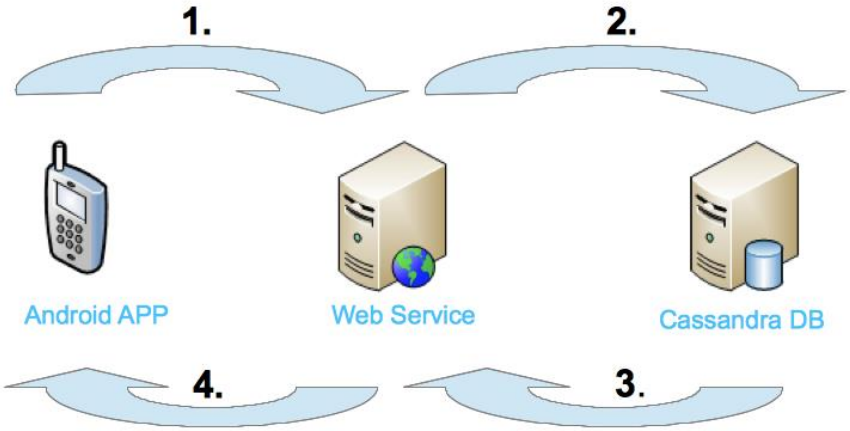

Abbildung 15: Systemübersicht Android Manager

Die Android App bietet dabei folgende Funktionalitäten:

- Einloggen oder das Registrieren eines Nutzers.
- Anzeigen und Bearbeiten von Nutzerstammdaten.
- Anzeigen, Bearbeiten und Hinzufügen von Telefondaten.
- Anzeigen, Bearbeiten und Hinzufügen von E-Maildaten.
- Anzeigen, Bearbeiten und Hinzufügen von Sicherheitsdaten.
- Anzeigen und Bearbeiten von Adressdaten.
- Anzeigen von Sensoren und Erstellen von „Sensor“-Verbünden.
- Anzeigen von Aktoren und Erstellen von „Aktor“-Verbünden.
- Darstellen der Messwerte in einem Diagramm.
- Anzeigen von Events und Ändern der Daten eines Events.
- Erstellen und Anzeigen von Gruppen.
- Einladen von Mitgliedern in eine Gruppe.
- Anzeigen und Erstellen von Servicelinien.

Abbildung 16: Hauptmenü SensorCloud Android Manager

2.3 Messwerte und deren Visualisierung

Für die Visualisierung der in der Datenbank der SensorCloud persistierten Sensordaten wurden mehrere Anwendungen implementiert, die die übertragenen Messwerte grafisch als Diagramm darstellen. Hierbei werden die Diagramme auf Client-Seite generiert, um die alleinige Entschlüsselung von verschlüsselten Messwerten [HUMM2012] durch den Anwender zu gewährleisten.

Die Anwendungen benutzen die Entitäten Sensor, SensorProdukt, Messwert und MessLinie des konzeptionellen Schemas [DBAP4]. Über die Entitäten Sensor und Messwert werden die Sensoren ermittelt, die Messwerte in die SensorCloud übertragen haben. SensorProdukt enthält die benötigte Semantik eines Sensors (Physikalischer Name des Messwerts, Aggregatfunktionen und Umrechnungsfunktionen). Die Entität MessLinie liefert die Tage, an denen Messwerte einer Messgröße zur Verfügung stehen.

2.3.1 Implementierung der Messwert Entität auf Cassandra

Folgende Erkenntnisse und die daraus resultierende Datenmodellierung der Entität Messwert entstammen aus [MA-TP] und werden hier größtenteils vom Autor des Ursprungwerkes an dieser Stelle zusammengefasst.

Bei dem eingesetzten Wide-Column-Store Cassandra benötigen - im Gegensatz zu relationalen Datenbanksystemen – Zugriffe auf gespeicherte Messwerte bei einer optimierten Datenhaltung eine konstante algorithmische Laufzeit. Eine Speicherung der Messwerte muss hierfür in auf die späteren Abfragen hin optimierten ColumnFamilies erfolgen. Die Implementierung der Entität Messwert in Cassandra als ColumnFamily eines KeySpaces wurde unter der Kenntnis der späteren Abfragen eines Visualisierungstools für Messwerte mit einer chronologischen Selektion der Messwerte vorgenommen (siehe 2.3.2.).

Besteht in Cassandra der Primary Key einer Tabelle (Column Family) aus mehreren Attributen, so wird dieser in Cassandra *Composite Key* genannt. Der Composite Key unterteilt sich in den *Partition Key* (auch *Row Key* genannt) und den *Clustering Key*. In Cassandra bestimmt der Partition Key, auf welchem Knoten ein Datensatz liegt. Der Clustering Key bestimmt, wie die Daten auf dem Knoten gespeichert werden. Dabei werden die Daten anhand der Attribute des Clustering Keys gruppiert und sortiert in einer durch den Row Key identifizierten Wide Row abgelegt.

Unter Berücksichtigung dieser Punkte kann die Messwert ID (MesWerID) als Primary Key der Entität Messwert des konzeptionellen Schemas nicht als Partition Key genutzt werden. Bei einer für eine Generierung von Messwertkurven angenommenen chronologischen Selektion der Messwerte bewirkt die Messwert ID als Partition Key eine Verteilung aller Messwerte aller Sensoren über das gesamte Cluster. Dies führt bei einer Selektion aufgrund der Netzwerkkommunikation und der auftretenden Netzwerklatenz zu einer höheren Laufzeit gegenüber einem sequentiellen Lesen zusammenhängender Daten aus einer Wide Row von einem Knoten.

Da bei einer Diagrammgenerierung eine chronologische Selektion der Messwerte eines Sensors vorgenommen wird, qualifiziert sich die Sensor ID als Partition Key. Hierdurch werden alle Messwerte eines Sensors auf einem Knoten gespeichert. Durch eine ausbalancierte Verteilung aller an das System angeschlossenen Sensoren auf alle Knoten des Clusters entstehen bei identischen Sendeintervallen keine Hotspots.

Ein Sensor misst 1 bis n mit $(n \in \mathbb{N})$ physikalische Größen (Temperatur, Luftdruck, Strömungsgeschwindigkeit, Energie, Leistung, ...). Für die Auswertung der Messwerte verschiedener physikalischer Größen werden unterschiedliche statistische Funktionen verwendet. Daher wird der Clustering Key um ein neues Attribut erweitert, welches die gemessene physikalische Größe beschreibt. Dies bewirkt eine zusammenhängende Speicherung der Messwerte eines Sensors zu einer physikalischen Größe. Dazu wird das Messwert-Tupel, welches bisher die unterschiedlichen Messwerte zu einem Zeitpunkt kommasepariert darstellte, erweitert. Zwei neue Attribute (MesWerNam und MesWerWer) werden eingeführt. Die vorher als Tupel zusammengefassten Messwerte unterschiedlichster Art erhalten dadurch den gleichen Zeitstempel, die gleiche Sensor ID und die gleiche Messwert ID. Dies führt zu einer Duplizierung von Informationen. Das bisherige Attribut MesWerTup wird entfernt.

Für einen beschleunigten Zugriff auf die Messwerte eines Tages werden die Messwerte anhand eines Tageszeitstempels (MesWerTimStaDay) im Clustering Key weiter gruppiert. Analog eines Primary Keys in einem RDBMS muss der Composite Key in Cassandra eindeutig sein. Daher wird für den Clustering Key ein Attribut benötigt, welches den Composite Key eindeutig identifiziert. Hierzu wird der Zeitstempel des Messzeitpunktes (MesWerTimSta) gewählt. Jeder Sensor kann bzw. sollte zu einem Zeitpunkt eine physikalische Größe nur einmal messen. Dies muss algorithmisch sichergestellt werden.

Große Unterschiede in den zu erwartenden Datenmengen der Sensoren können bei dieser Modellierung zu einem nicht ausbalancierten Cluster führen, was sich in einer unterschiedlichen Speicherplatzbelegung und Auslastung der einzelnen Knoten wiederspiegelt. Dies ist in der Speicherung sämtlicher Messwerte eines Sensors in einer Row begründet. Bei einer ungünstigen Balancierung entstehen dadurch Hotspots mit Engpässen der Speicherkapazität an einzelnen Knoten.

Durch Hinzunahme des Messwertnamens (Attribut MesWerNam) in den Partition Key können verschiedene Messwerte verschiedener Messgrößen auf unterschiedliche Rechner abgebildet werden. Dies verbessert die Verteilung der Messwerte, kann aber weiterhin zu einer nicht ausbalancierten Verteilung führen. Eine Gruppierung von Messwerten nach physikalischen Messgrößen in Wide Rows gleicht der Anlage von mehreren Sensoren, die jeweils nur Messwerte zu einer physikalischen Größe liefern. Dies verzögert, aber verhindert nicht, die Entstehung von Hotspots und einer ungleichmäßigen Speicherplatzbelegung.

Bei Hinzunahme des Tageszeitstempels in den Partition Key, steigt die Wahrscheinlichkeit einer ausbalancierten Verteilung der Messwerte, da die Anzahl der Elemente des Urbildes für die Abbildungsfunktion des Partitioners weitaus höher ist. Die Elemente ergeben sich aus der resultierenden mehrdimensionalen Tabelle über die Sensoren, deren Messgrößen und die Tage, in denen ein Sensor Messwerte zu einer Messgröße lieferte.

$$[MesWerSenID] \times [MesWerNam] \times [MesWerTimStaDay]$$

Folgendes CREATE-Statement ergibt sich für eine Datenmodellierung der Entität Messwert in Cassandra.

```
CREATE TABLE "Messwert" (
  "MesWerSenID" text,
  "MesWerNam" text,
  "MesWerTimStaDay" text,
  "MesWerTimSta" text,
  "MesWerID" text,
  "MesWerWer" text,
  PRIMARY KEY (("MesWerSenID", "MesWerNam", "MesWerTimStaDay"), "MesWerTimSta")
) WITH CLUSTERING ORDER BY ("MesWerTimSta" DESC);
```

Abbildung 17 zeigt die Messwertverteilung auf einem Vier-Knoten-Cluster bei unterschiedlich ausgeprägten Partition Keys nach dem Einspielen von 14.837.119 Messwerten. Bei dem Cassandra optimierten Schema erfolgt eine ausbalancierte Verteilung der Daten auf allen Knoten mit 25,00% ± 0,60% (24,42%, 24,51%, 25,15% und 25,92%).

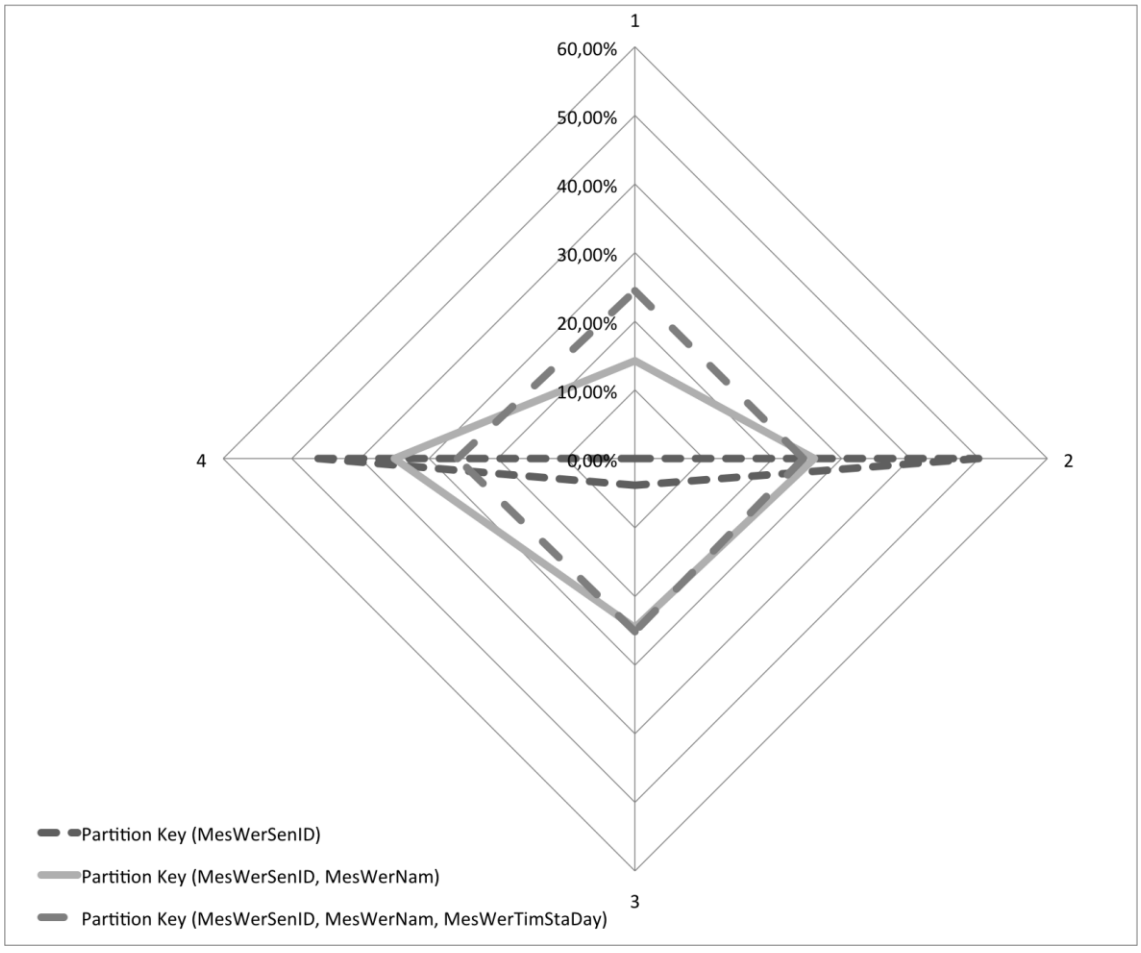

Abbildung 17: Gegenüberstellung der Messwertverteilung bei unterschiedlichen Partition Keys

2.3.2 Darstellung der Messwerte der SensorCloud

Der Zugriff auf in der Cloud gespeicherte Messwerte erfolgt über eine als Webservice realisierte API. Die API kennt die Datenhaltung der Messwerte in der Cloud-Datenbank und bietet eine einheitliche Zugriffsart auf die unveränderten und auch aggregierten Messwerte eines Zeitraums.

Die API für den Zugriff auf die Messwerte eines zeitlichen Intervalls wird über einen parametrisierbaren Webservice bereitgestellt und dient als Ersatz für lesende Zugriffe auf die in den SensorCloud-Datenbanken gespeicherten Messwerten. Hierdurch wird für alle nutzenden Programme die gleiche Performance bei lesenden Zugriffen gesichert. Gleichzeitig stellt die API integrierte Aggregationsfunktionen auf Messwerte bereit [DBAP8]. Anwendbare Aggregationsfunktionen auf eine Messentität sind in der Sensorsemantik definiert. Voraussetzung zur Nutzung des Messwert-Webservice auf verschlüsselte Messwerte ist die Autorisierung des Service durch den Besitzer der Messwerte unter dem Gesichtspunkt von TrustedCloud.

Eine Darstellung der Messwerte der SensorCloud erfolgt über eine Selektion und Aggregation der Messwerte über den bereitgestellten Messwertservice und eine anschließende Visualisierung der Messwerte über eine Webanwendung [SCSTAT] oder über plattformabhängige Anwendungen wie Androidanwendungen. Die Anwendungen benutzen die in der jeweiligen Programmiersprache zur Verfügung stehenden Visualisierungsbibliotheken, wie z.B. Google Charts[8] für Webanwendungen oder für die Android Chart Engine[9] für Android Apps. Abbildung 18 zeigt die Visualisierung von Messwerten über einen Webservice.

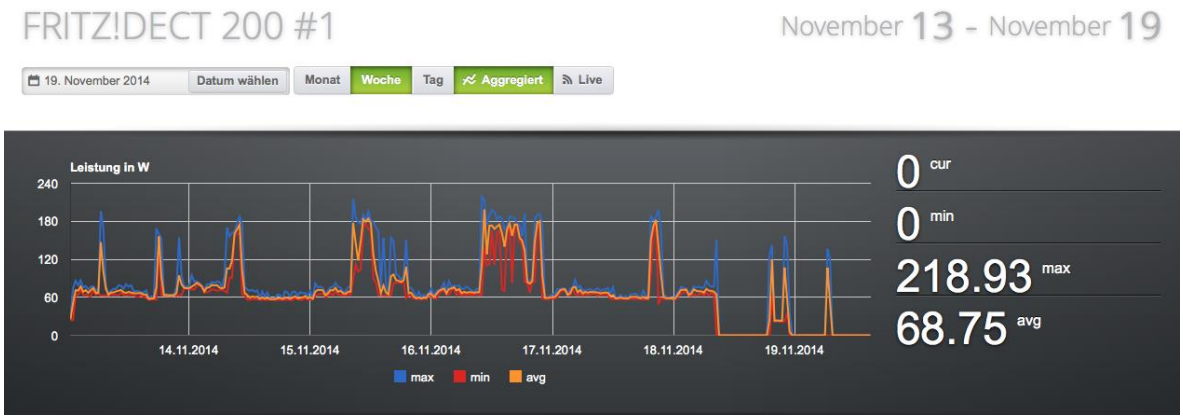

Abbildung 18: Darstellung von Messwerten der SensorCloud über einen Webdienst

2.3.3 Darstellung von Messwerten einer Photovoltaik Anlange

Eine weitere prototypische Anwendung ist die Aufnahme und Darstellung von Werten einer Photovoltaik Anlage [BA-BS]. Hierzu wurde ein eigenständiges Gateway entwickelt. Das Gateway übernimmt die Aufnahme der Werte vom Wechselrichter, bereitet diese auf und verschickt diese mittels SensorCloud Protokoll an die SensorCloud weiter.

[8] Developers.google.com/chart
[9] https://code.google.com/p/achartengine/

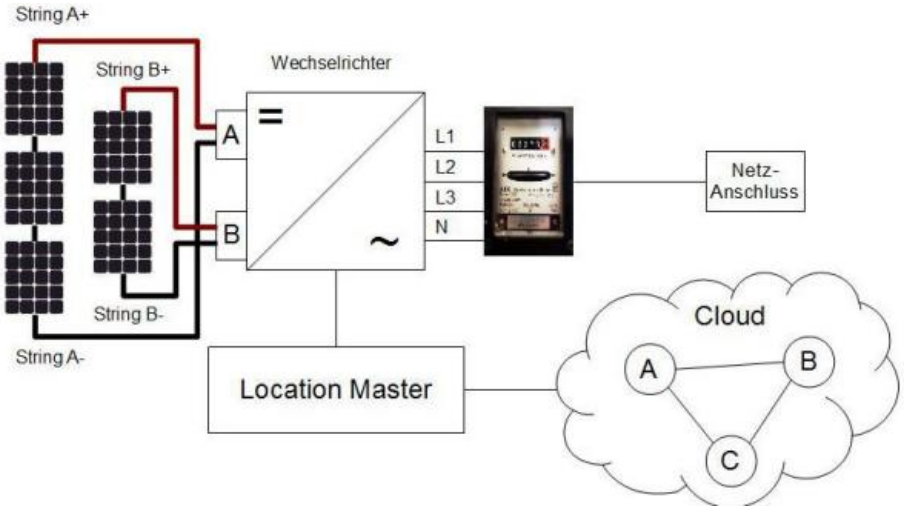

Abbildung 19: Photovoltaik in der SensorCloud

Die Photovoltaik Anlage ist ein erster Versuchsaufbau, der extern zum Projekt aufgebaut wurde und zusätzlich ein externes Gateway, welches nicht direkt aus dem Projekt SensorCloud entstanden ist, an die SensorCloud ankoppelt. Das externe Gateway ersetzt dabei den LocationMaster, der sonst als Standard Gateway der SensorCloud eingesetzt wird. Kompatibilität mit der SensorCloud wird dadurch erreicht, dass das Gateway das von der SensorCloud vorgegebene Protokoll spricht und die Anlage als Sensor mit entsprechender semantischer Beschreibung angelegt wurde

.

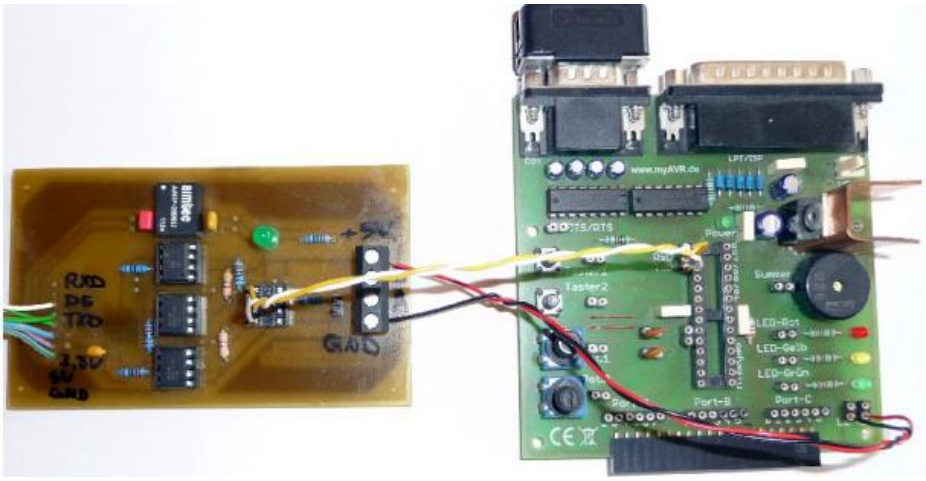

Abbildung 20: Gateway der Photovoltaik Anlage

Die Visualisierung der aufgenommenen Werte der Photovoltaik Anlage können, nachdem sie in der Cloud Datenbank der SensorCloud eingetroffen sind, übersichtlich dargestellt werden. Hierzu wurde eine Webanwendung in PHP geschrieben, die die Messwerte der Photovoltaik Anlage anschaulich und insbesondere im Vergleich zueinander darstellt. So lassen sich bspw. einzelne Strings, derzeitige Leistungsaufnahme und die Summe eines Tages in einem Diagramm darstellen und auswerten.

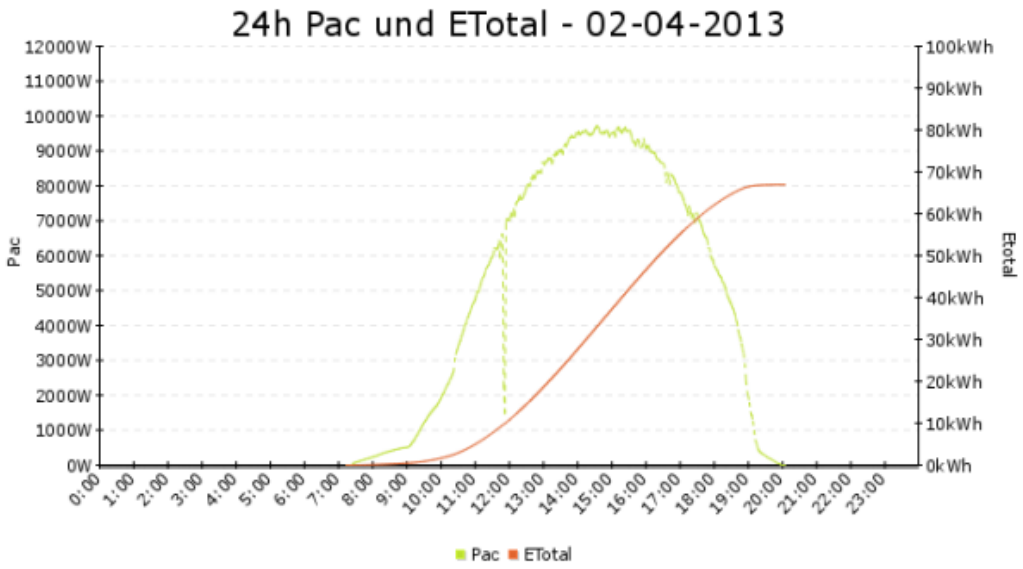

Abbildung 21: Visualisierung der Messwerte einer Photovoltaik Anlage

2.3.4 Darstellung von Energieverbräuchen (Regel-, Nichtregelenergie)

Kern dieser Arbeit [BA-MK] ist die Darstellung und Unterscheidung von Verbrauchern, die je nach Energiesituation geregelt werden dürfen oder im Umkehrschluss eben nicht. Hierfür müssen Verbraucher gekennzeichnet werden, ob sie von außen geregelt werden dürfen. Nicht regelbare Verbraucher sind Verbraucher, die auf kontinuierlichen Strom angewiesen sind und keine Toleranz gegenüber Verfügbarkeit haben. Ein Kühlschrank ist ein Beispiel für ein Gerät das auf kontinuierlichen Strom angewiesen und unzulässig für externe Regelung ist. Andere Geräte hingegen könnten durch externe Faktoren, wie gerade günstig hergestellter Strom oder kurzfristiger Ausfall eines anderen Strom Großabnehmers, geregelt werden und so nur zur bestimmten Uhrzeiten betrieben und von extern gesteuert werden. Diese Geräte können deswegen als regelbare Verbraucher bezeichnet werden und sind nicht zwangsläufig auf einen kontinuierlichen Fluss von Energie angewiesen. Als Beispiel für einen regelbaren Verbraucher bietet sich ein elektrisches Fahrzeug an. Diesem ist es egal, wann es in der Nacht geladen wird, wichtig ist nur, dass es irgendwann in der Nacht geladen wird.

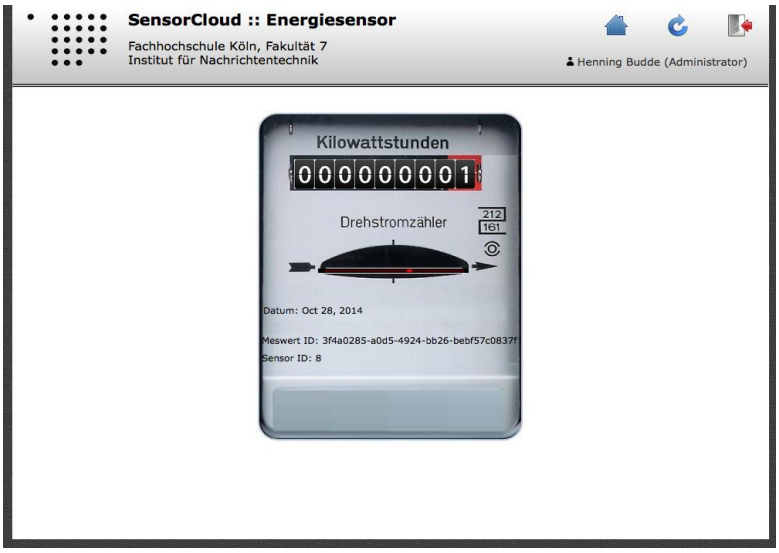

Abbildung 22: Energieverbauchsanzeige

Die Applikation ist als „responsive“ Webapplikation implementiert, d.h. sie kann auf verschiedenen Geräten wie PCs, Tablets oder auch SmartPhones ausgeführt werden und skaliert je nach Bildschirmgröße die Ansicht der Applikation.

Abbildung 23: Leistungsanzeige

2.4 Virtuelle Sensoren und Aktoren

Virtuelle Sensoren und Aktoren sind SensorCloud-Dienste, die eine Funktionalität bereitstellen und sich wie Sensoren und Aktoren verhalten. Es wird anhand des Funktionsumfangs zwischen virtuellen Sensoren, virtuellen Aktoren und virtuellen Aktor-Sensoren unterschieden. Je nach Einteilung verfügen die Dienste über eine Sensor-ID und/oder Aktor-ID und lassen sich über den Hardware Abstraction Layer (HAL[10]) bzw. den Service Abstraction Layer (SAL[11]) in das Regelwerk einbinden. Hierbei kann sowohl auf Nachrichten von

[10] Der HAL ist eine auf eine auf dem LocationMaster (Gateway) installierte Komponente (vgl. 3.1)
[11] Der SAL ist eine in der Cloud installierte Software Komponente (vgl. 3.1)

SensorCloud-Diensten im Regelwerk reagiert werden als auch SensorCloud-Dienste über das Regelwerk getriggert werden.

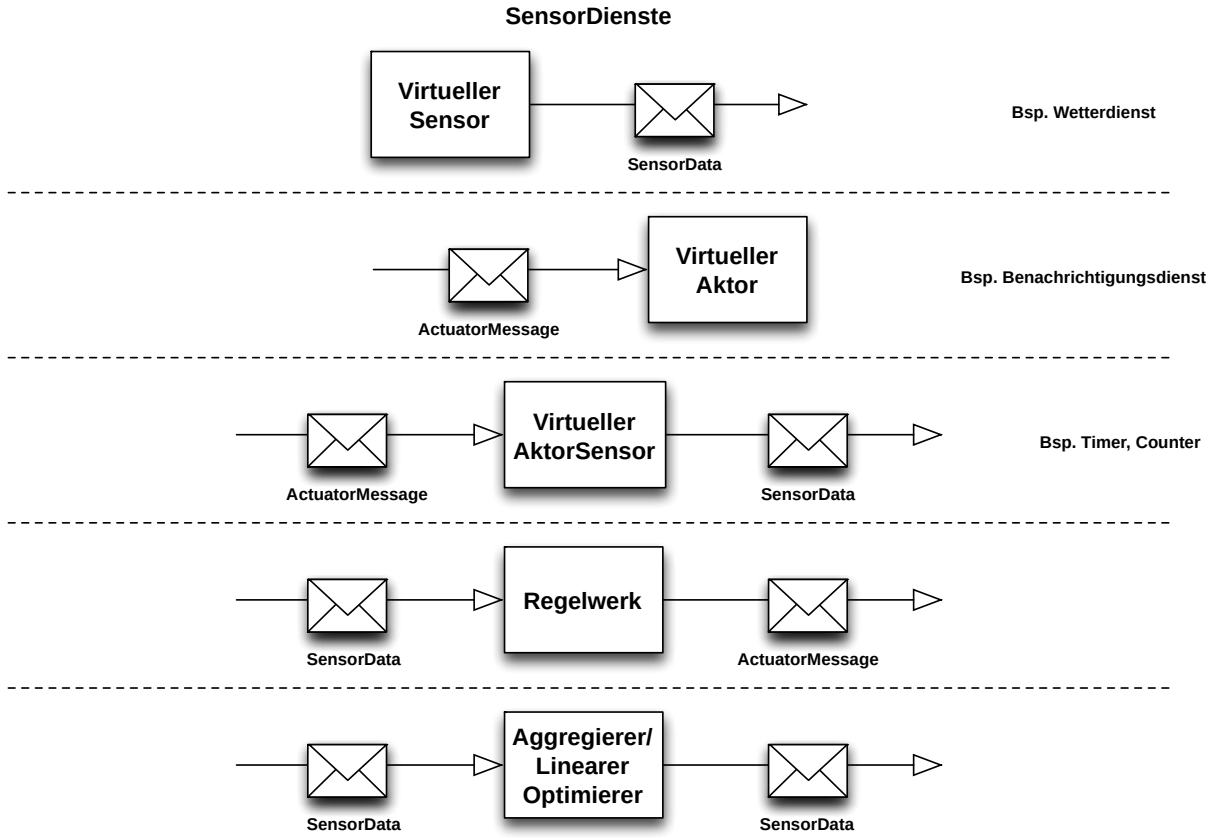

Abbildung 24: SensorCloud-Dienste in der Übersicht

SensorCloud-Dienste lassen sich anhand ihrer Eingangs- und Ausgangsnachrichten in virtuelle Sensoren (senden nur SensorData), virtuelle Aktoren (empfangen nur ActuatorMessages) und virtuelle AktorSensoren (empfangen ActuatorMessages und senden SensorData) klassifizieren. Weiterhin sind auch SensorCloud-Dienste für die Aggregierung von Messwerten oder der Lösung von linearen Optimierungsaufgaben denkbar. Diese Dienste empfangen und senden SensorData-Nachrichten.

2.4.1 Virtuelle Sensoren

Als Beispiel eines virtuellen Sensors sei ein Wetterdienst genannt, der Wetterdaten auswertet und Wetterwarnungen als Messwerte generiert.

Hierzu wurde in der Fachhochschule Köln ein Wetterdienst implementiert, der externe Informationsquellen auswertet und über das Regelwerk mit Messwerten physikalischer Sensoren verknüpft [BA-SW]. So kann der Verlauf einer Schlechtwetterfront analysiert werden und Wetterwarnungen für zukünftig betroffene Gebiete generiert werden.

In dem genannten Beispiel des Wettersensors werden Wetterdaten als Messwerte über den Nachrichtentyp 1 in das Cloud-Regelwerk gesendet, um z. B. entfernte Aktoren von Fenstern eines Hauses zu steuern.

```
{
      "typ": "1",
      "gw": "string",
      "bn": "015dacf0-93ef-11e3-baa8-0800200c9a66",
      "bt": "4857531546289",
      "e": [
            {
```

```
                "n": "Orkan",
                "t": "0",
                "sv": "true"
            }, …
        ]
}
```

Skripting 1: Sensordata eines Wetterdienstes

Der Cloud-Regelinterpreter wertet dann die in der SensorCloud generierten Messwerte aus, verknüpft sie mit den entfernten Messwerten (z.B. Schließkontakte) und führt anhand des Regelwerkes Aktionen aus (z. B. alle Fenster zu schließen).

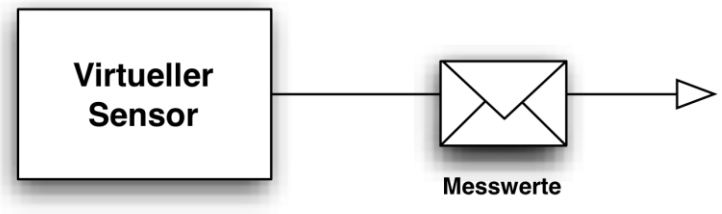

Abbildung 25: Virtueller Sensor

2.4.2 Virtuelle Aktoren

SensorCloud-Dienste können auch als virtuelle Aktoren implementiert werden. So ist es denkbar einen Benachrichtigungsdienst als virtuellen Aktor zu implementieren. Die Parametrisierung des virtuellen Aktors erfolgt hierbei über den Nachrichtentyp 4.

Ein Benachrichtigungsdienst kann als Beispiel die Funktionen sendeEmail(), sendeSMS() oder rufeAn() bereitstellen, die mit den Parametern Nachricht in Kombination mit Email-Adresse, Handynummer oder Telefonnummer aufgerufen werden können. Eine Parametrisierung eines solchen Benachrichtigungsdienstes über den Nachrichtentyp 4 sieht wie folgt aus:

```
{
        "typ"  : "4",
        "src"  : "97771f01-93ef-11e3-baa8-0800200c9a66",
        "bn"   : "a47d78c0-93ef-11e3-baa8-0800200c9a66",
        "bt"   : "4857531543156",
        "fn"   : "sendeEmail",
        "e" :[
                {
                        "n":"Email",
                        "sv":"empfanger@domain.tld"
                },
                {
                        "n" :"Nachricht",
                        "sv":"Hallo Welt!"
                },...
    ]
}
```

Skripting 2: ActuatorMessage eines Benachrichtigungsdienstes

Der Benachrichtigungsdienst wertet die ActuatorMessage aus und versendet anhand seiner Parametrisierung die entsprechende Nachricht.

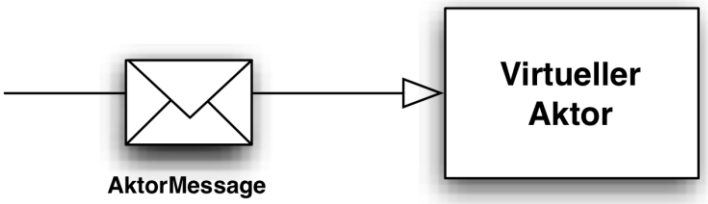

Abbildung 26: Virtueller Aktor

2.4.3 Virtuelle AktorSensoren

Virtuelle AktorSensoren sind Dienste, die wie ein Aktor parametrisierbar sind und wie ein Sensor Messwerte produzieren. Dies können z. B. ein Timer oder ein Datumsdienst sein.

Ein Timer lässt sich mit einem zu messenden Zeitintervall konfigurieren (z. B. 30 Sekunden) und liefert nach Ablauf des Intervalls einen Messwert zurück (z. B. Timeout = True).

Abbildung 27: Virtueller AktorSensor

Ein als virtueller AktorSensor realisierter Timer kann für die Erstellung eines Regelwerkes mit zeitlichen Abhängigkeiten genutzt werden. Eine zeitabhängige Lichtschaltung, die sich nach 30 Sekunden wieder abschaltet, lässt sich analog zu der Beschreibung in Kapitel 2.1.2 im Datenmodell wie folgt realisieren:

Tabelle 1: Entität SensorEvent

SenEveID	SenEveQueID	SenEveQue	SenEvePhyNam	SenEveVop	SenEveWer
4711	S1	Sensor	Zustand	=	An
4712	VAS2	Sensor	Timeout	=	True
4713	VAS2	Sensor	UUID	=	9999

In der Entität SensorEvent werden Events von Sensoren hinterlegt. Hierbei sei S1 ein Lichtschalter und VAS2 die Sensor-ID des als virtuellen AktorSensor realisierten Timers. Der Timer liefert nach Ablauf des parametrisierten Zeitintervals zwei Messwerte zurück. Diese sind „Timeout = True“ und „UUID = 9999“. UUID ist hierbei eine Identifikationskennzahl, die für eine eindeutige Zuordnung bei der Parametrisierung des Timers mit angegeben werden kann.

Tabelle 2: Entität EventMitglieder

EveMitID	EveMitEveID	EveMitSenEveID	EveMitRei
103	2030	4711	1
104	2031	4712	1

105	2031	4713	2

In der Entität EventMitglieder werden SensorEvents bestimmten Events zugeordnet. Zum Beispiel werden die SensorEvents 4711 dem Event 2030 und 4712 und 4713 dem Event 2031 zugeordnet.

Tabelle 3: Entität Event

EveID	EveArt	EveBez
2030	400	Licht an
2031	400	Licht aus nach 30 Sekunden

In der Entität Event ist das Event mit einer Bezeichnung und einer Eventart hinterlegt. Die EventArt kann als Priorität interpretiert werden: EveArt > 1000: Alarm (zeitkritisch); EveArt < 500: Regeln (nicht zeitkritisch).

Tabelle 4: Entität EventAktion

EveAktiID	EveAktiEveID	EveAktiBez	EveAktiZieID	EveAktZie	EveAktiZiePar	EveAktiZieWer	EveAktiPrio
8002	2030	Licht	A1	Aktor	LichtAnFunktion	{ "n":"Licht", "sv":"anschalten" }	1
8003	2030	Benachrichtigung	VA1	Aktor	Sende-Email	{ "n":"Email", "sv":"empfanger@domain.tld" }, { "n" :"Nachricht", "sv":"Licht an!" }	2
8004	2030	Timer	VAS1	Aktor	Starte-Timer	{ "n":"Timer", "sv":"30" }, { "n" :"uuid", "sv":"9999" }	1
8005	2031	Licht	A1	Aktor	LichtAusFunkti	{	1

					on	"n":"Licht", "sv":"ausschalten" }	

8006	2031	Benachrichtigung	VA1	Aktor	SendeSMS	{
						"n":"Handynummer",
						"sv":"0177/7777777"
						},
						{
						"n" :"Nachricht",
						"sv":"Licht aus nach 30 Sekunden!"
						}
							2

In EventAktion ist die durch ein Event zu startende Aktion hinterlegt. Dazu wird das Ziel (z.B. ID A1 in der Entität Aktor) abgelegt. In diesem Beispiel ist A1 die Aktor-ID der Lampe, VA1 die Aktor-ID des Benachrichtigungsdienstes und VAS1 die Aktor-ID des als virtuellen AktorSensor realisierten Timers mit der Sensor-ID VAS2.

2.5 Lokale Datenbank

2.5.1 Deployment Analyse

Ein Ziel der Deploymentanalyse [DBAP2] war das Finden geeigneter DBMS für die DB des LocationMasters. Innerhalb dieses Arbeitspaketes ist für das Zielsystem ein optimales Produkt unter den Bedingungen einer public license, dem Umfang des darstellbaren Datenmodells und den Programmieranforderungen in Hinsicht auf die Verwaltung der Sensordaten auszuwählen.

Die Kriterien, nach denen eine Entscheidung für geeignete DBMS getroffen werden soll, lassen sich in vier Kategorien unterteilen:

1. Kapazitätsverbrauch
2. Modellierungseigenschaften
3. Trusted-Aspekt
4. Kosten

Die folgenden acht Kriterien werden als relevant für die Auswahl geeigneter DBMS für den LocationMaster gesehen und diesen Kategorien wie folgt zugewiesen:

1. Kapazitätsverbrauch:
    a. Zeitverbrauch
    b. Speicherverbrauch
2. Modellierungseigenschaften:
    a. Datenmodell
    b. Erweiterbarkeit
3. Trusted-Aspekt:
    a. Sicherheit
    b. Transaktionen

   c. Kommunikation
4. Kosten:
   a. Lizenz

Zu Testzwecken wurden sieben verschiedene DBMS auf dem Gateway (LocationMaster) installiert und ausprobiert:

- SQLite
- db4o
- Apache Derby
- Berkeley DB
- HSQLDB
- PostgreSQL
- MySQL

Als Testumgebung wurde ein erster Prototyp der SensorCloud Teststrecke (vgl. 2.2.5) benutzt.

Das Ergebnis der Deploymentanalyse ergab, dass sich auf der Teststrecke alle DBMS verwenden ließen und die Möglichkeit bestehe jedes dieser DBMS auf dem LocationMaster zu verwenden. Keines der DBMS fällt soweit ab, dass man von einer Benutzung abraten würde. Um die oben erwähnten Kriterien zu gewichten wurden verschiedene Szenarien angenommen.

Szenario 1: Einfamilienhaus

Ein Einfamilienhaus mit 50-100 Sensoren, die jeweils pro Minute einen Messwert abgeben. Somit müsste das Gateway und die lokale Datenbank max. 1,6 Messwerte pro Sekunden verarbeiten können.

Szenario 2: Hotel

Ein Hotel (oder auch Mehrfamilienhaus) besitzt bis zu 1000 Sensoren, die alle über das gleiche Gateway angebunden sind. Gateway und Datenbank sollten nun mit 16 Messwerten pro Sekunden klar kommen. In diesem Szenario muss beachtet werden, dass der Anschluß von 1000 Sensoren auch die Ressourcen des Gateways deutlich stärker beansprucht werden als es im Fall des Einfamilienhaus ist.

Da das Projekt SensorCloud mit beiden Szenario arbeiten möchte, hat sich in der Deploymentanalyse die Datenbank PostgreSQL als Kandidat für eine auf dem Gateway betriebene Datenbank ergeben. Die PostgreSQL unterstützt nicht nur schnelle Lese- und Schreibvorgänge sondern ist auch Ressourcen schonend. Im Vergleich zu manch einer Embedded Datenbank (bspw. SQLite) fällt sie zwar etwas ab, dafür sind diese eingeschränkter in ihrer Erweiterbarkeit und Netzkommunikation.

Trotz Empfehlung der Deploymentanalyse ist das FDBS der SensorCloud so aufgebaut, dass sämtliche anderen untersuchten Datenbanken flexibel anstelle der PostgreSQL Datenbank genutzt werden können. Dies bedeutet, dass ein Hersteller , der eigene Gateway Funktionalitäten in der SensorCloud wahrnehmen möchte, auch auf andere DBMS setzen kann, weil seine Schwerpunkte anderes verteilt sind.

2.5.2 Transaktionen

Das Hauptziel des Arbeitspakets [DBAP6] zur Untersuchung von Transaktionen für Sensordaten ist die Entwicklung von Diensten wie Transaktionen, die zwischen den Gateways und der SensorCloud durchgeführt werden können. Dabei sollen die Transaktionen einerseits von der SensorCloud und andererseits von dem Gateway initialisiert werden können (Pull- und Push-Prinzip), wie die beiden

nachfolgenden Abbildungen verdeutlichen sollen.

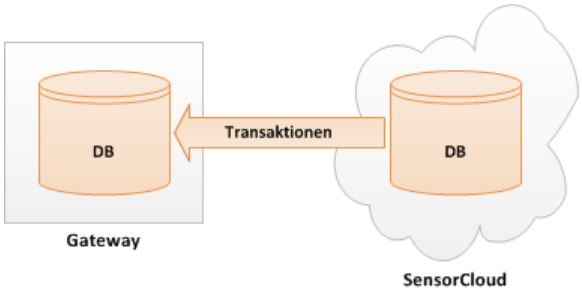

Abbildung 28: Transaktionen initialisiert von der SensorCloud (Pull-Prinzip)

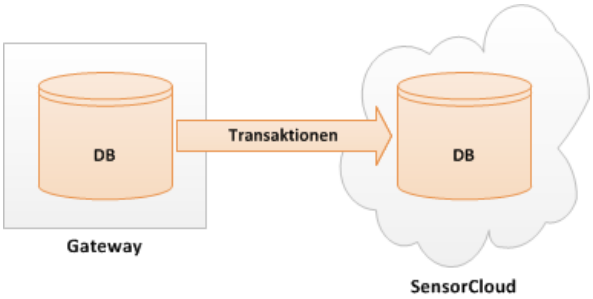

Abbildung 29: Transaktionen initialisiert vom Gateway (Push-Prinzip)

Robuste Transaktionsmechanismen sind heutzutage weit erforscht und eine Selbstverständlichkeit für Schreib- oder Lesezugriffe innerhalb eines Datenbankmanagementsystems (DBMS). Vorausgesetzt, dass die Engine des DBMS auch Transaktionen unterstützt (Stichwort Transaktionssicherheit). Kommerzielle wie auch OpenSource Transaktionsmanager übernehmen die Verwaltung und Durchführung einer Transaktion auf einem DBMS.

Um eine verteilte Transaktion, wie in den beiden vorgestellten Szenarien durchführen zu können, wurde das X/Open DTP Modell genutzt, genauer der X/Open XA-Standard der Open Group, welches in der Java Transaction API (JTA) implementiert ist.

Die JTA beschreibt eine Programmierschnittstelle für verteilte Transaktionen über mehrere XA-Ressourcen (das können z.B. Datenbanken sein). Der Zweck der JTA ist es eine lokale Java Schnittstelle zu definieren, welche der Transaktionsmanager für sein Transaktionsmanagement in verteilten Java Enterprise Umfeld unterstützt.

Die JTA Schnittstelle ist ein Bestandteil der JavaEE. Möchte man anstatt der JavaEE (Java Enterprise Edition) jedoch die JavaSE (Java Standard Edition) verwenden, so muss die Implementierung der JTA in Form eines Transaktionsmanager zugrunde gelegt werden. Bekannte Open Source Transaktionsmanager sind z.B.:

- Atomikos TransactionEssentials
- JBoss Transaction Service
- Bitronix Transaction Manager
- Java Open Transaction Manager

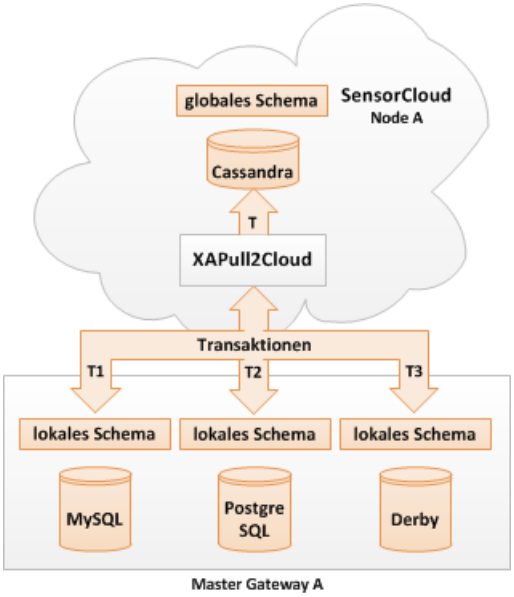

Abbildung 30: XA-Transaktionen in der FDBS-Teststrecke

Die Transaktionskonzepte nach dem Pull- und Push-Prinzip bieten für die SensorCloud eine flexible Art der Transaktionsinitialisierung. Soll die SensorCloud selbst die Initialisierung der Transaktionen auf die Gateways durchführen (Pull-Prinzip), so entlastet dies die Gateways. Die SensorCloud behält die Kontrolle über die zu verwaltenden Transaktionen. Jedoch müssen die DBMS auf dem Gateway einen Zugriff aus der Ferne zulassen (Client/Server Modell).

Für die SensorCloud sind die Gatways nicht autonom. Sie dienen der SensorCloud nur als Datenpuffer, bis die nächste Transaktion aus der SensorCloud auf die Gateways initialisiert wird.

Für die Initialisierung der Transaktionen vom Gateway aus (Push-Prinzip) spricht einerseits, dass auch DBMS-Systeme, die keinen Zugriff aus der Ferne zulassen, auf dem Gateway verwendet werden können (z.B. SQLite). Hierbei muss das Gateway selbst die Transaktionen Richtung SensorCloud initialisieren und kontrollieren.

Die Gateways sind bei der Anwendung des Push-Prinzips autonom. Sie warten nicht nur darauf, dass die Daten wie nach dem Pull-Prinzip von ihnen geholt werden, sondern leiten die Transaktionen selbst in die Wege. Das ist vor allem dann ein Vorteil, wenn z.B. ein Sensor einen Alarm meldet, oder ein kritischer Messwert eines Sensors überschritten wird. Ein Analyseprogramm könnte diesen Alarm erkennen und die Transaktion in die Cloud initialisieren, so dass der Kunde direkt über diesen Messwert informiert wird.

Verteilte und klassische Transaktionen, wie in [DBAP6] untersucht, beeinflussen bei einem verteilten und parallelisierten System wie der SensorCloud die Systemverfügbarkeit. Klassische Lockingmechanismen sperren einen Datensatz bei Änderungen für einen Zeitraum, was nach dem CAP-Theorem nach Brewer [BRECAP] bei einem stark verteilten System mit hohem Grad an Parallelität zu Problemen in der Verfügbarkeit führt (Beispiel: Mehrere Services verschiedenster Diensteanbieter in der Cloud versuchen einen gelockten Datensatz zu ändern, der aber aufgrund eines Netzwerkfehlers zum LM nicht wieder freigegeben wurde). Aus diesem Grunde benutzen verteilte NoSQL-Systeme auch kein Pessimistic Locking sondern ein Optimistic Locking mithilfe von Vector-Clocks.

Im Projekt SensorCloud wurde aufgrund der asynchronen Kommunikation zwischen den LocationMastern und der Cloud eine zweistufige Synchronisation von Datenänderungen eingeführt, welche im folgenden Kapitel beschrieben ist.

2.5.3 Cloud-Synchronisierung von Stammdaten

Die Cloud-Synchronisierung von Stammdaten unterscheidet zwischen einer inkrementellen und einer vollständigen Synchronisierung mit den LocationMastern. Hierbei ist eine einseitige Synchronisierung in Richtung LocationMaster aufgrund von ausschließlich lesenden Zugriffen auf die LocationMaster-Datenbanken vorgesehen.

Bei der vollständigen Synchronisierung erhalten die LocationMaster die Stammdaten (u. a. angeschlossene virtuelle und physische Sensoren und Aktoren, Regeln und Semantiken) über einen Dienst des DALs. Der DAL-Synchronisierungsdienst (einer der DAL-Dienste aus Kapitel 2.2.1) wird vom LocationMaster in periodischen Zeitabständen aufgerufen, um im Falle eines eventuellen Verlusts von durch die Cloud versandten inkrementellen Änderungsnachrichten wieder einen konsistenten Zustand der LocationMaster-Datenbank herzustellen. Über das LocationMaster-Zertifikat stellt der DAL-Synchronisierungsdienst sicher, dass nur die für einen LocationMaster bestimmten Daten übermittelt werden.

Die inkrementelle Synchronisierung wird über einen lokalen Sensordienst auf dem LocationMaster realisiert, der die inkrementellen Änderungen über ActuatorMessages empfängt (LM-Synchronisierungsdienst – in Abbildung 31 als „Incremental Synchronisation“ bezeichnet). Der LM-Synchronisierungsdienst ist hierbei als virtueller Aktor [VIRTSEN] auf den LocationMastern implementiert. Der DAL identifiziert bei verändernden Zugriffen auf den Cloud Datenbestand die tangierten LocationMaster über die betroffenen Entitäten, die mit den tangierten LocationMastern über Fremdschlüssel verbunden sind und adressiert die ActuatorMessage gezielt an deren lokale LM-Synchronisierungsdienste. Die lokalen LM-Synchronisierungsdienste empfangen die ActuatorMessage mit dem Inhalt der durchzuführenden Änderung (INSERT, UPDATE, DELETE) und führen diese in der lokalen Datenbank der LocationMaster aus.

Abbildung 31 zeigt den Nachrichtenfluss einer cloudseitig erzeugten ActuatorMessage vom DAL bis zum Synchronisierungsdienst des LocationMasters. Der Nachrichtenfluss der ActutatorMessage ist in der Abbildung durch graue Briefumschläge und gestrichelte Flusslinien gekennzeichnet.

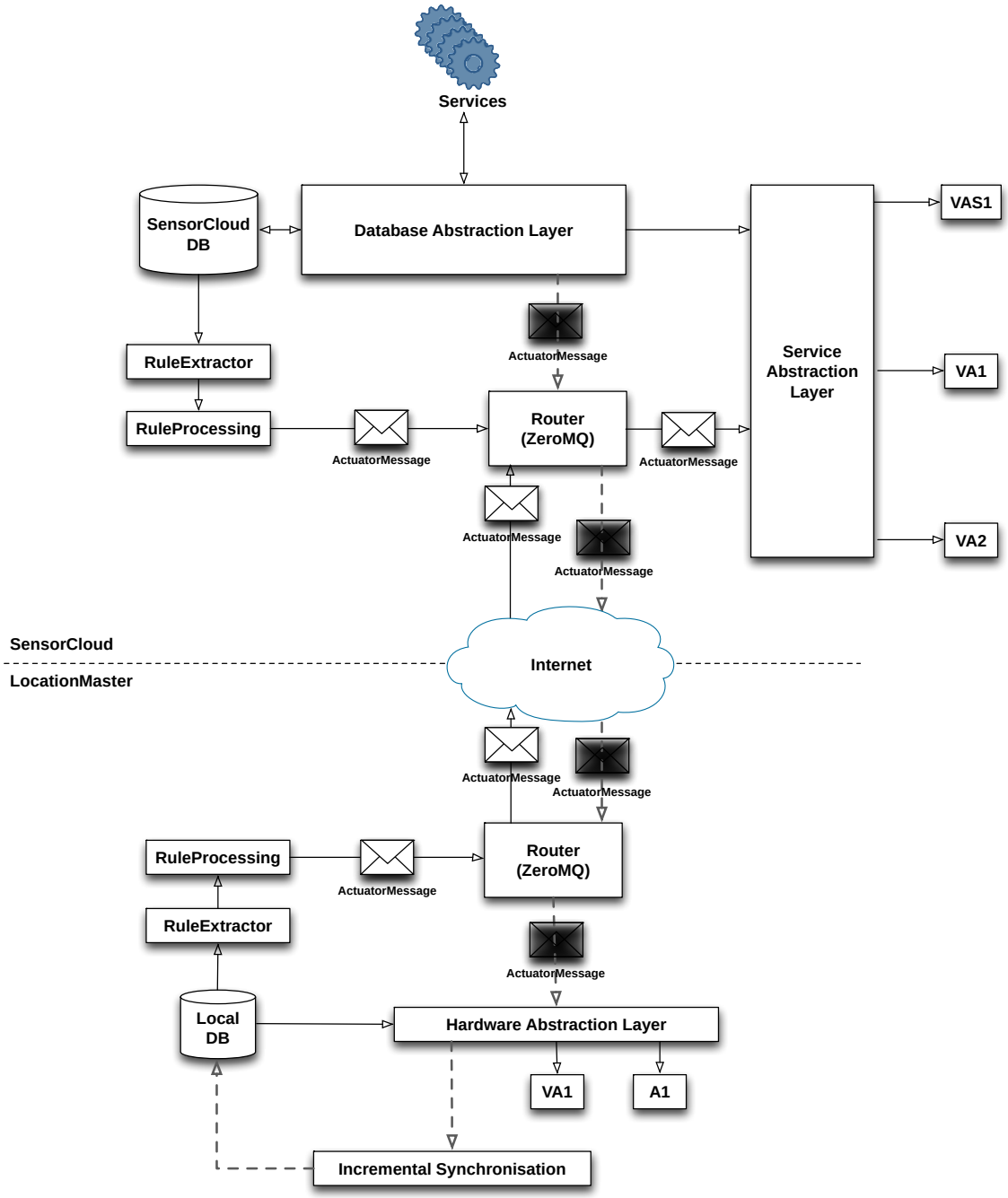

Abbildung 31: Inkrementelle Synchronisation via HAL (LocationMaster)

2.5.4 Cloud-Synchronisierung von Messwerten

Bei bestehender Verbindung des LocationMasters zur SensorCloud werden Messwerte kontinuierlich als SensorData über einen XMPP-Client in Richtung SensorCloud versendet. Die Cloud-Synchronisierung von Messwerten synchronisiert die bei einem Kommunikationsausfall zwischenzeitlich auf dem LocationMaster angefallenen Messwerte mit der Cloud-Datenbank. Im Falle eines Kommunikationsausfalls beschreibt Abbildung 32 den veränderten Nachrichtenfluss der Messwerte.

Tritt eine Störung der Verbindung zu dem XMPP-Server der SensorCloud auf (1), wird eine SensorData Nachricht mit dem Sensorwert „offline=true“ für das Regelwerk des LocationMasters generiert (2). Über eine Regel des Regelwerks wird eine an den Hardware Abstraction Layer (HAL) adressierte ActuatorMessage

erzeugt, die die Funktion des HALs zum Umadressieren der zu versendenden SensorData-Nachrichten triggert (3). Nach Ausführen der durch die ActuatorMessage aufgerufenen Funktion werden neue Messwerte als SensorData-Nachrichten an den lokalen Syncon Comloss Dienst adressiert (4). Der Syncon Comloss Dienst speichert die Messwerte in der lokalen Datenbank (5) und nimmt ggf. bei einem längeren Kommunikationsausfall eine Verdichtung der Messwerte vor (6).

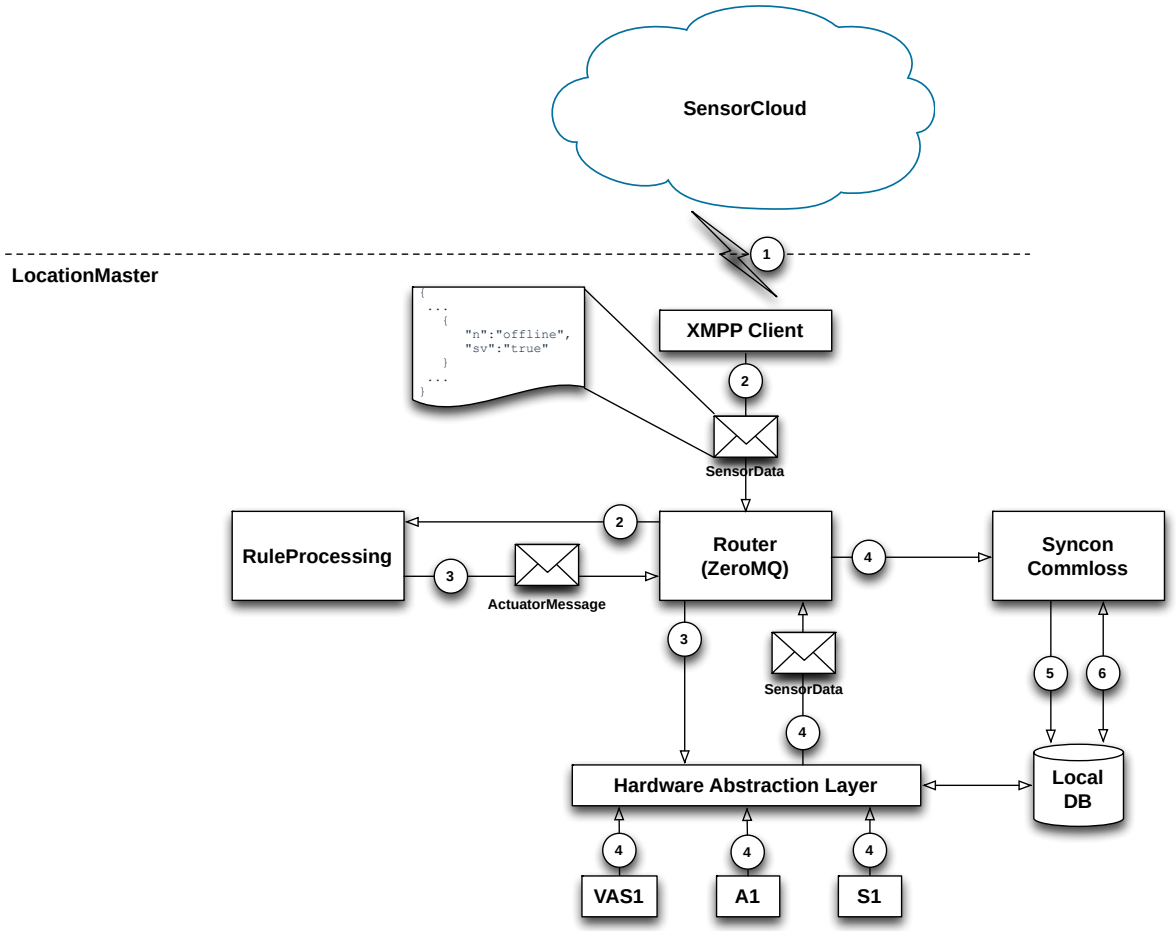

Abbildung 32: Nachrichtenfluss bei Kommunikationsausfall

Abbildung 33 veranschaulicht den notwendigen Nachrichtenfluss für die Wiederaufnahme des Messwertversands in Richtung SensorCloud-DB nach einem Kommunikationsverlust. Der XMPP-Client erkennt die Wiederaufnahme seiner Konnektivität zu dem XMPP-Server der SensorCloud (1) und versendet eine SensorData-Nachricht in Richtung lokales Regelwerk mit dem Sensorwert „offline=false“ (2). Über eine Regel des Regelwerks werden zwei ActuatorMessages erzeugt (3). Eine ist an den Hardware Abstraction Layer adressiert und triggert die Funktion des HALs zum Umadressieren der zu versendenden SensorData-Nachrichten. Die zweite ist an den lokalen Syncon Comloss Dienst adressiert, der selbst als virtueller Aktor an den HAL angeschlossen ist (3). Nach Ausführen der durch die ActuatorMessage aufgerufenen Funktion des HALs werden neue Messwerte als SensorData-Nachrichten an die SensorCloud adressiert (4). Nach Ausführen der durch die ActuatorMessage aufgerufenen Funktion des Syncon Comloss Dienst werden die zwischenzeitlich in der lokalen Datenbank gespeicherten Messwerte selektiert (5) und als SensorData-Nachricht an die SensorCloud versendet (6).

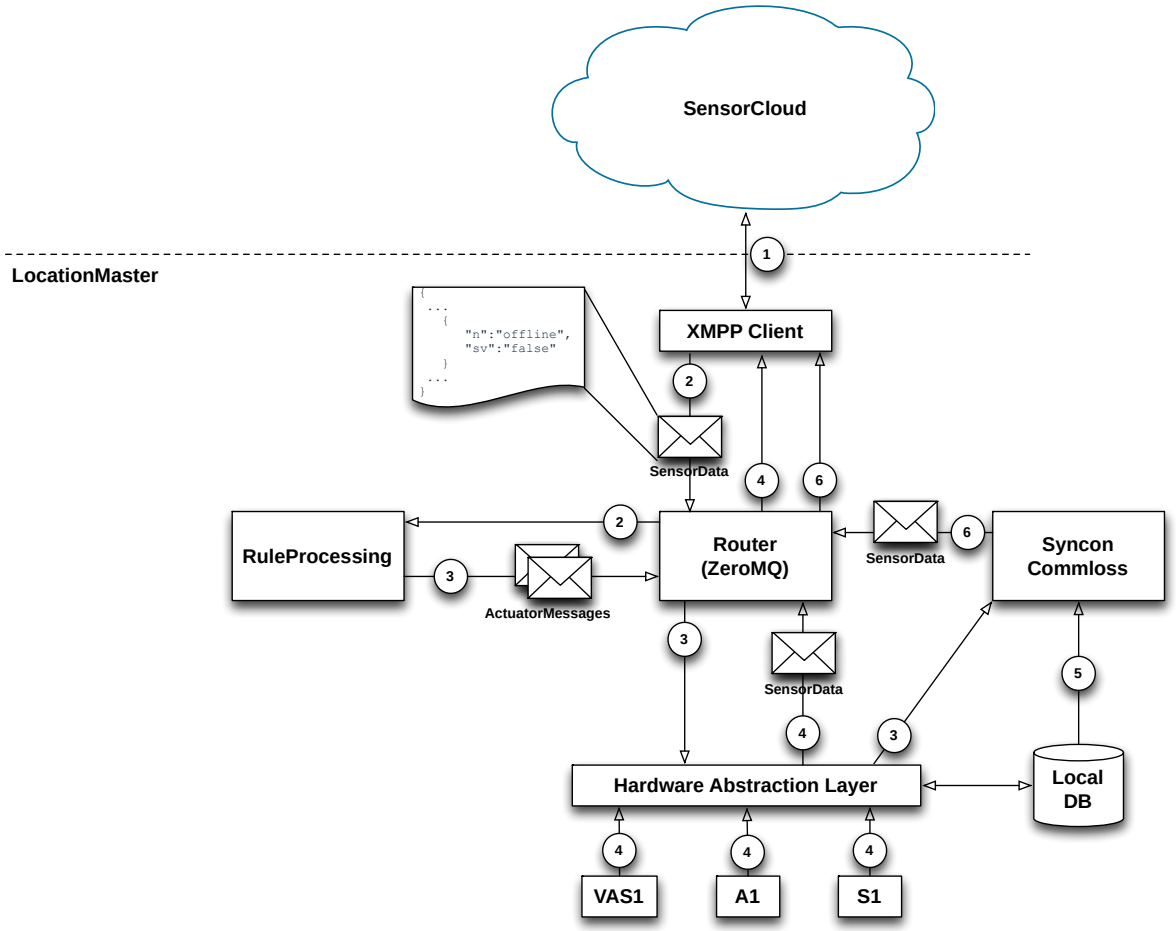

Abbildung 33: Nachrichtenfluss nach Verbindungsaufbau

2.6 Aktorsteuerung

Aktoren können auf dem LocationMaster physische und virtuelle Aktoren sein und werden über einen Hardware Abstraction Layer (HAL) an das Gesamtsystem angebunden [TBAP3]. Virtuelle Aktoren der Cloud werden über den Service Abstraction Layer (SAL) an das Gesamtsystem angebunden [DBAP12]. Für die Übermittlung der Steuerbefehle wird der Nachrichtentyp ActuatorMessage des SensorCloud-Protokolls genutzt. Die maschinelle Übersetzung des SensorCloud-Protokolls in z. B. proprietäre RPC-Funktionsaufrufe erfolgt über die Aktorsemantik innerhalb des HALs oder des SALs. Die Schnittstellen zu den Aktoren können RPC, Socket, proprietäre Verbindungen und weitere sein.

2.6.1 Regelinterpreter und Aktorsteuerung

Die für eine Aktorsteuerung verwendeten ActuatorMessages werden im Regelwerk erzeugt (Abbildung 34). Sensorwerte treffen über SensorData-Nachrichten des SensorCloud-Protokolls im Regelwerk ein, wo diese ausgewertet werden. Zutreffende Regeln erzeugen zur Steuerung von Aktoren ActuatorMessages. Regeln mit nur lokal angebundenen Sensoren und Aktoren laufen auf den jeweiligen LocationMastern im lokalen Regelwerk. Regeln mit Sensoren und Aktoren verschiedener LocationMastern laufen im Cloud-Regelwerk als globale Regeln. Hierbei werden alle in der Cloud eintreffenden Messwerte in das Cloud-Regelwerk geleitet. Trifft eine Regel zu, wird im Cloud-Regelwerk eine ActuatorMessage generiert, die über den Cloud-Dispatcher an den LocationMaster mit dem zu steuernden angeschlossenen Aktor weitergeleitet wird.

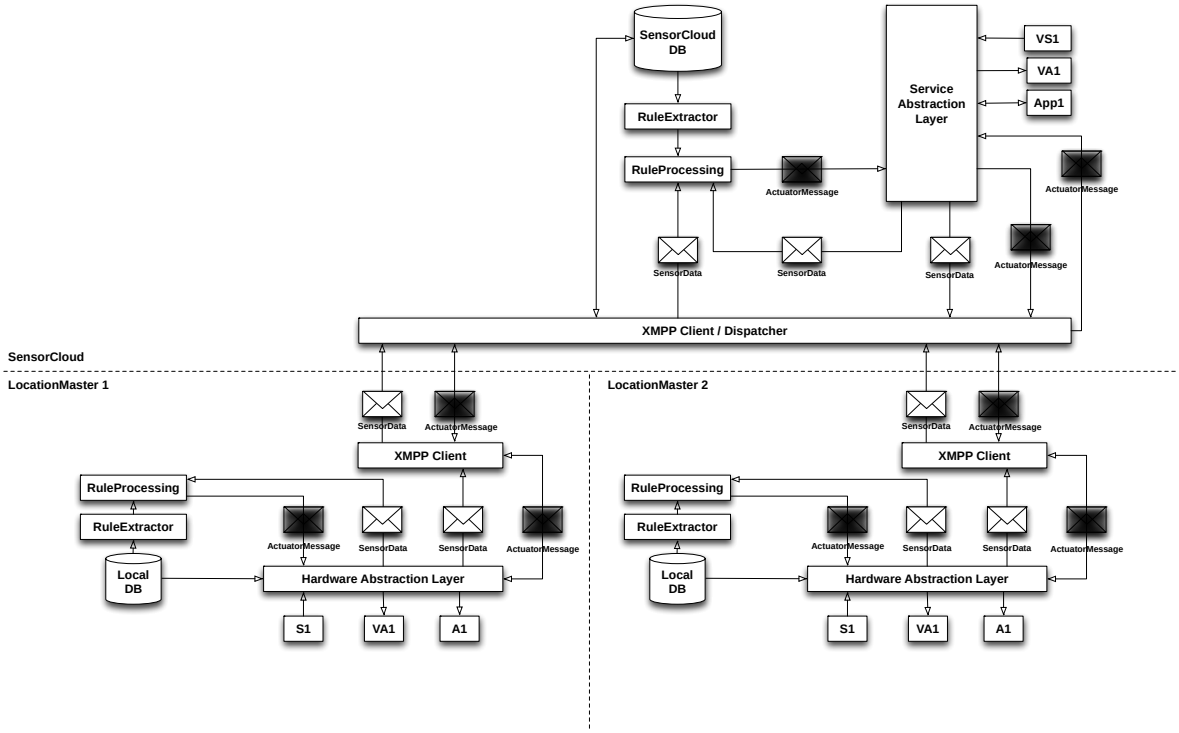

Abbildung 34: Für Aktorsteuerung notwendiger Nachrichtenfluss

Der LocationMaster empfängt die ActuatorMessage und reicht diese an den HAL weiter (vgl. Abbildung 35), der das Interface zu der angeschlossenen Hardware ist. Der HAL übergibt die eingehenden Nachrichten an Funktionsbausteine, die je nach Verbindungstyp die Kommunikation mit der Hardware realisieren.

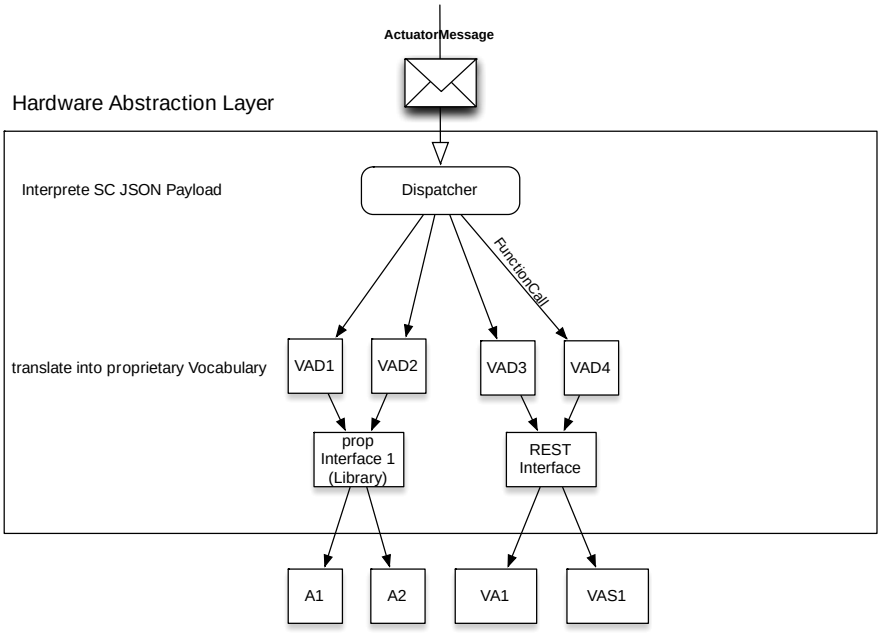

Abbildung 35: Hardware Abstraction Layer

Die Funktionsbausteine übersetzen den Funktionsnamen des kontrollierten Vokabulars identifiziert durch die URI im dst-Feld (vgl. 2.1.3) in eine herstellerspezifische Kommunikationsart und einen herstellerspezifischen Funktionsnamen und führen die Funktion des adressierten Aktors mit den Parametern aus dem Payload aus.

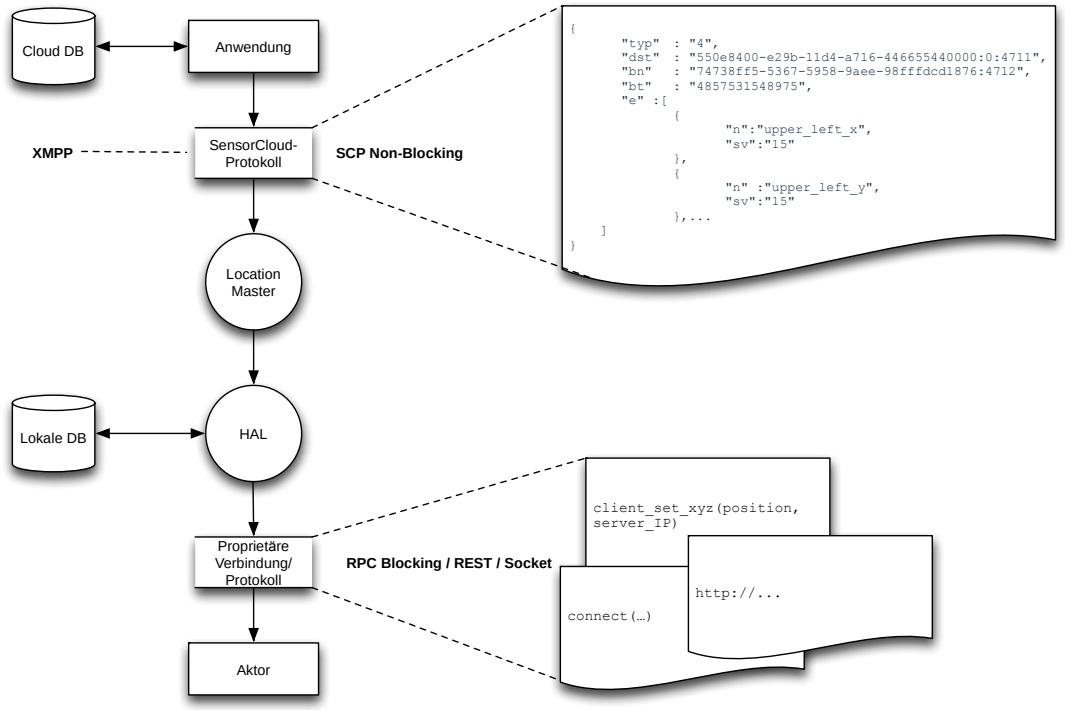

Abbildung 36: Datenflussdiagramm SensorCloud Richtung LocationMaster

Eine eventuell vorhandene Rückgabe der aufgerufenen Funktion des Aktors wird durch den HAL in eine SensorData-Nachricht verpackt und zurück an den Absender gesendet. Der Absender ist durch die bn-Angabe aus der eingegangenen ActuatorMessage adressierbar und beschreibt zum Beispiel eine Cloud-Anwendung (mit Sitzungsnummer), einen RESTful Service oder einen WebDAV-Server. Je nach Kommunikationsrichtung wird ein Datenfluss in Richtung LocationMaster (Abbildung 36) oder in Richtung SensorCloud (Abbildung 37) erzeugt.

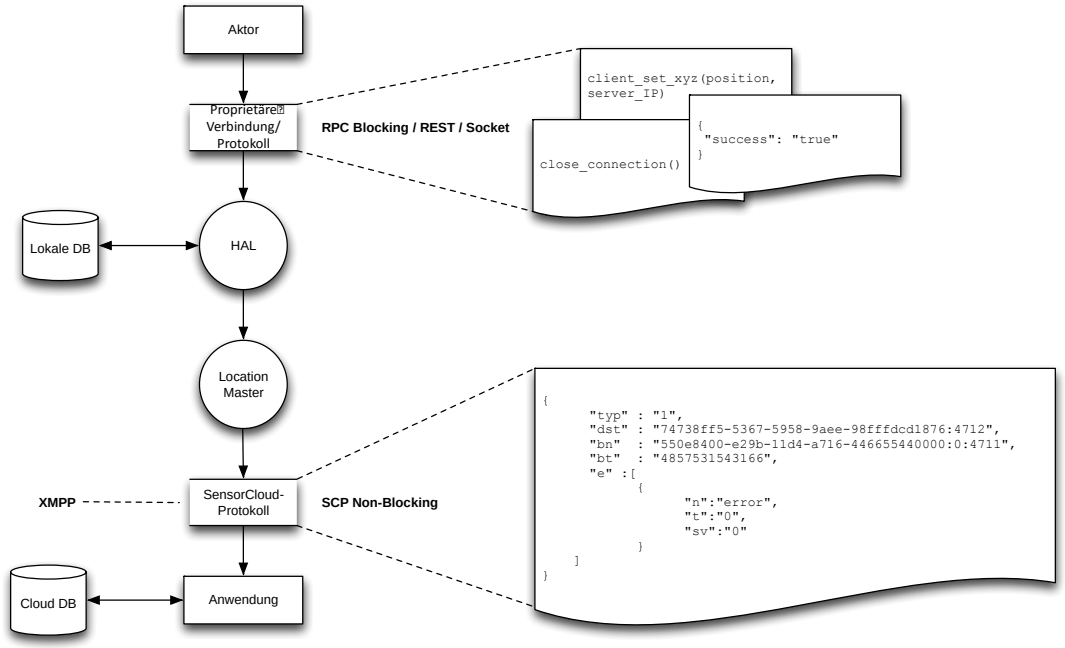

Abbildung 37: Datenflussdiagramm LocationMaster Richtung SensorCloud

2.6.2 Robot in SensorCloud

Die „Robot in SensorCloud“ Applikation [BA-JP] ist als erste Pilotanwendung zur Aktorsteuerung innerhalb der SensorCloud entwickelt worden. Sie besteht aus zwei Komponenten zusätzlich zur bestehenden SensorCloud Infrastruktur: eine Android Applikation und eine auf dem Gateway laufende Schnittstelle zu einem Lego Mindstorms Roboter mit Bluetooth Schnittstelle.

Die native Android Applikation wird als Eingabeschnittstelle für die Bewegung eines Roboters benutzt. Einem Roboter können so Fahrbefehle erteilt werden und diese auch an die SensorCloud weitergereicht werden.

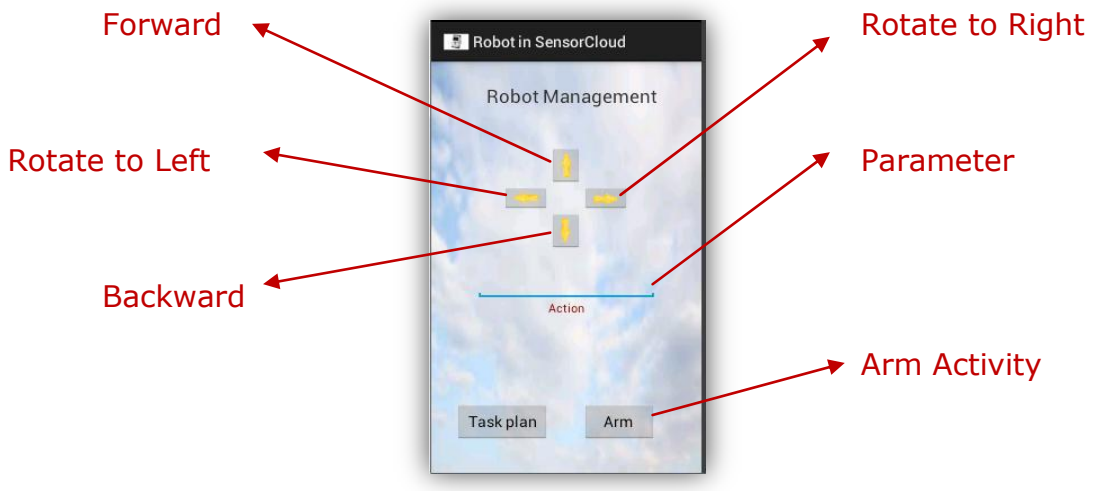

Abbildung 38: Android App "Robot in SensorCloud"

Die zweite Komponente dient als Treiber zwischen SensorCloud und dem Roboter Aktor. Als Aktor wird ein Lego Mindstorm NXT 2.0 Roboter verwendet.

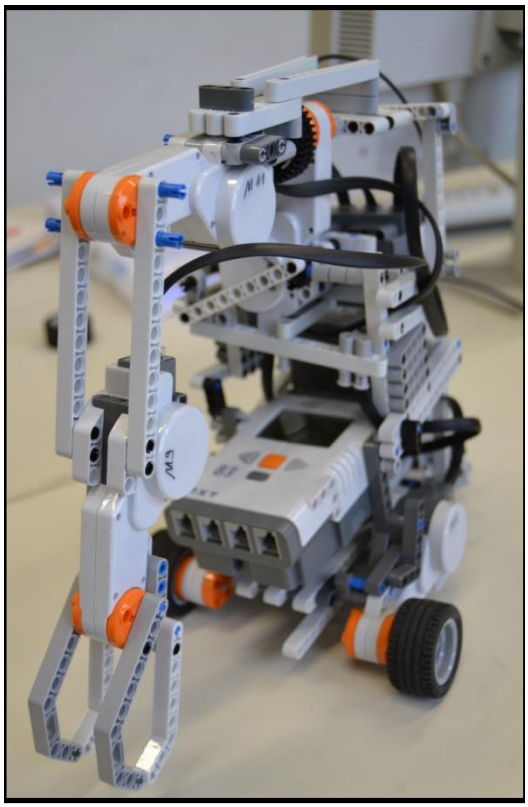

Abbildung 39: Lego Mindstorm NXT 2.0

2.7 Modellierung von Sensor- und Aktordiensten mittels einer DSL

Um die Erstellung von Apps und Diensten für die SensorCloud zu vereinfachen wird auf eine domänenspezifische Sprache zurück gegriffen, welche über die angebotenen Dienste des DAL (vgl. 2.2.1) auf das FDBS der SensorCloud zugreifen kann. Mit Hilfe der in einer Bachelorarbeit [BA-JK] erstellten DSL (**D**omain **S**pecific **L**anguage), kann Quellcode zur Unterstützung der Entwicklung von SensorCloud Applikationen erstellt werden.

Die entwickelten Dienste mittels der zuvor definierten DSL befassen sich in erster Linie mit Aktoren, die auf vorher definierten Regeln zu schalten sind. Beispielsweise kann ein Garagentor-Aktor auf Befehl hin sofort oder auf Wunsch mit Verzögerung geöffnet bzw. geschlossen werden. Zusätzlich lassen sich die Stammdaten eines Nutzers der SensorCloud mittels der modellierten Dienste verwalten und verändern.

Zur Erstellung der DSL wurde mit Hilfe von Xtext eine Grammatik definiert, die es erlaubt aus Terminalen und Nichtterminalen neue Regeln zur Erstellung von Services zu erstellen. Xtext ist ein Java Framework welches auf Eclipse aufsetzt und dort zusätzliche unterstützende Tools bietet, wie beispielsweise einen Content Assist für selbst erstellte Grammatiken.

Beispiel einer typischen Regel:

```
Regelname:
        ‚schluesselwort‘ eigenschaft = WeitereRegel
        ‚schluesselwort2‘ eigenschaft2 = TERMINALREGEL
;
```

Beispiel einer Regel zur Öffnung eines Garagentors mit Verzögerung:

```
AwGarage:
        'where doorID=' (garID = Uuid | garId = 'setLater')
        ('daytime open =' time0 = Time)?
        ('daytime close =' timeC = Time)?
        ('delay open =' delay0 = INT)?
        ('delay close =' timeC = INT)?
        (menu ='menu')?
;
```

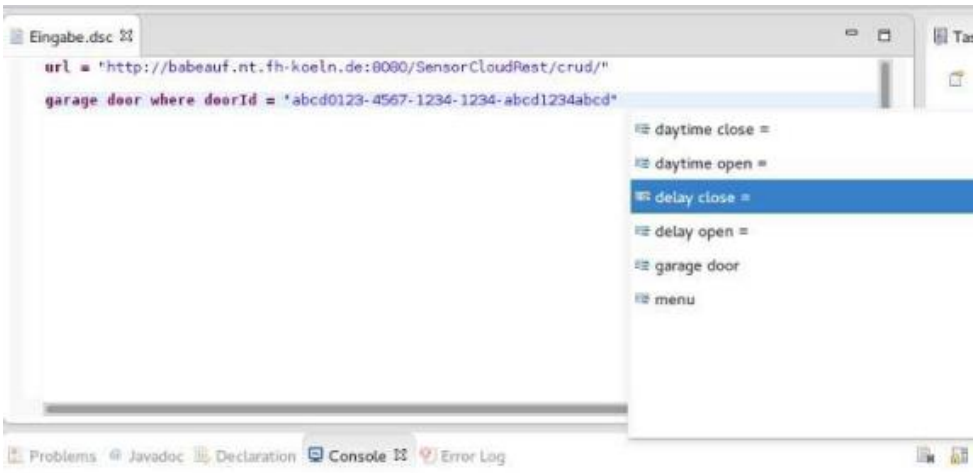

Abbildung 40: Editor mit Content-Assist

3. Integration Teststrecke

3.1 Gesamtübersicht Teststrecke

Nachfolgende Systemübersicht zeigt das Zusammenwirken aller im Projekt entwickelten Module des FDBS, die in den vorherigen Kapiteln beschrieben wurden. Weitere in dieser Übersicht auftretende Module (z.B. Regel- und Konfig-Extraktor, Regelinterpreter, Hardware Abstraction Layer, Software Abstraction Layer) gehören in ihrer Entwicklung zum Testbett-System und sind dort beschrieben

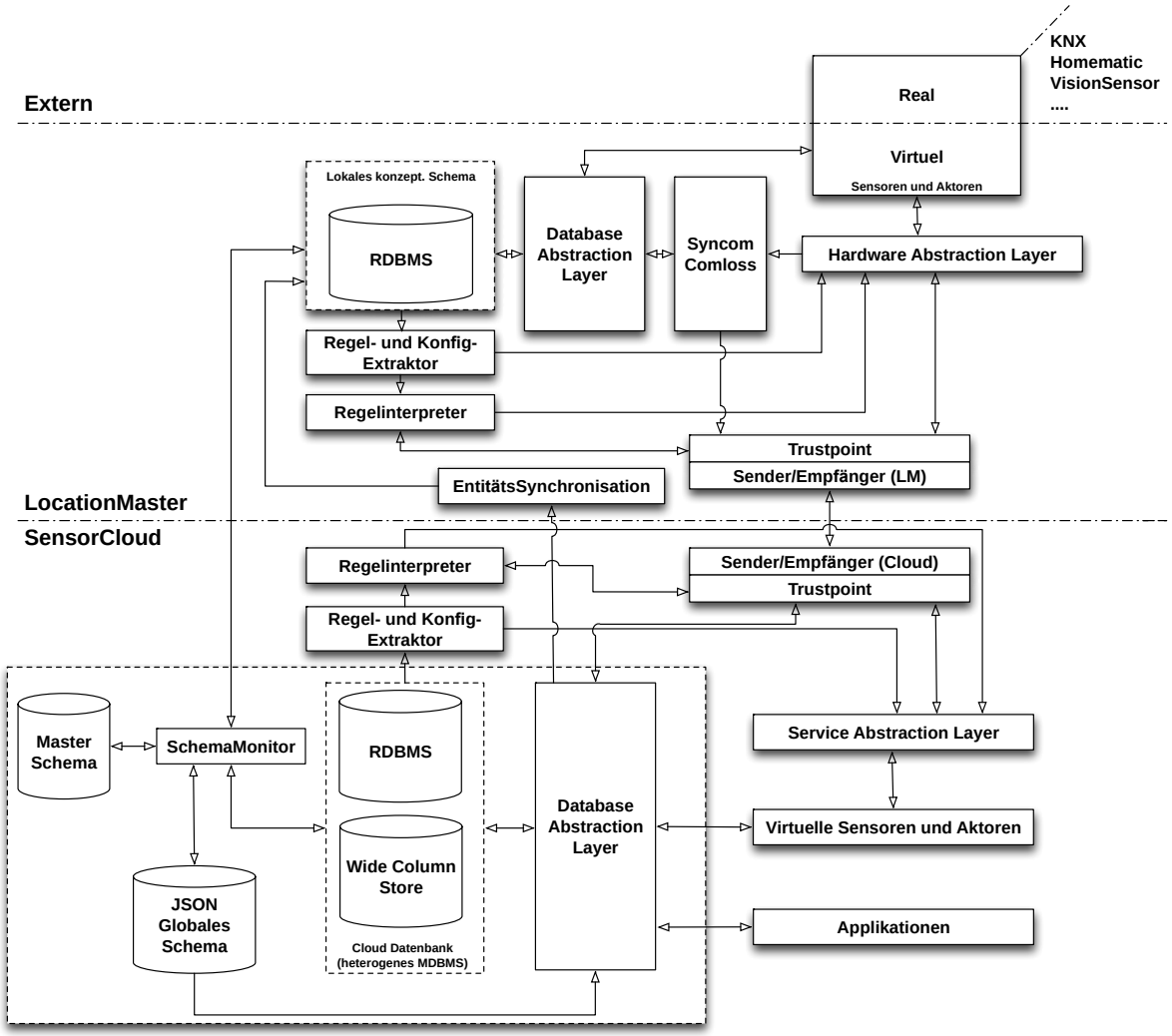

Abbildung 41: Gesamtübersicht Teststrecke

3.2 Integration Energiesensor

Ein speziell für die SensorCloud angepasster Energiesensor konnte mit Hilfe von [BA-MK] in die bestehende SensorCloud Umgebung samt Datenbank, Regelwerk und Kommunikationsinfrastruktur integriert werden. Dabei wurde beispielhaft demonstriert, welche Schritte ein Sensor (mit aktorischen Eigenschaften) durchlaufen muss, um an die SensorCloud angeschlossen werden zu können.

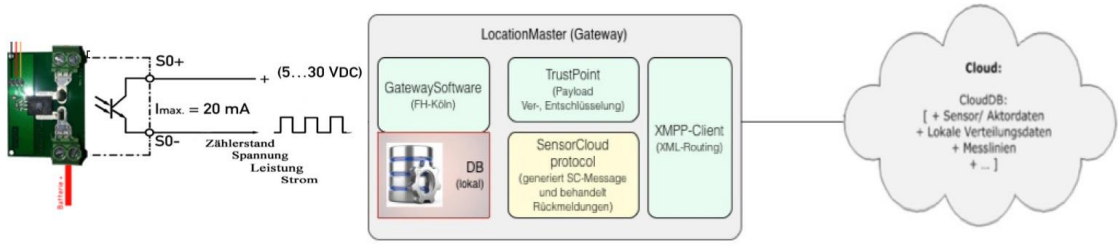

Abbildung 42: Aufbau und Integration des Energiesensors

Der Energiesensor wird wie jeder Sensor/Aktor über den HAL (**H**ardware **A**bstraction **L**ayer) an einen LocationMaster angebunden. Damit der HAL mit dem Energiesensor kommunizieren kann, musste ein Treiber speziell für den Energiesensor geschrieben werden. Der entwickelte Treiber kann in einem späterem Produktivszenario automatisiert heruntergeladen und dem HAL zur Verfügung gestellt werden.

In der folgenden Tabelle ist die nötige Beschreibung für einen Energiesensor zu sehen. Es werden sein Messwert-Tupelaufbau mit Einheit, Wertebereich und die SensorProduktSemantik beschrieben.

SensorProdukt	Messwerttupel	Messwerttupel (schematisch)	SensorProduktSemantik
EnergieErfassungsSensor	6873;5.6353285005173 54;true	**A** • Aufsummierte Energie eines Verbrauchers P • Einheit: Kilowattstunde (kWh) ; **B** • benötigte Leistung eines Verbrauchers P • Einheit: Watt (W) ; **C** • Zustand des Geräts an dem der Energiesensor angeschlossen ist • Gerät kann den Zustand AN oder AUS haben	**3**; **1**;Energie;energy;0,65535;int;min,max,avg;kWh;Aufsummierte Energie in Kilowattstunde; **2**;Leistung;power;0,65535;int;min,max,avg;W;momentan verspeiste Wirkleistung in Watt; **3**;Zustand;state;false,true;boolean;min,max,avg;momentan verspeiste Wirkleistung in Watt;

Beispiel einer SensorProdukSemantik für einen Energiesensor in JSON-Format:

```
{
    "n": 3,
    "parvor": [
        {
            "nr": 1,
```

```
        "pns": "energy",
        "pn": "Energie",
        "w": {
            "min": 0,
            "max": 65535
        },
        "dt": "int",
        "eh": "kWh",
        "aggfkt": [
            "min",
            "max",
            "avg"
        ],
        "erl": "Aufsummierte Energie in Kilowattstunde"
    },
    {
        "nr": 2,
        "pns": "power",
        "pn": "Leistung",
        "w": {
            "min": 0,
            "max": 65535
        },
        "dt": "int",
        "eh": "W",
        "aggfkt": [
            "max",
            "min",
            "avg"
        ],
        "erl": "Momentan verspeiste Wirkleistung in Watt"
    },
    {
        "nr": 3,
        "pns": "state",
        "pn": "Zustand",
        "w": {
            "enum": [
                "false",
                "true"
            ]
        },
        "dt": "boolean",
        "sem": {
            "false": "Aus",
            "true": "An"
        },
        "aggfkt": [
            "count",
            "min",
            "max",
            "avg"
        ]
```

```
            }
        ]
}
```

Am Beispiel des Energiesensors lässt sich auch analog zu einem Sensor die Integration eines Aktors zeigen, da der Energiesensor über eine Reset-Funktionalität und eine Schaltsteckdosen-Funktionalität verfügt, die zusammen als eigenständiger Aktor in die SensorCloud integriert werden.

Jedes Aktorprodukt bietet zur Steuerung der physikalischen Aktivitäten eines Aktors eine unterschiedliche Menge an Funktionen. Dabei können die Übergabeparameter jeder Funktion verschieden sein. Um eine geeignete AktorProduktSemantik für Aktorprodukte zu definieren, muss jede Funktion folgende Eigenschaften berücksichtigen (vgl. [DBAP3]):

Eigenschaft	Beschreibung	Funktion	Beispiel
fnr	Nummer der Funktion	Dient für einen direkten Zugriff auf die Semantik der n-ten Funktion in einer Parsingvorschrift.	"fnr":1
fnam	Name der Funktion	Ist eine Zeichenkette und beschreibt den Funktionsnamen (in URI-Schreibweise z.B.: urn:rm_rpc_lib:get_last_pic).	"fnam":"hoch"
fart	Funktionsart	Ist eine Zeichenkette und beschreibt eine im **kontrollierten Vokabular (VokFArt[12])** erläuterte Funktionsart, welche AktorTyp bezogen ist.	"fart":"up"
fpanz	Anzahl der Parameter	Ist eine natürliche Zahl und beschreibt die Anzahl der Parameter.	"fpanz":1
pnr	Nummer des Parameters	Ist eine Zahl und beschreibt die Parameternummer.	"pnr":1
pnam	Parametername	Ist eine Zeichenkette und beschreibt den Parameternamen.	"pnam":"distanz"
part	Parameterart	Ist eine Zeichenkette und beschreibt eine im **kontrollierten Vokabular (VokPArt[13]),** erläuterte Parameterart.	"part":"distance"
pw	Wertebereich	Beschreibt den Wertebereich des übergebenen Parameters und hat den Aufbau WI: W1-W2 (Intervall, Beispiel: 0-50), WA: W1,W2, ... WN (Aufzählung, Beispiel na,a0,a1) oder WW.: W0, W1 (Wahrheitswert, Beispiel w0,w1).	"pw":{"min":1,"max":100}
peh	Physikalische Einheit	Ist eine Zeichenkette aus einem **kontrolliertem Vokabular (VokEH)** und beschreibt die physikalische Einheit des Übergebenen Parameters.	"peh":"W"
pdt	Datentyp	Ist ein Datentyp aus einem **kontrolliertem Vokabular (VokDT)** für Datentypen und beschreibt den Datentyp des Übergebenen	"pdt":"int"

[12] Vokabular Funktionsart
[13] Vokabular Parameterart

		Parameters.	
perlz	Erläuterungen für pw	Ist eine Erläuterung für den Wertebereich des Parameters pw und ist eine Zeichenkette W1-perlz(pw), W2-perlz(pw), ... WN-perlz(pw) .	"perlz":"{"Beschreibt den minimalen Wert","Beschreibt den maximalen Wert"}
franz	Anzahl der Rückgabewerte	Ist eine natürliche Zahl und beschreibt die Anzahl der Rückgabewerte.	"franz":1
rnr	Nummer des Rückgabewerts	Ist eine Zahl und beschreibt die Rückgabewertnummer.	"rnr":1
rnam	Name des Rückgabewerts	Ist eine Zeichenkette und beschreibt den Rückgabewertnamen.	"pnam":"status"
rart	Art des Rückgabewerts	Ist eine Zeichenkette und beschreibt eine im **kontrollierten Vokabular,** erläuterte Rückgabewertart.	"rart":"status"
rw	Wertebereich Rückgabewert	Beschreibt den Wertebereich des Rückgabewerts der Funktion. Rw hat den Aufbau WI: W1-W2 (Intervall, Beispiel: 0-50), WA: W1,W2, ... WN (Aufzählung, Beispiel na,a0,a1) oder WW.: W0, W1 (Wahrheitswert, Beispiel w0,w1) .	"rw":"enum" [0,1]
rdt	Datentyp Rückgabewert	Ist ein Datentyp aus dem **kontrollierten Vokabular (VokDT)** und beschreibt den Datentyp des Rückgabewerts.	"rdt":"boolean"
reh	Physikalische Einheit des Rückgabewerts	Ist eine Zeichenkette aus dem **kontrollierten Vokabular (VokEH)** und beschreibt die Einheit des Rückgabewerts.	"reh":["W"]
rerlz	Erläuterungen für rw	Ist eine Zeichenkette W1-rerlz(rw), W2-rerlz(rw), ... WN-rerlz(rw) und ist eine Erläuterung für den Wertebereich des Rückgabewerts.	"rerlz":{"0":"nicht vollständig runtergefahren", "1":" vollständig runtergefahren"}
ferl	Erläuterung der Funktion	Ist eine Zeichenkette, die diese Komponente als Fließtext beschreibt.	"ferl": "Funktion 1 lässt den Aktor die Rolllade hochfahren. Der Übergabeparameter bestimmt die Höhe in Prozent."

AktorProduktSemantik (für einen Energiesensor):

```
{
    "n": "2",
    "parvor": [
        {
            "fnr": "1",
            "fname": "zuruecksetzen",
            "fart": "reset",
            "param": [
                {
                    "pnr": "1",
                    "pnam": "setback",
                    "pw": {
                        "enum": [
                            "false",
                            "true"
                        ]
                    },
                    "pdt": "boolean",
                    "perlz": {
                        "enum": [
                             "Beschreibt den Wert der den Energiesensor
                             in die Lage versetzt sich zu reseten",
                             "Beschreibt den Gegenwert, bei diesem Wert
                             passiert nichts!"
                        ]
                    }
                }
            ],
            "rdt": "boolean",
                 "ferl": "Funktion 1 lässt den Aktor den Energiesensor
                 zurücksetzen!"
        },
        {
            "fnr": "2",
            "fname": "schalteGeraet",
            "fart": "switchDevice",
            "param": [
                {
                    "pnr": "1",
                    "pnam": "switch",
                    "pw": {
                        "enum": [
                            "on",
                            "off"
                        ]
                    },
                    "pdt": "boolean",
                    "perlz": {
                        "enum": [
                             "Beschreibt den Wert der
```

```
                                    Schaltsteckdosenfunktion die im
                                    Energiesensor eingebaut ist, bei diesem
                                    Wert wird das Gerät eingeschaltet an dem
                                    der Energiesensor angeschlossen ist",
                                    "Beschreibt den Gegenwert, bei diesem Wert
                                    geht das Gerät an!"
                                ]
                            }
                        }
                    ],
                    "rdt": "boolean",
                        "ferl": "Funktion 2 lässt den Aktor das Gerät an dem
                        der Energiesensor angebracht ist, ein-, bzw.
                        ausschalten!"
                }
            ]
}
```

Zur Integration des Energiesensors müssen nachfolgende Tabellen mit speziell für den Energiesensor stehenden Informationen gefüllt werden:

- Sensor und Aktor und die damit verbundenen Sensor-/Aktortypen und deren ProduktSemantiken
- ein NetzwerkManager
- ein LocationMaster, an dem der Energiesensor angeschlossen ist
- ein Nutzer, dem der Energiesensor gehört
- eine Lokation an dem sich der Energiesensor befindet

Soll der Energiesensor (z.B. die Schaltsteckdose des Energiesensors) über das SensorCloud Regelwerk gesteuert werden, müssen auch entsprechende Regeln in die Event Entitäten der SensorCloud abgelegt werden.

3.3 Integration TrustPoint

Im Projekt SensorCloud spielen die Sicherheit und der Datenschutz eine zentrale Rolle. Aus diesem Grund wurde eine Trust-Point-Sicherheitsarchitektur (vgl. [DBAP9]) geschaffen. Der Trustpoint (Abbildung 43) ist ein Modul auf einem LocationMaster, das zur Verschlüsselung von Daten und zur Erzeugung von Schlüsseln vorhanden sein muss. Mit dem Einsatz des Trust Points soll Vertrauen in die SensorCloud geschaffen und die Akzeptanz des Systems gesteigert werden.

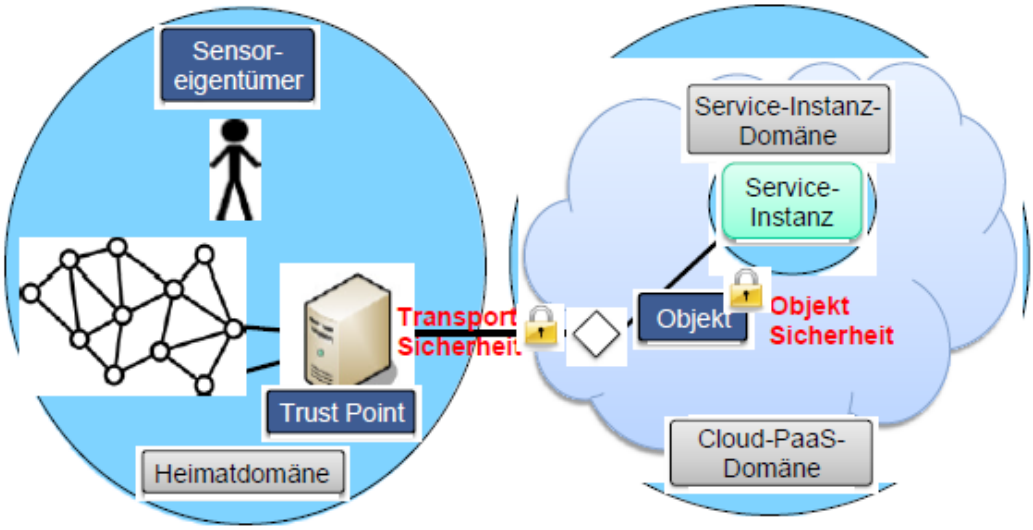

Abbildung 43: TrustPoint - Architektur

Um den TrustPoint auf die Prototyp Testrecke der SensorCloud zu integrieren sind nur wenige Schritte notwendig. Die SensorCloud Infrastruktur ist dabei so geschaffen, das man das Modul TrustPoint einfach als Komponente in den Nachrichtenverkehr zwischen LocationMaster und Cloud einhängen kann und somit zusätzlich zur Transportsicherheit der Daten auch eine Sicherheit der Datenintegrität und Datenautorität erreichen kann.

Der Trust Point läuft als eigenständige Software, die über Unix-Pipes kommuniziert bzw. gesteuert wird. In eine Pipe "data.input" werden zeilenweise Payload-Objekte nach der SensorCloud Protokoll Spezifikation (vgl. 2.1.3) geschrieben. Im TrustPoint wird das unverschlüsselte SensorCloud Protokoll Paket mit eigens dafür generierten Schlüsseln verschlüsselt und das so verschlüsselte Objekt wieder in eine Pipe "data.output" gesendet. Über eine dritte Pipe "DataProcessor.command_fifo" kann der Trust Point gesteuert werden, d.h. über diese Pipe ist derzeit das Starten und Stoppen des Trust Points möglich.

Im nachfolgenden Beispiel (Abbildung 44) wird der Temperaturwert im Feld „sv“ (Spalte 1) verschlüsselt. Der verschlüsselte Wert wird im neuen Payload „ev“ (encrypted value) im Feld „ciphertext“ übertragen (Spalte 2).

```
{
        "typ":"1",
        "gw":"string",
        "bn":"string",
        "bt":"number",
        "e":[
        {

        "n":"Temperatur",
               "sv":"ab1"









}
        ]









}
```

```
{
        "typ":"1",
        "gw":"string",
        "bn":"string",
        "bt":"number",
        "e":[
        {
                "n":"Temperatur",
                "ev":[
                {
                        "unprotected":
                        {
                                "alg":"dir",
                                "enc":"AESGCM256",
            "kid":"851acaf912c08
                        3843549508e7d411fb73a7a8c5b",
                                "typ":"sv"
                        },
                        "iv":" AQABAAEAAAAAAAAAAAAAAA==",
                        "ciphertext":" AvFvx6kAPd1V",
                        "tag":" NpOsZHe8Ys+RFSQ8vbTD5g=="
                }
                ]
        }

        ],
 "sig":
 {
        "signatures":[
        {
                "header":
                {
                        "alg":"ES256"
                },
                        "signature":"
MC0CFG2YmIu64ltOuPH2otrO1e7

/xOHsAhUAyAQbkP6JdzIfE/wsF8WKCn936eQ="
        }
        ]
 }
}
```

Abbildung 44: Beispiel der unverschlüsselten Übertragung eines Temperaturwerts im SensorCloud-Protokoll (Spalte 1) und seiner verschlüsselten Übertragung (Spalte 2).

Die Schlüssel, die zur Verschlüsselung generiert wurden, werden vom Trustpoint in einer zum TrustPoint gehörenden Datenbank abgelegt.

Zur Entschlüsselung von Daten kann eine speziell dafür entwickelte Bibliothek genutzt werden. Diese Bibliothek kann über eine Callback Funktion mit entsprechender Übergabe der Parameter der zur entschlüsselnden Nachricht aufgerufen werden umso an den Schlüssel zu kommen, der die in der SensorCloud verschlüsselten Nachrichten wieder entschlüsseln kann.

Derzeit wird das Modul zur Entschlüsselung noch in sämtlichen Applikationen verlangt, dies kann aber in einem weiteren Arbeitsschritt durch einen (Web)Service ersetzt werden, der die entsprechen Parameter entgegennimmt und den Schlüssel als Rückgabewert liefert. So können Anwendungen entschlackt werden und Anwendungsentwickler müssen sich bis auf Service Aufruf und Rückgabewert nicht mit dem Schlüsselmanagement in ihren Applikationen beschäftigen.

4. Ergebnisse und Fazit

Das Teilvorhaben der FDBS Gruppe der FH Köln im Projekt SensorCloud wurde in allen Bereichen erfolgreich abgeschlossen. Alle zu leistenden Arbeitspakete sind erfolgreich durchgeführt worden und können wie in der Teilvorhabensbeschreibung der FH Köln schon jetzt, aber auch in der Zukunft verwertet werden.

Das entwickelte föderierte Datenbanksystem ist als sehr robustes und hoch skalierendes Cloud System entwickelt worden, das sowohl mit einer großen Menge an lokalen Datenbanken auf LocationMastern als auch mit verschiedensten in der Cloud gehosteten Datenbanksystemen operieren kann. Die einzelnen Cloud Knoten müssen dabei selbst keine Hochleistungsrechner sein. Bei schwächerer Ausstattung eines einzelnen Knotens kann die Anzahl der Gesamtknoten linear erhöht werden. Die für die protypische Entwicklung ausgewählte Cassandra Datenbank unterstützt dabei das Konzept der horizontal skalierbaren Cloud Anwendung, so dass auch hier Knoten unmittelbar hinzu oder auch entfernt werden können.

Das Konzeptionelle Schema (vgl. 2.1.1, vgl. [DBAB4]), welches auf Basis der Anforderungsanalyse (vgl. [DBAP1]) entstanden ist, führt zu derzeit 69 Entitäten. Innerhalb dieser Entitäten werden zusätzliche ontologische Informationen (vgl. 2.1.2, vgl. [DBAP3]) hinterlegt, die für verschiedene Anwendungsfälle angereichert werden können. Für den Anwendungsfall der Hausautomatisierung wurden die Entitäten sehr detailreich beschrieben. Eine Erweiterung für andere Anwendungsfälle zur Steuerung von Energienetzen oder von Industrieanlagen ist vorgesehen und kann im Zuge von zukünftigen Arbeiten geschehen.

Zur Überwachung der verschiedenen Schemata (Cloud und LocationMaster) sind verschiedene SchemaMonitor Dienste (vgl. 2.2.3, vgl. [DBAP5]) entwickelt worden, die für eine durchgängige Konsistenz des SensorCloud Datenmodells sorgt.

Für die LocationMaster wurden innerhalb des Arbeitspakets Deploymentanalyse (vgl. 2.5.1, vgl. [DBAP2]) verschiedenste Datenbankmanagement Systeme untersucht und PostgreSQL als Empfehlung zur Verwendung auf LocationMastern ausgegeben. Die LocationMaster besitzen nun eine Teilmenge an Entitäten des Konzeptionellen Schemas und können darüber individuell konfiguriert werden. Individuelle Konfigurationen sind von wesentlicher Bedeutung, wenn es um die Erstellung und Ausführung von auszuführenden Regeln geht. Regeln können in der Datenbank des LocationMasters persistent gesichert werden und über den Regel- und Konfigextraktor für das Regelwerk der SensorCloud aufbereitet werden.

Zur Synchronisierung von angefallenen Daten im Kommunikationsausfall wurde im Arbeitspaket Syncom Commloss [DBAP7] eine über das Regelwerk definierte Schrittfolge definiert, die einen Kommunikationsausfall überbrückt und dabei sicherstellt, dass keine Daten verloren gehen.

Der Trustpoint auf einem LocationMaster sichert, zusätzlich zur Transportsicherheit über SSL, Messwerte gegenüber Dritten ab und stellt Schnittstellen auf autorisierte Benutzer zur Entschlüsselung bereit. Der Trustpoint wurde erfolgreich in Verbindung mit dem föderiertem Datenbanksystem der SensorCloud getestet und integriert [DBAP9].

Last- und Stresstests [DBAP11] wurden erfolgreich durchgeführt und zeigten, dass die föderierte Datenbank erfolgreich in die SensorCloud integriert werden konnte. Lokale und Cloud Datenbanken sind erfolgreich auf alle ihre Funktionen getestet worden. Automatisierte Test konnten erfolgreich durchgeführt werden und zeigten keine Fehler.

Die Ergebnisse der Arbeitspakete der FDBS Gruppe der Fachhochschule Köln können als wissenschaftliche wertvolle Erkenntnisse in zukünftige Projekte eingebracht werden und können dadurch als komplexe Basis Technologie für diverse Cloud Anwendungen funktionieren. In Planung sind Projekte innerhalb der Energiebranche, so dass Klein-Kraftwerke (Photovoltaik Anlagen, Windkraftwerke) über die Cloud gesteuert werden können. Eine Erweiterung der SensorCloud um Komponenten für BigData Analysen sind leicht in die FDBS-Umgebung einzubetten. Damit können Berechnungen und Vorhersagen für virtuelle Kraftwerke der regenerativen Energieerzeugung durchgeführt werden. Dieses sind Arbeitschritte, die in Nachfolgeprojekten des Projekts SensorCloud durchgeführt werden können.

Zusätzlich werden die Ergebnisse des Projekts SensorCloud in die Lehre der Fachhochschule eingehen und Themen für viele weitere Abschlussarbeiten liefern, sowie das auch schon über die Projektlaufzeit hin

durchgehend geschehen ist.

Quellen

Hinweis: **Quellen, die auf das BSCW der Fachhochschule Köln verweisen können nur auf Anfrage zugegriffen werden**

[BA-AS] A. Schneider: „*Webbasierte Zugriffe mit mobilen Komponenten auf Daten der SensorCloud*“, Januar 2014, http://bscw.fh-koeln.de/bscw/bscw.cgi/d3951049/Arthur_Schneider_BachelorArbeit.pdf

[BA-BS] B. Schmitz: „*Entwicklung eines Datenloggers für die Überwachung einer Photovoltaikanlage als Komponente einer SensorCloud*“, Bachelor Arbeit, 2013, http://bscw.fh-koeln.de/bscw/bscw.cgi/3942945

[BA-JK] J. Köhn: „*Modellierung einer auf SensorCloud bezogenen DSL mit Xtext zur Generierung von Java-Anwendungsprogrammen mit Webservice-Zugriff auf ein FDBS*“, Bachelor Arbeit, 2014, http://bscw.fh-koeln.de/bscw/bscw.cgi/d4076928/BA%20K%c3%b6hn.pdf

[BA-JP] J. Rodriguez Puigvert: „*Robot in SensorCloud*“, Final Year Project, 2013, http://www.nt.fh-koeln.de/fachgebiete/inf/vma/index_31.html

[BA-MK] M. Kucin: “*Integration eines Energiesensors in die SensorCloud*”, Bachelor Arbeit, 2014 http://bscw.fh-koeln.de/bscw/bscw.cgi/d4073338/Bachelorarbeit_MaykilKucin.pdf

[BA-SW] S. Wieben: “*Wetterberichte für die SensorCloud*”, Bachelor Arbeit, 2014 http://bscw.fh-koeln.de/bscw/bscw.cgi/d3951089/BachelorThesis.pdf

[BRECAP] Brewer, E. A. (2012). CAP twelve years later: How the "rules" have changed. (I. C. Society, Hrsg.) *Computer , 45* (2), 23-29.

[COPCA] H. Budde: „*CopyCass*“, Software, 2014 http://bscw.fh-koeln.de/bscw/bscw.cgi/d4072510/Copy2Cass2CQL3.rar.zip

[DBAP2] A. Lockermann et al.: „*Deploymentanalyse. DBAP2*“, Fachhochschule Köln, Gruppe FDBS, Köln, 2013, http://bscw.fh-koeln.de/bscw/bscw.cgi/d3618380/%5bA2%5d%20Deploymentanalyse,%20Ergebnisse%20zu%20%5bDBAP2%5d.pdf

[DBAP3] H. Budde et al.: „*Semantische Analyse mittels Ontologien*”, Fachhochschule Köln, Gruppe FDBS, Köln, 2014,http://bscw.fh-koeln.de/bscw/bscw.cgi/d3944510/DB3%20Semantische%20Analyse%20mittels%20Ontologien.pdf

[DBAP4] H. Budde et al.: „*Konzeptionelles Schema. DBAP4*“, Version 2, Fachhochschule Köln, Gruppe FDBS, Köln, 2014 http://bscw.fh-koeln.de/bscw/bscw.cgi/d3862223/Konzeptionelles%20Schema.docx

[DBAP5] H.Budde: „*Schema Monitor*“, Version 2, Fachhochschule Köln, Gruppe FDBS, Köln, 2014

http://bscw.fh-koeln.de/bscw/bscw.cgi/d4085623/DBAP5%20Schema%20Monitor.pdf

[DBAP6] A. Stec et al.: „*Übertragen von Sensordaten unter Berücksichtigung von Transaktionskonzepten in der SensorCloud. DBAP6*“, Fachhochschule Köln, Gruppe FDBS, Köln, 2013, http://bscw.fh-koeln.de/bscw/bscw.cgi/d3927152/%c3%9cbertragung%20von%20Sensordaten%20unter%20Ber%c3%bccksichtigung%20von%20Transaktionskonzepten%20in%20der%20SensorCloud,%20Zwischenergebnisse%20%20%5bDBAP6%5d.pdf

[DBAP7] T. Partsch et al.: „*Syncom Comloss - Zwischenspeicherung von Sensordaten bei fehlender Internetverbindung zur SensorCloud*“, Fachhochschule Köln, Köln, 2014, http://bscw.fh-koeln.de/bscw/bscw.cgi/d3923032/Syncon%20Commloss%20-%20Zwischenspeicherung%20von%20Sensordaten%20bei%20fehlender%20Internetverbindung%20zur%20SensorCloud,%20Ergebnisse%20%5bDBAP7%5d.pdf

[DBAP8] T. Partsch et al.: „*DB Dienste zur Unterstützung von Sensordiensten*”, Fachhochschule Köln, Köln, 2014, http://bscw.fh-koeln.de/bscw/bscw.cgi/d3923055/DB8%20Dienste%20zur%20Unterst%c3%bctzung%20von%20Sensordiensten%20Zwischenbericht.pdf

[DBAP9] T. Partsch et al.: „*Dienste zur Unterstützung des LocationMasters als Trustpoint“,* Fachhochschule Köln, Köln, 2014, http://bscw.fh-koeln.de/bscw/bscw.cgi/d4091025/DB9%20DB%20Dienste%20zur%20Unterst%c3%bctzung%20des%20LocationMaster%20als%20Trustpoint.pdf

[DBAP11] T. Partsch et al.: „*Lasttest*“, Fachhochschule Köln, Köln, 2014
http://bscw.fh-koeln.de/bscw/bscw.cgi/d4101344/DB11%20DB%20Lasttest.pdf

[DBAP12] T. Partsch: „*Systemintegration*“, Fachhochschule Köln, Köln, 2014
http://bscw.fh-koeln.de/bscw/bscw.cgi/d4100863/DB12%20Systemintegration.pdf

[DIER2008] T. Dierks, E. Rescorla: “*The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246*“**,** IETF, 2008

[FIE2000] R. Fielding: “*Architectural Styles and the Design of Network-based Software Architectures”*. Doctoral dissertation, University of California, Irvine, 2000.

[Häuß2011] R. Häußling, B. Rumpe, K. Wehrle: „*SensorCloud Teilvorhabenbeschreibung RWTH Aachen University“*, Förderantrag der RWTH Aachen University zur Einreichung beim Bundesministerium für Wirtschaft und Technologie im Rahmen des Wettbewerbes TrustedCloud, RWTH Aachen University, Aachen, 2011

[HUM2012] Hummen, R., Henze, M., Catrein, D., & Wehrle, K.: “A Cloud Design for User-controlled Storage and Processing of Sensor Data” In IEEE (Hrsg.), *Proceedings of the 2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom), Taipei, Taiwan*, (S. 232-240).

[JSCHECRE] H. Budde: „*JSON Schema Creator*“, Software, 2013
http://bscw.fh-koeln.de/bscw/bscw.cgi/d4072536/JSONSchemaCreator.rar.zip

[KLE2014] Klettke, M., Scherzinger, St., Störl, U.: „*Datenbanken ohne Schema*“, Datenbank Spektrum Band 14 Heft 2 Juli 2014, Springer

[MA-AS] A. Stec: „*Transaktionen in einem föderierten Datenbanksystem der SensorCloud*“, Master Thesis, Fachhochschule Köln, Köln, 2013, http://bscw.fh-koeln.de/bscw/bscw.cgi/d3700427/Master-Thesis%20-%20A.%20Stec.pdf

[MA-TE] M. Teske, „Spezifikation einer Robotic Task Definition Language (RTDL)“, Master Thesis, Fachhoschule Köln, 2011 http://opus.bibl.fh-koeln.de/volltexte/2011/311/pdf/MA_Spezifikation_RTDL.pdf

[MA-TP] T. Partsch: „*SensorCloud Datenmodellierung und Performanceüberlegungen mit einem strukturierten Datenspeicher als Zielsystem*“, Master Thesis, Fachhochschule Köln, 2014 http://bscw.fh-koeln.de/bscw/bscw.cgi/4036247

[**PRAX-AK**] A. Klein: „*Entwicklung eines Schema Monitors im Rahmen des SensorCloud Projekts*“, Praxissemester, Fachhochschule Köln, November 2013, http://bscw.fh-koeln.de/bscw/bscw.cgi/3846969

[PRAX-OE] M. Oetz: „*Generierung einer Ontologie zur Beschreibung der SensorCloud, insbesondere von Sensoren und Aktoren*“, Praxissemester Fachhochschule Köln, Mai 2014, http://bscw.fh-koeln.de/bscw/bscw.cgi/3846941

[PRAX-TP] T. Partsch: „*Invertiertes Dateisystem für Cloud-Systeme“*, Praxissemesterbericht, Fachhochschule Köln, Köln, 2013,http://bscw.fh-koeln.de/bscw/bscw.cgi/d3646405/Invertiertes%20Dateisystem%20f%c3%bcr%20Cloud-Systeme%20Dokumentation.pdf

[REGU] H. Budde: „*RegelGUI*“, Software, 2013, http://bscw.fh-koeln.de/bscw/bscw.cgi/d4072554/RegelGUI.rar.zip

[SCEDITEV] T. Partsch: „*SCEDIT für Regeln*“, http://babeauf.nt.fh-koeln.de/scedit/index.php?entity=Event

[SCPROT] R. Hummen et. al.: „*SensorCloud Protokoll*“, SensorCloud, 2013 http://bscw.fh-koeln.de/bscw/bscw.cgi/d4101331/SensorCloud%20Protokoll.pdf

[SCSTAT] T. Partsch: „*SCSTAT*“, Tool zur Visualisierung von Messwerten für die SensorCloud, 2013, https://babeauf.nt.fh-koeln.de/scstat/

[SWP-DS] Heitze, Lassmann, Scholz: „*DatastoreSync*“, Software Praktikum. 2013 http://bscw.fh-koeln.de/bscw/bscw.cgi/d3646128/SWP%20-%20DatastoreSync.7z

[TBAP3] D. Scholz etl al.: „*TBAP3 lokale Sensorverwaltung*“, Fachhochschule Köln, Gruppe TB, Köln, 2014.

[VIRTSEN] T. Partsch, D. Scholz: „*SensorCloud-Dienste als virtuelle Sensoren und Aktoren*“, Fachhochschule Köln, Gruppen FDBS und TB, Köln, 2014.