SensorCloud

DBAP 3 Semantische Analyse mittels Ontologien

15.01.2014, Henning Budde

Gregor Büchel

Andreas Lockermann

Thomas Partsch

Sensor- und AktorSemantik

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.

Um die Heterogenität von Sensoren und Aktoren geeignet verwalten zu können, wird auf das logische Konzept einer Ontologie

O = (C, R, HC, A0)

([1], [2]) Bezug genommen. Hierbei ist C ein kontrolliertes Vokabular, das z.B. mittels OWL-Klassen modelliert werden kann ([3]), R eine Menge von Relationen (in OWL: ObjectProperties), HC eine Begriffshierachie und A0 eine Menge von Axiomen, die mittels einer Beschreibungslogik (Description Logic, [4]) modelliert werden kann. Ein Beispiel für eine in OWL definierte Ontologie für Sensoren und Aktoren im Bereich der Hausautomation ist durch das System DogOnt der Universität Turin gegeben [5].

Persistente Anteile einer Ontologie, wie die Stammdaten zur Beschreibung von Klassen des kontrollierten Vokabulars C und der Relationen R sowie ihre Instanzen c und die Relationstupel r(x, y) (die Instanzen von r) können durch Entitäten bzw. Relationen zwischen Entitäten in einem DBMS verwaltet werden [6]. Auch besteht die Möglichkeit, wenn das Datenmodell des DBMS diese strukturelle Differenzierung bietet, eine Begriffshierachie HC abzubilden.

Da das Datenmodell des FDBS des Projekts SensorCloud eine hohe strukturelle Differenziertheit bietet [7] und von der Konzeption her bereits als Top-Level-Ontologie entwickelt wurde ([6] …), bot sich die Möglichkeit alle persistenten Anteile einer Sensor- und Aktor-Ontologie durch Bestandteile des konzeptionellen Schemas des FDBS abzubilden, wie es als Übersicht in folgender Tabelle dargestellt ist:

Bestandteile von Ontologien O für Sensoren und Aktoren	**Werden dargestellt durch Entität**	
Kontrolliertes Vokabular der Sensor-Klassen C	SensorProdukt	K.1
Kontrolliertes Vokabular der Aktor-Klassen C	AktorProdukt	K.2
Instanzen der Sensor-Klassen C	Sensor	K.1
Instanzen der Aktor-Klassen C	Aktor	K.2
Top-Level-Knoten der Begriffshierachie der Sensoren	SensorTyp	K.1

Top-Level-Knoten der Begriffshierachie der Aktoren	AktorTyp	K.2
Relationen zwischen Sensoren	SensorVerbund	K.1
Relationen zwischen Aktoren	AktorVerbund	K.2
Relationen zwischen Sensoren und Aktoren (regelbasierte Steuerung von Aktoren)	SensorEvent, AktorEvent, EventMitglieder, Event, EventAktion, EventBenachrichtigung	K.3

Die Betrachtung von ontologischen Axiomen im engeren Sinne (prädikatenlogische Konstrukte, die durch einen Reasoner maschinell entscheidbar sind) sind im Kontext der beantragten FDBS-Arbeitspakete des Projekts SensorCloud theoretisch zu speziell. Im praktischen Kontext der FDBS-Arbeitspakete steht jedoch die Verwaltung von Regelwerken, um regelbasierte Aktoren einer Lokation durch sensorielle Daten und lokationsreleveante Events (z.B. Unwetterwarnungen) steuern zu können (siehe Kapitel 3). Insbesondere soll die Möglichkeit bestehen, für das Laufzeitsystem der Aktorsteuerung einer Lokation, die auf einem Rechner vom Typ LocationMaster installiert ist, Regelwerke aus der FDBS zu generieren.

1. SensorSemantik

In diesem Kapitel wird die SensorSemantik beschrieben. Als erstes wird die Abbildung von Sensoren und SensorProdukten im Datenmodell gezeigt, dann wird eine Parsingvorschrift für die Sensoren dargestellt. Dieses Kapitel zeigt zudem ein kontrolliertes Vokabular für die SensorSemantik, die Darstellung der Parsingvorschrift und API-Funktionen für den Zugriff.

1.1 Sensoren in Datenmodell

Um die Vielfalt von Sensoren verwalten zu können, benötigt man eine geeignete Parsingvorschrift, die es einem Rechnerknoten der SensorCloud (LocationMaster oder Cloud-Knoten) erlaubt Messwerte korrekt zu interpretieren. Parsingvorschriften werden jeweils für ein Sensorprodukt entwickelt. Gemäß der Anforderungsanalyse und dem konzeptionellen Schema (DBAP4) gehört ein Sensor zu einem Sensorprodukt und dieses ist in seiner allgemeinen Struktur durch einen Sensortyp bestimmt.

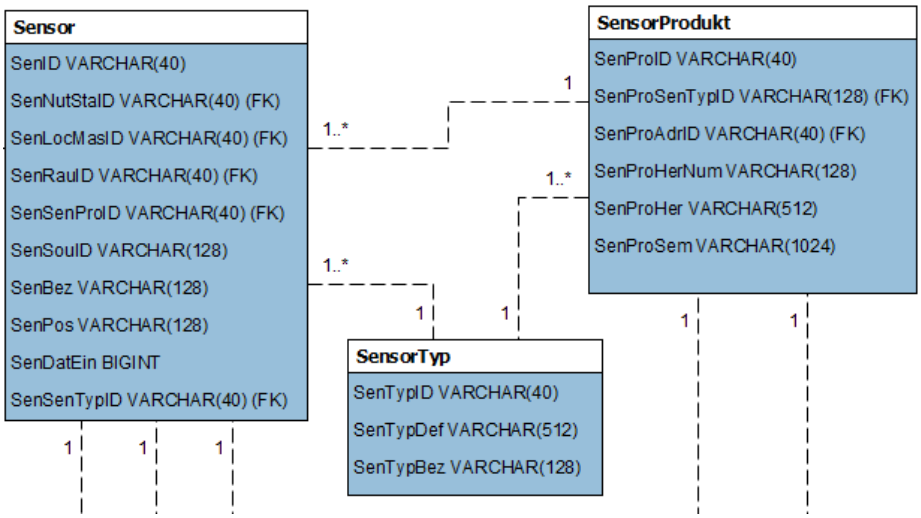

Abbildung 1.1: Sensor, SensorProdukt und SensorTyp im konzeptionellen Schema

Die Entität Sensor beschreibt einen konkreten Sensor, der z.B. bei einem Kunden zu Hause steht und Messwerte aufnimmt. Dieser konkrete Sensor gehört zu einer Produktreihe einer Firma. Alle Sensoren, die zu dieser Produktreihe gehören, haben später die gleiche Parsingvorschrift und somit kann eine Parsingvorschrift speziell für jede Produktreihe (jedes Sensorprodukt) entwickelt werden. Zusätzlich werden den Produkten noch Typen zugeordnet (SensorTyp), dies dient der Kategorisierung von Sensoren und nicht der Beschreibung der Messwerte.

Beispiel: Der Messwert eines Multi-Raum-Sensors wird vom Microcontroller des Sensors an den LocationMaster als Tupel von vier Hexadezimalzahlen übertragen. Zum Beispiel wird folgender Messwert übertragen:

```
00;31;B6B;10E5
```

Erklärung:

00: Bewegungsmelder (00: keine Bewegung, 01: Bewegung)

31: Luftfeuchtigkeit als Hexadezimale Zahl in %

B6B: Temperatur als Hexadezimale Zahl in °C (von 0°C – 50°C, Wert muss durch 100 geteilt werden)

10E5: Luftqualität als Hexadezimale Zahl in VOC (flüchtige organische Stoffe)

1.2 Parsingvorschrift für Sensorprodukte

Die Analyse einiger Sensorprodukte führte zu einer Folge von Deskriptoren, die für eine Parsingvorschrift von Messwerten erforderlich sind. Ein Messwert kann dabei aus mehreren Komponenten (wie aus dem vorherigen Beispiel hervorgeht) bestehen. 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 folgende Deskriptoren Einträge zu hinterlegen:

- nr: Nummer/Stelle des Messwertes; dient für einen direkten Zugriff auf Deskriptoren des n. Messwerts in einer Parsingvorschrift.
- 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[1] (z.B. Temperatur in °C), Messwerte ohne Einheiten (<ohne Einheit>) und Wahrheitswerte.
- dt(w): Datentyp von w (z.B. int, float, …)[2].

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

- 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.

Parsingvorschrift in Data Dictionary Notation nach dem oben beschriebenen Aufbau:

```
n + { [nr + pn | pn + pns] + (w) + (eh) + (dt) + (erlz) + (aggfkt) + (ufkt) + (erl) }
```

mit:

- n und nr sind natürliche Zahlen.
- n beschreibt die Gesamtanzahl, nr ist der Index zu jedem Messwert, der beschrieben wird.
- pn ist eine Zeichenkette aus einem kontrollierten Vokabular für physikalische Namen (VokPhynam).
- pns beschreibt den physikalischen Namen, mit dem der Sensor die eigenen Messwerte identifiziert/benennt (z.B. „Humidity“).
- w hat den Aufbau w_I: w_1-w_2 (Intervall, Beispiel: 0-50.5) oder w_A: w_1,w_2,w_3, … w_N (Aufzählung, Beispiel: na, a0, a1).
- eh ist eine Zeichenkette aus einem kontrollierten Vokabular von Einheiten (Vokeh) oder ist ohne Einheit. Im Fall <ohne Einheit> wird der „eh“-Eintrag im JSON-Format weggelassen.
- dt ist eine Zeichenkette aus einem kontrolliertem Vokabular für Datentypen (Vokdt).
- erlz ist eine Zeichenkette w_1-erlz,w_2-erlz,w_3-erlz, … w_N-erlz.
  Beispiel:
  - na: 0 => nicht aktiviert
  - a_0: 1 => aktiviert & aus
  - a_1: 1 => aktiviert & an

  erlz gibt für jeden möglichen Zustand des Messwerts, der durch einen Aufzählungswert m gegeben ist, eine Erläuterung in der Form: w_i : Erläuterung_Zustand_i.
- aggfkt beschreibt die anwendbaren Aggregatsfunktionen als Aufzählung. Mögliche Aggregatsfunktionen sind dabei: count, sum, min, max und avg (Kontrolliertes Vokabular VokAggfkt).
- ufkt ist eine Zeichenkette, die die Umrechnung des Messwertes beschreibt. $w ist hierbei der Platzhalter für den Messwert.
  - Beispiel:

- *hex2dec*($w)/100
  - *hex2dec* beschreibt hierbei die Funktion zur Umrechnung eines Hex-Wertes in eine Dezimalzahl.
- Die möglichen Funktionen müssen in einem kontrolliertem Vokabular (VokUfkt) definiert sein.

- erl ist eine Zeichenkette, die diese Komponente als Fließtext beschreibt.
- Anmerkungen:
  1) Nicht ganze Zahlen werden durch Punkte in Vor- und Nachkommastellen aufgeteilt. (Beispiel: 0.5)
  2) nr und pns sind optional: Eine Parsingvorschrift enthält entweder nr oder pns.
  3) Erhält ein Attribut keinen Wert, wird dieses im JSON-Format weggelassen.

1.3 Kontrolliertes Vokabular für die SensorSemantik

Für die im vorherigen Kapitel definierte Parsingvorschrift müssen verschiedene kontrollierte Vokabulare festgelegt werden:

- VokPhynam
- VokEH
- VokDT
- VokAggfkt
- VokUfkt

Die zu definierenden Vokabulare werden aus den zur Zeit auf der Teststrecke eingesetzten Sensoren gebildet[3].

VokPhynam:

Physikalischer Name	**Beschreibung**
Bewegungsmelder	Meldet Bewegung (0, 1)
Luftfeuchte	
Temperatur	
Luftqualität	
Zustand	Zustand, z.B. Tür auf oder zu (0, 1)
Bild	Hat als Messwert ein Bild

VokEH:

Einheit	**Erläuterung**

[3] Das Gesamtvokabular ist inzwischen umfangreicher als das in SENML gegebene kontrollierte Vokabluar [8].

°C	
KWH	KW pro Stunde (h)
Ampere	
%	
Watt	
Volt	

VokDT:

Für das kontrollierte Vokabular werden zunächst die folgenden Datentypen als zulässige Datentypen zugelassen.

Datentyp	**Wertebereichsbeschränkung**
Boolean	
Int	
Decimal	
String	Maximale Länge

Das VokDT orientiert sich bei der Bestimmung der Datentypen an der Referenz ISO/IEC/IEEE 21451-1. Die dort beschriebenen Datentypen sind in primitive (primitive datatypes) und abgeleitete Datentypen (derived datatypes) unterteilt. Die Datentypen sind in der folgenden Tabelle mit den Datentypen aus dem Vokdt gegenübergestellt:

ISO/IEC/IEEE 21451-1 datatypes (Auszug)	**Vokkdt**
Boolean	Boolean
Integer8	Int
UInteger8	
Integer16	Int
UInteger16	
Integer32	Int
UInteger32	
Integer64	Int
UInteger64	
Float32	Decimal
Float64	Decimal
Octet	
IDL struct String	String

VokAggfkt:

Aggregationsfunktion	**Funktionsbeschreibung**
count	Zählt die Anzahl der Messwerte (z. B. in einem Intervall)
sum	Summiert eine Menge von Messwerten auf
max	Gibt den größten Wert zurück
min	Gibt den kleinsten Wert zurück
avg	Berechnet den Mittelwert aus einer Menge von Messwerten

Vokufkt:

Umrechnungsfunktion	**Funktionsbeschreibung**
hex2dec($w)	Wandelt den Wert w von Hexadezimal in Dezimal um
toBool($w)	Parst den Wert w und wandelt ihn in einen Boolean Wert um (true/false) „00“ , „0“, 0.0, 0 -> false „01“ , „1“, 1.0, 1 -> true

Die dargestellten Vokabulare dienen als Grundstock und werden zur Zeit denen auf der Teststrecke eingesetzten Sensoren zugeordnet. Die Menge kann somit bei neu hinzukommenden Sensoren erweitert werden.

1.4 Übersicht der eingesetzten Sensorprodukte

Die folgende Tabelle zeigt die verschiedenen Sensorprodukte, die derzeit auf der Teststrecke eingesetzt werden bzw. deren Einsatz in naher Zukunft angedacht ist. Zu sehen ist jeweils der allgemeine Messwertaufbau mit Einheit, Wertebereich und einer Parsingvorschrift bezogen auf die definierte Parsingvorschrift dargestellt in CSV-Syntax (Kommata als Trennzeichen).

Sensorprodukt	**Messwert**	**Messwerte (schematisch)**	**Parsingvorschrift in CSV-Syntax**
Multiraumsensor	00;31;b6b;10e5	A;B;C;D A: Bewegungsmelder B Wert = B ist 00 oder 01 00 = keine Bewegung, 01 = Bewegung Einheit = [siehe dt] dt = boolean	4; 1;Bewegungsmelder;00,01;;boolean;0:keine Bewegung,1:Bewegung;count;toBool($w);; 2;Luftqualitaet;0-1023;;int;;min,max,avg;hex2dec($w);VOC; 3;Temperatur;0-50;°C;decimal;;min,max,avg;hex2dec($w)/100;;

		B: Luftfeuchte f 0-100 Wert = w Einheit = % C: Temperatur t, Wert = t von 0 bis 50 Einheit = °C D: Luftqualität lq 0 bis 1024 Wert = lq Einheit = (VOC, (flüchtige organische Verbindungen), zunächst ohne Einheit)	4;Luftfeuchte;0-100;%;int;;min,max,avg;hex2dec($w);;
Türsensor	true	A A: Zustand Tür auf bzw. Tür zu Wert = true oder false true=Tür zu, false=Tür auf Einheit = String (Hex)	1; 1;Zustand;true,false;;String;false:Tuer zu,true:Tuer auf;count;;;
Stromerfassungssensor (Photovoltaik B. Schmitz, Kürten)	57;11923.720566;0.002;0.035;0.035;483.22;0.065;395.22	A A: Leistung eines Panelsegments P_i (1 <= i <= 5) Wert: Einheit Kwh B B: Strom Eingang A String 2	8; 1;Wirkleistung;0-;Watt;decimal;;min,max,sum;;momentan eingespeiste Wirkleistung in Watt; 2;Leistung;0-;kWh;decimal;;min,max,sum;;Eingespeiste Gesamt Leistung in kWh; 3;DC Strom Eingang A String 1;0-;Ampere;decimal;;min,max,sum;;; 4;DC Strom;0-;Ampere;decimal;;min,max,sum;;DCStrom Eingang A String 2; 5;Strom;0-;Ampere;decimal;;min,max,sum;;DCStrom Eingang A String 3; 6;Spannung;0-;Volt;decimal;;min,max,sum;;DC Spannung Eingang A; 7;Strom;0-;Ampere;decimal;;min,max,sum;;DC Strom Eingang B String 1; 8;Spannung;0-;Volt;decimal;;min,max,sum;;DC Spannung Eingang B;
Schaltbare Steckdose	true	A A: Schaltzustand der Steckdose SZ Wert: SZ (true, false) mit false=aus, true=an Einheit = String (Hex)	1; 1;Zustand;true,false;;String;false:aus,true:an;count;;;
VisionSensor	/user/01.jpg	B B: Bild	1;Bild;;jpg;byte;;;Pfad zum Bild;
Temperatur- und Luftfeuchtesensor	24	A oder B A: Luftfeuchte f 0-100 Wert = w Einheit = % B: Temperatur t, Wert = t von 0 bis 50 Einheit = °C	2; 1;Temperatur;0-50;°C;int;;min,max,sum;hex2dec($w)/100;; 2;Luftfeuchte;0-100;%;int;;min,max,sum;hex2dec($w);;

Ausführliches Beispiel anhand eines Multiraumsensors:

Parsingvorschrift in CSV-Syntax	**Erklärung**
4;	Anzahl der Komponenten des Multi-Raum-Sensors
1;	Index der ersten Komponente
Bewegungsmelder;	pn (physikalischer Name)
00,01;	w (Wertebereich)
;	eh (Einheit)
boolean;	dt (Datentyp)
00:keine Bewegung, 01:Bewegung;	erlz (Erläuterung der Zustände)
count, max, sum;	Es können die Aggregationsfunktionen *count*, *max* und *sum* eingesetzt werden
toBool($w);	Umrechnung in einen bool-Wert
;	erl (Erläuterung)
2;	Index der zweiten Komponente
Luftqualitaet;	pn (physikalischer Name)
0-1023;	w (Wertebereich)
;	eh (Einheit)
int;	dt(Datentyp)
;	sem (Semantik)
min,max,avg;	Es können die Aggregationsfunktionen *min*, *max* und *avg* eingesetzt werden
hex2dec($w);	Umrechnungsfunktion
VOC;	erlz (Erläuterung der Zustände)
3;	Index der dritten Komponente
Temperatur;	pn
0-50;	w
°C;	eh
decimal;	dt
;	erlz (Erläuterung der Zustände)
min,max,avg;	Es können die Aggregationsfunktionen *min*, *max* und *avg*

	eingesetzt werden
hex2dec($w)/100;	Umrechnungsfunktion
Temperatur in °C;	erl
4;	Index der vierten Komponente
Luftfeuchte;	pn
0-100;	w
%;	eh
int;	dt
;	erlz (Erläuterung der Zustände)
min,max,avg;	Es können die Aggregationsfunktionen *min*, *max* und *avg* eingesetzt werden
hex2dec($w);	Umrechnungsfunktion
Luftfeuchte in %	erl

1.5 Darstellung der Parsingvorschrift

Die definierte Parsingvorschrift wird in einem einheitlichen Format abgebildet in dem Attribut SenProSem der Entität SensorProdukt abgelegt. Das gewählte Format der Parsingvorschrift ist die Java Script Object Notation (JSON). 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 Parsingvorschrift nicht dauerhaft ausgetauscht werden muss, ist der durch die umfangreichere Beschreibung anfallende „Overhead“ zu vernachlässigen.

Beispiel: Parsingvorschrift in JSON

```
{
    "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>

},

{

…

}

]

}

Die Darstellung der Parsingvorschrift des Multiraumsensors im JSON-Format wird im Folgenden dargestellt:

```
{
        "n":4,
        "parvor":[
                {
                        "nr":1,
                        "pn":"Bewegungsmelder",
                        "w":{"enum":["00","01"]},
                        "dt":"boolean",
                        "erlz":{
                                "00":"Keine Bewegung",
                                "01":"Bewegung"
                        },
                        "aggfkt":["min","count","max"],
                        "ufkt":"toBool($w)"
                },
                {
                        "nr":2,
                        "pn":"Luftfeuchte",
                        "w":{"min":0, "max":100},
                        "eh":"%",
                        "dt":"decimal",
                        "aggfkt":["max","min","avg"],
                        "ufkt":"hex2dec($w)/100"
                },
                {
                        "nr":3,
                        "pn":"Temperatur",
                        "w":{"min":0, "max":50},
                        "eh":"°C",
                        "dt":"decimal",
                        "aggfkt":["max","min","avg"],
                        "ufkt":"hex2dec($w)/100"
                },
                {
                        "nr":4,
                        "pn":"Luftqualitaet",
                        "w":{"min":0, "max":65535},
                        "dt":"int",
                        "aggfkt":["max","min","avg"],
                        "ufkt":"hex2dec($w)"
                }
        ]
}
```

1.6 API-Funktionen für den Zugriff auf die SensorSemantik

Für den Zugriff auf die SensorSemantik wird eine API zur Verfügung gestellt. Diese Schnittstelle benötigt nur die Information, von welchem Sensor ein Messwert kommt und zusätzlich, wenn nur eine bestimmte Komponente erfragt werden soll, deren Nummer. Möchte man beispielsweise von einem Sensor den physikalischen Namen der n. Komponente haben, wäre diese Abfrage folgendermaßen zu stellen:

getType(<SensorID>, n)

Folgende API Funktionen sind derzeit geplant:

getFullSpecification():

Beschreibung: Holt die komplette Beschreibung aller Komponenten eines Messwerts

Parameter: Sensor ID

Return: Objekt mit vollständiger Beschreibung aller Komponenten eines Messwerts

getNumberOfComponents():

Beschreibung: Gibt die Anzahl der Komponenten eines Messwerts zurück

Parameter: Sensor ID

Return: Anzahl der Komponenten eines Messwerts

getFullSpecificationOfComponent():

Beschreibung: Holt die komplette Beschreibung einer Komponente eines Messwerts

Parameter: Sensor ID, Komponenten Nummer

Return: Objekt mit vollständiger Beschreibung einer Komponente eines Messwerts

getTypes():

Beschreibung: holt alle physikalische Namen (pn) eines Sensor, z.B. Bewegungsmelder, Luftfeuchte, Temperatur und Luftqualität

Parameter: Sensor ID

Return: Zeichenketten Array mit allen pn

getType():

Beschreibung: holt einen physikalischen Namen zu einer Komponente

Parameter: Sensor ID, Komponenten Nummer

Return: pn

getUnit():

Beschreibung: holt die physikalische Einheit einer Komponente

Parameter: Sensor ID, Komponenten Nummer

Return: eh

getRange():

Beschreibung: holt den Wertebereich einer Komponente

Parameter: Sensor ID, Komponenten Nummer

Return: Darstellung des Wertebereichs w

getDataType():

Beschreibung: holt den Datentyp einer Komponente

Parameter: Sensor ID, Komponenten Nummer

Return: dt (Zeichenkette mit Datentypnamen)

getDesS():

Beschreibung: holt die Erläuterung der Zustände, die durch einen Aufzählungswert w repräsentiert werden

Parameter: Sensor ID, Komponenten Nummer

Return: erlz (Zeichenkette mit Erläuterungen)

getAgg():

Beschreibung: holt die möglichen Aggregationsfunktionen einer Komponente

Parameter: Sensor ID, Komponenten Nummer

Return: Zeichenketten Array mit Aggregationsfunktionen (aggfkt)

getFunc():

Beschreibung: holt die Umrechnungsfunktion einer Komponente

Parameter: Sensor ID, Komponenten Nummer

Return: Zeichenkette mit dem Inhalt des JSON-Elements ufkt

getDes():

Beschreibung: holt die Erläuterung zu einer Komponente

Parameter: Sensor ID, Komponenten Nummer

Return: Zeichenkette mit dem Inhalt des JSON-Elements erl

getPhynamSen():

Beschreibung: holt den physikalischen Namen aus Sicht des Sensors

Parameter: Sensor ID, Komponenten Nummer

Return: Zeichenkette mit dem Inhalt des JSON-Elements pns

Übersicht über die Prototypen der API-Funktionen:

SensorProduktSemantik [] *getFullSpecification*(String sourceID)
ParVor *getFullSpecificationOfComponent* (String sourceID, int componentNr)
int *getNumberOfComponents*(String sourceID)
String *getUnit*(String sourceID, int componentNr)
String *getDataType*(String sourceID, int componentNr)
String [] *getTypes*(String sourceID)
String *getType*(String sourceID, int componentNr)
String *getDes*(String sourceID, int componentNr)
String *getDesS*(String sourceID, int componentNr)
String *getRange*(String sourceID, int componentNr)
String *getAgg*(String sourceID, int componentNr)
String *getFunc*(String sourceID, int componentNr)
String *getPhynamSen*(String sourceID, int componentNr)
List<SensorProduktSemantik> getAll()

1.7 SensorVerbund im Datenmodell

Das folgende Unterkapitel beschäftigt sich mit Verbünden von Sensoren. Ein Verbund ist ein Zusammenschluss von mehreren (unabhängigen) Sensoren. Zum Beispiel der Zusammenschluss von allen Temperatursensoren in einem Haus, der die Durchschnittstemperatur im gesamten Haus ermitteln sollen.

Die folgende Abbildung zeigt die Darstellung der Sensorverbünde im Datenmodell:

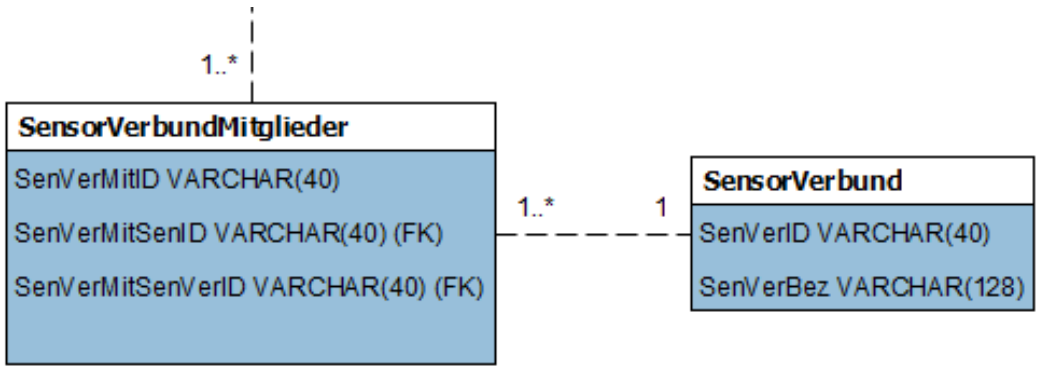

Abbildung 1.2: SensorVerbund im konzeptionellen Schema

Über die Entität SensorVerbundMitglieder wird eine n:n-Beziehung zwischen den Entitäten SensorVerbund

und Sensor hergestellt. Ein Sensor kann demnach mehreren Verbünden angehören und ein Verbund enthält mehrere Sensoren (SensorVerbundMitglieder).

2. AktorSemantik

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 Beschreibungssprache/Parsingvorschrift definiert.

2.1 Aktoren im Datenmodell

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

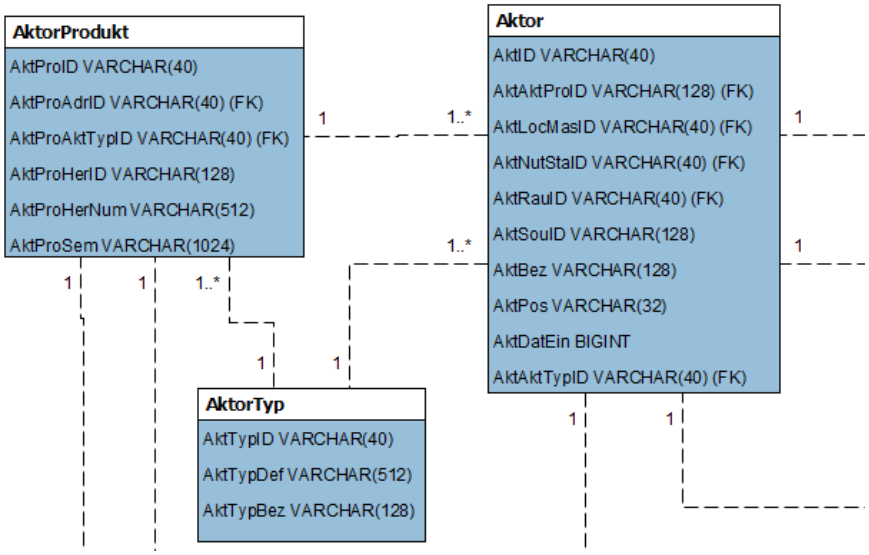

Abbildung 2.1: Aktor, AktorProdukt und AktorTyp im konzeptionellen Schema

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

2.2 Parsingvorschrift für AktorProdukte

Die folgende Parsingvorschrift bezieht sich auf die Funktionen, die ein Aktorprodukt anbietet. Um eine Parsingvorschrift 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.

[4] Die Spezifikation dieser Eigenschaften folgt der Beschreibung von Funktionen in einer RPC-ähnlichen Struktur (vgl. a. die Beschreibung von Funktionen in einer RTDL [9])

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
- 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

Ansatz für eine Parsingvorschrift in Data Dictionary Notation nach dem oben beschriebenen Aufbau:

n + { fnr + fnam + (fart) + fpanz + { pnr + pnam + part + (pw) + (peh) + (pdt) + (perlz) } + franz + { rnr + rnam + rart + (rw) + (reh) + (rdt) + (rerlz) } + (ferl) }

mit:

- n (natürliche Zahl) beschreibt die Gesamtanzahl der Funktionen
- fnr ist der Index zu jeder Funktion, die beschrieben wird
- fnam ist eine Zeichenkette und beschreibt den Funktionsnamen (z.B. forward oder arm)
- fart ist eine Zeichenkette und beschreibt den Funktionsart des Aktors
- fpanz ist eine Zahl und gibt die Anzahl der Parameter einer Funktion an
- pnr ist eine Zeichenkette und beschreibt die Parameternummer

- pnam ist eine Zeichenkette und beschreibt den Parameternamen
- part ist eine Zeichenkette und beschreibt die Parameterart aus einem kontrollierten Vokabular (VokPArt)
- pw hat den Aufbau w_I: w_1-w_2 (Intervall, Beispiel: 0-50.5) oder w_A: w_1,w_2,w_3, … w_N (Aufzählung, Beispiel: na,a0,a1)
- peh ist eine Zeichenkette aus einem kontrollierten Vokabular (VokEH) oder ist ohne Einheit
- pdt ist ein Datentyp aus einem kontrolliertem Vokabular (VokDT, Zeichenkette)
- perlz ist eine Zeichenkette wp_1-perlz,wp_2-perlz,wp_3-perlz, … wp_N-perlz Beispiel:
  - na: 0 => nicht aktiviert
  - a_0: 1 => aktiviert & aus
  - a_1: 1 => aktiviert & an
- franz ist eine Zahl und gibt die Anzahl der Rückgabewerte einer Funktion an
- rnr ist eine Zeichenkette und beschreibt die Rückgabewertnummer
- rnam ist eine Zeichenkette und beschreibt den Rückgabewertnamen
- rart ist eine Zeichenkette und beschreibt die Rückgabewertart aus einem kontrollierten Vokabular (VokPArt)
- rw hat den Aufbau w_I: w_1-w_2 (Intervall, Beispiel: 0-50.5) oder w_A: w_1,w_2,w_3, … w_N (Aufzählung, Beispiel: na,a0,a1)
- reh ist eine Zeichenkette aus einem kontrollierten Vokabular (VokEH) oder ist ohne Einheit
- rdt ist ein Datentyp aus einem kontrolliertem Vokabular (VokDT, Zeichenkette)
- rerlz ist eine Zeichenkette wr_1-rerlz,wr_2-rerlz,wr_3-rerlz, … wr_N-rerlz Beispiel:
  - na: 0 => nicht aktiviert
  - a_0: 1 => aktiviert & aus
  - a_1: 1 => aktiviert & an
- fkterl ist eine Zeichenkette, die diese Komponente als Fließtext beschreibt

Anmerkung: Nicht ganze Zahlen werden durch Punkte in Vor- und Nachkommastellen aufgeteilt (Beispiel: 0.5)

2.3 Kontrolliertes Vokabular

Für die im vorherigen Kapitel definierte Parsingvorschrift müssen verschiedene kontrollierte

Vokabulare festgelegt werden:

- VokEH
- VokDT
- VokFArt

VokEH:

Einheit	**Erläuterung**
C	Grad in Celsius
KWH	Kilo Watt pro Stunde (h)
Ampere	Stromstärke
%	Prozent
Watt	Leistung
Volt	Spannung

VokDT:

Datentyp	**Wertebereichsbeschränkung**
Boolean	keine
Int	keine
Decimal	keine

VokFArt:

Das kontrollierte Vokabular VokFArt ist im Moment nur exemplarisch für die Aktortypen Rollladen und Markisen entwickelt. Bei der Integration weiterer Aktortypen kann dieses Vokabular nach der im Folgenden hier ausgeführten Tabelle erweitert werden.

Funktion	**Erläuterung**
up	Alle Funktionen, welche veranlassen das etwas hochgefahren wird, werden auf "up" gematched.
down	Alle Funktionen, welche veranlassen das etwas runtergefahren wird, werden auf "down" gematched.
in	Alle Funktionen, welche veranlassen das etwas eingefahren wird, werden auf "in" gematched.
out	Alle Funktionen, welche veranlassen das etwas ausgefahren wird, werden auf "out" gematched.

VokPArt:

Parameterart	**Erläuterung**
distance	Alle Parameter, welche eine Strecke beschreiben, werden auf "distance" gematched.

speed	Alle Parameter, welche eine Geschwindigkeit beschreiben, werden auf "speed" gematched.
angle	Alle Parameter, welche einen Winkel beschreiben, werden auf "angle" gematched.

Die dargestellten Vokabulare dienen als Grundstock und können bei neu hinzukommenden Aktoren gegebenenfalls erweitert werden.

2.4 Darstellung der Parsingvorschrift

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

Beispiel: Parsingvorschrift 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>
}
```

Die Darstellung der Parsingvorschrift im JSON-Format für die Menge der Beispielfunktionen eines Rollladenaktors die aus zwei Elementen (n=2) besteht, wird im Folgenden dargestellt:

```
{
    "n":2,
    "parvor":[
        {
            "fnr":1,
            "fnam":"hoch",
            "fart":"up",
            "fpanz":1,
            "param":[
                {
                    "pnr":1,
                    "pnam":"distanz",
                    "part":"distance",
                    "pw":{"min":1, "max":100},
                    "peh":"%",
                    "pdt":"int"
                }
            ],
            "franz": 1,
            "r": [
                {
                    "rnr": 1,
                    "rnam": "status",
                    "rart": "status",
                    "rw":{"enum":[0,1]},
                    "rdt":"boolean",
                    "rerlz":{
                        "0":"nicht vollständig hochgefahren",
                        "1":"vollständig hochgefahren"
                    }
                }
            ]
    },
    {
            "fnr":2,
            "fnam":"runter",
            "fart":"down",
            "fpanz":1,
            "param":[
                {
                    "pnr":1,
                    "pnam":"distanz",
                    "part":"distance",
                    "pw":{
                            "min":1,
                            "max":100
                    },
```

```
                        "peh":"%",
                        "pdt":"int"
                    }
                ],
                "franz": 1,
                "r": [
                    {
                        "rnr": 1,
                        "rnam": "status",
                        "rart": "status",
                        "rw":{"enum":[0,1]},
                        "rdt":"boolean",
                        "rerlz":{
                            "0":"nicht vollständig hochgefahren",
                            "1":"vollständig hochgefahren"
                        }
                    }
                ]
        }
],
"erl":      "Funktion 1 lässt den Aktor die Rollade hochfahren.
            Der Übergabeparameter bestimmt die Höhe in Prozent. Der
            Rückgabewert sagt aus, ob die Rollade vollständig
            hochgefahren ist oder nicht. Funktion 2 lässt den Aktor die
            Rollade runterfahren. Der Übergabeparameter bestimmt die Höhe
            in Prozent. Der Rückgabewert sagt aus, ob die Rollade
            vollständig runtergefahren ist oder nicht."
}
```


2.5 AktorVerbund im Datenmodell

Analog zu Verbünden von Sensoren lassen sich auch Verbünde von Aktoren definieren. Ein Aktorverbund fasst mehrere gleiche aber auch andersartige Aktoren zu einem Verbund zusammen. Beispielsweise lassen sich alle Jalousien eines Hauses mittels eines Verbunds gruppieren und mit nur einem Steuerbefehl ansteuern.

Die folgende Abbildung zeigt die Darstellung der Aktorverbünde im Datenmodell:

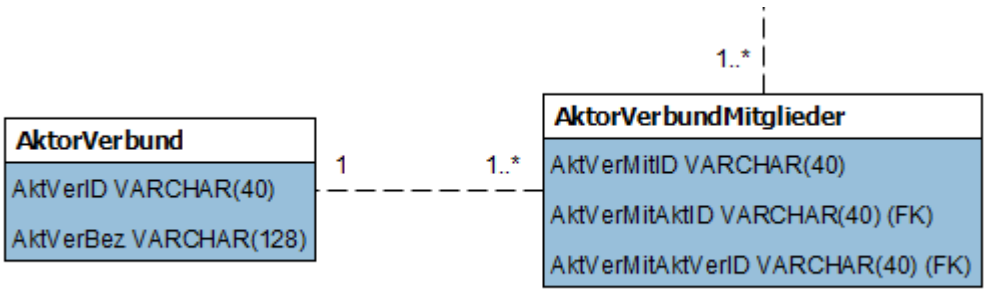

Abbildung 2.2: AktorVerbund im konzeptionellen Schema

Über die Entität AktorVerbundMitglieder wird eine n:n-Beziehung zwischen den Entitäten AktorVerbund und Aktor hergestellt. Ein Aktor kann demnach mehreren Verbünden angehören und ein Verbund kann mehrere Aktoren enthalten.

3. Regelbasierte Steuerung von Aktoren

Aktoren sollen regelbasiert gesteuert werden können. Die Idee dazu sieht wie folgt aus: Ein Satz von Basisregeln zur Steuerung von Aktoren ist in der DB hinterlegt. Dieses Kapitel beschreibt einen Ansatz für eine regelbasierte Steuerung von Aktoren.

3.1 Eventverhalten mit Verbünden

Das Eventverhalten kann anstelle von einzelnen Sensoren als Voraussetzung für ein Event (if <Voraussetzung>) auch mit Verbünden gesehen 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.

3.2 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“ { $V_1 \wedge V_2 \wedge V \ldots \wedge V_m$ } und A „Element aus“ { $A_1 \wedge A_2 \wedge \ldots \wedge A_n$ } 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) Λ hat(4711, 29) Λ mw < 24

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

V = iTypV(4910, Tür/Auf/Zu) Λ hat(4910, Zu) Λ 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“))

)

3.3 Abbildung im Datenmodell

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.

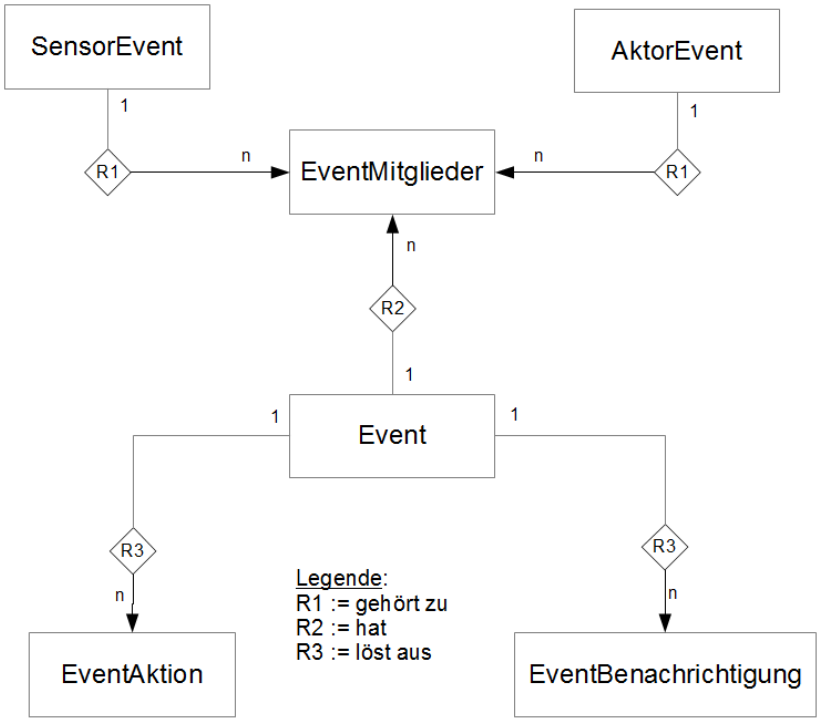

Abbildung 3.1: Darstellung der Entitäten im Zusammenhang mit Events als ERD

3.4 Erzeugung von Regeln aus der Datenbank

Zur Generierung von Regeln innerhalb der SensorCloud werden derzeit drei Dokumente benötigt. Es handelt sich dabei jeweils um ein Dokument zur Beschreibung der **Lokation**, der **Konfiguration** von Sensoren und Aktoren und von **Regeln**.

Das **SensorCloud-Lokation**-Dokument beschreibt Lokationen, also den strukturellen Aufbau von Gebäuden mit Etagen und Räumen. Dabei ist einem Raum als kleinster/tiefster Entität eine Menge von Sensoren und Aktoren zugeordnet. Das Dokument wird aus verschiedenen Attributen mehrerer Tabellen der Datenbank zusammengesetzt. Involviert sind die Entitäten `Lokation`, `Gebaeude`, `Etage`, `Raum`, `Sensor` und `Aktor`.

Das **SensorCloud-Konfiguration**-Dokument beschreibt Aktoren und Sensoren. Dies umfasst sowohl den Typ, die Adressierung und Ansprechbarkeit der Geräte, als auch das Intervall, in dem Messwerte geliefert oder abgerufen werden sollen, und die Beschreibung, wie Messwerte interpretiert werden müssen (Semantik). Das Dokument wird aus verschiedenen Attributen mehrerer Tabellen der Datenbank zusammengesetzt. Involviert sind die Entitäten `Sensor`, `SensorTyp`, `SensorProdukt`, `SensorKonfiguration`, `Aktor`, `AktorTyp`, `AktorProdukt`, `AktorKommunikation` und `NetzwerkManager`.

Das **SensorCloud-Regel**-Dokument beschreibt eine Menge von Regeln. Regeln beziehen sich auf Sensoren und Aktoren, die im Konfigurations-Dokument angegeben wurden und beschreiben, welche Aktionen Aktoren ausführen sollen, wenn Sensoren, beziehungsweise deren gelieferte Messwerte, bestimmte Bedingungen erfüllen. Das Dokument wird aus verschiedenen Attributen mehrerer Tabellen der Datenbank zusammengesetzt. Involviert sind die Entitäten `Event`, `SensorEvent`, `AktorEvent`, `EventAktion` und `EventBenachrichtigung`.

Literatur

[1] Gregor Büchel: „Datenbanken und Wissensrepräsentation“ (Folien: http://www.nt.fh-koeln.de/fachgebiete/inf/buechel/dbw.html, Aufruf: 04.12.2013)

[2] Heiner Stuckenschmidt: „Ontologien – Konzepte, Technologien und Anwendungen“, Heidelberg (Springer) 2009

[3] Web Ontology Language (OWL),
http://www.w3.org/TR/owl2-overview/
Aufruf: 04.12.2013

[4] Franz Baader, Ian Horrocks, Ulrike Sattler: „Description Logic“ in: S. Staab, R. Studer: “Handbook on Ontologies”, Berlin, Heidelberg (Springer), 2004, S. 3-28

[5] DOG Ont: Dario Bonino, D.; Emiliano Castellina, E.; Fulvio Corno, F.: “The DOG Gateway: Enabling Ontology-based Intelligent Domotic Environments“, in: IEEE Transactions on Consumer Electronics, Vol. 54, No. 4, NOVEMBER 2008, S. 1656–1664

Bonino, Dario; Castellina, Emiliano; Corno, Fulvio (2008): DOG: An Ontology-Powered OSGi Domotic Gateway. In: 2008 20th IEEE International Conference on Tools with Artificial Intelligence: IEEE, S. 157–160

[6] „Ontologierorientierter Datenbankentwurf für eine föderiertes Datenbanksystem im Projekt SensorCloud“ ,
http://dmc.fim.uni-passau.de/wp-content/uploads/2013/03/Buechel_DMC_2013.pdf
Aufruf: 10.12.2013

[7] Konzeptionelles Schema,
http://vma.web.fh-koeln.de/index.php?option=com_joomdoc&task=doc_download&gid=15&Itemid=91
Aufruf: 10.12.2013

[8] Media Types for Sensor Markup Language,
http://tools.ietf.org/html/draft-jennings-senml-10,
Aufruf: 10.12.2013

[9] „Spezifikation einer Robotic Task Definition Language (RTDL)“
Markus Teske, Master Thesis, 2011
http://opus.bibl.fh-koeln.de/volltexte/2011/311/pdf/MA_Spezifikation_RTDL.pdf
Aufruf: 10.12.2013