DB Systemintegration

DBAP12

15.12.2014, Henning Budde

Thomas Partsch

1. Gesamtübersicht Teststrecke

Nachfolgende Systemübersicht zeigt das Zusammenwirken aller im Projekt entwickelten Module der Forschungsgruppe Föderierte Datenbanksysteme (FDBS) sowie eine Auswahl an Komponenten der Forschungsgruppe Testbett. Das Arbeitspaket „DB12 Systemintegration“ beschreibt die einzelnen Komponenten und deren Zusammenwirken im Gesamtsystem aus Sicht der Forschungsgruppe FDBS, sowie die eingesetzte Hard- und Software (Kapitel 2). Detaillierte Modulbeschreibungen der hier aufgezeigten Komponenten finden sich in den Arbeitsberichten der FDBS- und Testbett-Gruppe und des Konsortialpartners RWTH-Aaachen/Comsys wieder.

Kapitel 3 dieses Berichts beschreibt den Nachrichtenfluss des Gesamtsystems und Kapitel 4 die FDBS-Module. Weitergehend wird in Kapitel 5 auf die Aktorsteuerung der SensorCloud eingegangen. Eine Erweiterung des SensorCloud-Systems kann über virtuelle Aktoren und virtuelle Sensoren erfolgen, welche in Kapitel 6 beschrieben ist.

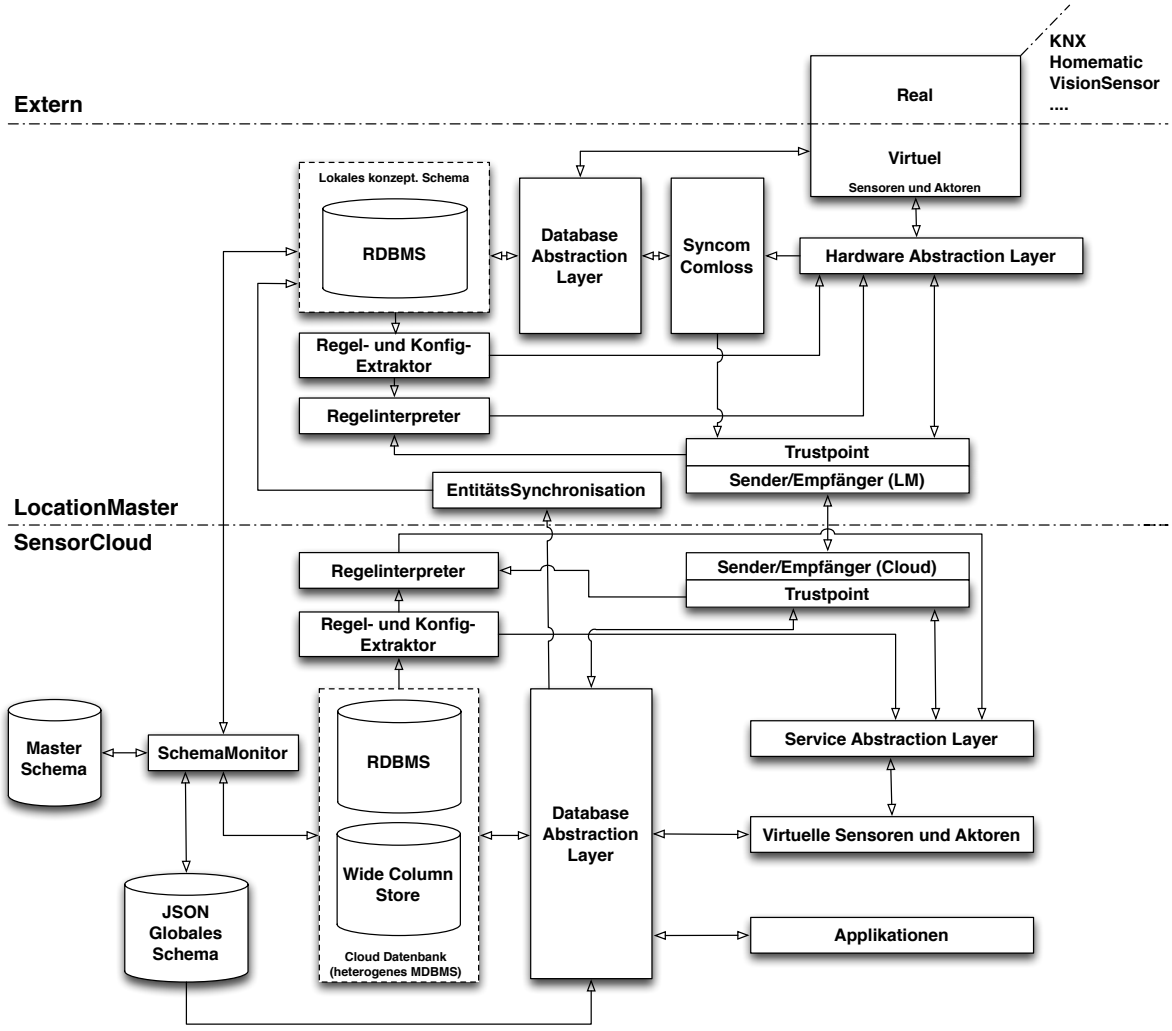

Abbildung 1: Gesamtübersicht Teststrecke

2. Systemlandschaft

Eine minimale Anforderung von Konsistenz wird in Cassandra mit einem Replikationsfaktor größer zwei und

Lese- und Schreibvorgängen mit QUORUM[1] erreicht. Da dies zu einer redundanten Speicherung von Daten führt, ermöglicht es eine finanziell wirtschaftliche horizontale Skalierbarkeit von verteilten Datenbanksystemen durch kostengünstige Hardware. Dieser Ansatz findet sich auch in dem durch Facebook im Jahr 2011 gestarteten Open Compute Project[2] zum Hardwaredesign für verteilte Systeme wieder. Eine Hardwareredundanz wird im Open Compute Hardwaredesign aus Kostengründen weggelassen, da moderne verteilte Datenbanksysteme aufgrund der softwareseitigen Datensatzreplikation nach Austausch eines Knotens selbständig einen konsistenten Zustand wiederherstellen können [PRACAS].

2.1 Hardware (Cloud)

Bei der genutzten Hardware handelt es sich um Desktopsysteme der Firma Fujitsu. Um Einflüsse der Festplattenlatenzzeiten auf die Messungen der Lese- und Schreibgeschwindigkeiten der Cassandra Datenbank zu minimieren werden SSD-Festplatten verwendet. Tabelle 1 listet die für die Messungen relevanten Ausstattungsmerkmale der eingesetzten Computer des Cassandra-Clusters auf.

Modell	Fujitsu Eprimo P720 E90+
CPU	Intel Core i5-4570, 4x 3.20GHz
RAM	1x 4GB 2Rx8 PC3 12800U
Netzwerkkarte	Intel I217LM/LV Gigabit LAN
Festplatte	256GB Samsung SSD MZ7TD256HAFV
Nettopreis	607 €

Tabelle 1: Hardware der Cassandra Knoten

Die Geschwindigkeit der verbauten Hardware wird über Programme des Linux-Kernels und optionale Linux-Tools bestimmt. Die angegebene Festplatte besitzt eine über *dd*[3] gemessene durchschnittliche nicht gepufferte Lesegeschwindigkeit von 249,5 MB/s bei einer dem eingesetzten ext4-Dateisystem entsprechenden Blockgröße von 4KB. Die mit *hdparm*[4] gemessene durchschnittliche ungecachte Lesegeschwindigkeit beträgt 457,10 MB/s. Im Gegensatz zu dd misst hdparm die Geschwindigkeiten nicht über Dateilesevorgänge auf dem Dateisystem, sondern über ein sequentielles Lesen der Festplattensektoren. Über die eingesetzte Netzwerkkarte wird an einem EX 4200-Switch von Juniper Networks mit *iperf*[5] eine durchschnittliche Übertragungsgeschwindigkeit von 935 Mbit/s gemessen. Dies begrenzt vorab die maximal erreichbaren Schreib- und Lesegeschwindigkeiten des Cassandra Clusters auf die Übertragungsgeschwindigkeit der Netzwerkkarte.

2.2 Software (Cloud)

Als Betriebssystem wird die Debian-Distribution[6] im Wheezy-Release für x86-64 Systeme verwendet. Aufgrund der Integration in den Linux-Kernel wird der unter der GNU GPL v2 veröffentlichte XEN Hypervisor[7] für eine Virtualisierung der Cassandra-Knoten verwendet. Die Gastsysteme sind mit einem Debian Wheezy x86-64 paravirtualisiert, um eine hohe Performance zu erzielen [XENVIR]. Die hohe Performance wird über die Paravirtualisierung des Hosts erreicht. Aufgrund der Virtualisierung können Systemzustände und Systemkonfigurationen bei der Versuchsdurchführung mühelos eingefroren und zu einem späteren Zeitpunkt wiederhergestellt werden.

[1] Mechanismus zur Wahrung der Datenintegrität im Fall eines Teilausfalls
[2] http://www.opencompute.org
[3] http://man7.org/linux/man-pages/man1/dd.1.html
[4] http://man7.org/linux/man-pages/man8/hdparm.8.html
[5] http://iperf.fr
[6] http://www.debian.org
[7] http://www.xenproject.org

2.3 Systemtopologie

Der Systemaufbau besteht aus einem vierknotigen Cassandra-Cluster. Die Cassandra-Knoten sind dabei einzeln auf den vier physikalischen Knoten virtualisiert.

Die Knoten befinden sich in einem separierten privaten Netzwerk und sind über einen ebenfalls virtualisierten Router an das Internet angeschlossen (Abbildung 2).

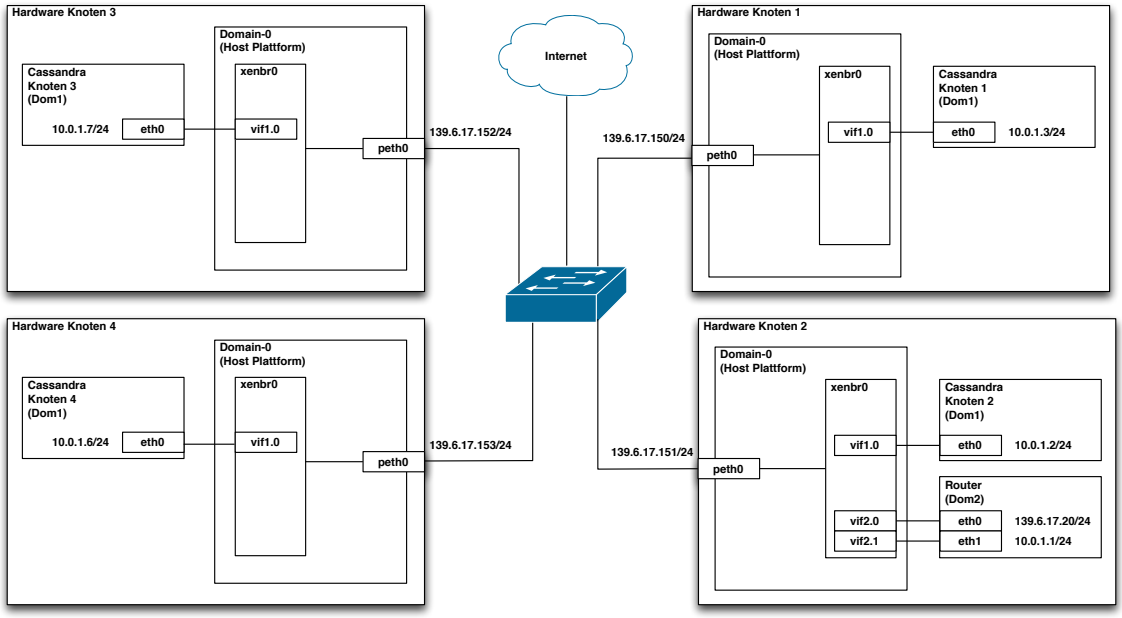

Abbildung 2: Systemtopologie des Versuchsaufbaus

2.4 Hardware (LocationMaster)

Aus der Analyse der Anforderungen an einen Location Master (siehe „Zwischenbericht: Projekt SensorCloud Teilvorhaben: TBAP1“) ergibt sich, dass ein preiswerter Linux-Einplatinenrechner die Anforderungen an einen ersten Prototyp erfüllt. Die Wahl fällt daher auf ein ARM-System „BeagleBoard-xM, Rev. C“ mit 1GHz CPU-Takt und 512MB RAM, das über eine Ethernet-Ankopplung und USB-2.0-Anschlüsse verfügt. Als persistenter Datenspeicher dient eine Micro-SD-Karte. In Tests wurden mit kleinen Beispielprogrammen folgende Hardware-Schnittstellen angesprochen:

- Serielle Schnittstelle RS 232-C (bis zu 115 kBaud).
- USB-Geräte (Tastatur,..)
- DVI (Monitoranschluss) mit HDMI zu DVI-Adapter
- Ethernet-Anschluss (100 MBit/s)

2.5 Betriebssystem (LocationMaster)

Das Betriebssystem soll längerfristig mit Systemupdates versorgt werden. Die Wahl fiel auf das Linuxderivat „Ubuntu 12.04 LTS“ (LTS – Long Term Support), welches fünf Jahre nach Veröffentlichung (26. April 2012) noch mit Updates unterstützt wird.

2.6 Software aus Sicht des FDBS für den LocationMaster

Auf dem LocationMaster ist zur lokalen Verwaltung eine lokale Datenbank installiert. Im Arbeitspaket DBAP2 Deploymentanalyse wurden ausführliche Tests durchgeführt, die zur Nutzung einer PostgreSQL Datenbank führten. Entitäten des Konzeptionellen Schemas für einen LocationMaster können mit Hilfe des SchemaMonitors auf die LocationMaster eingespielt werden.
Zur lokalen Verwaltung der Public Keys für den Trustpoint wird zusätzlich eine lokale MySQL Datenbank eingesetzt.

3. Nachrichtenfluss

Der die Aktorsteuerung betreffende Nachrichtenfluss der SensorCloud-Module wird über den Austausch von SensorData und ActuatorMessages im SensorCloud-Protokoll beschrieben. SensorData und ActuatorMessages werden hierbei im HAL/SAL und dem Regelwerk erzeugt. Dies ermöglicht eine einfache Erweiterung des Gesamtsystems um weitere Dienste durch eine Ankopplung neuer Dienste an den Hardware Abstraction Layer (HAL) bzw. Software Abstraction Layer (SAL) und die Erstellung zusätzlicher Regeln für das Regelwerk. Nachfolgende Systemübersicht zeigt das Zusammenwirken der Cloud- und LocationMaster-Dienste, die für eine Aktorsteuerung benötigt werden und deren Kommunikation über die Nachrichtenflüsse SensorData und ActuatorMessages. Die Trustpoint-Komponenten wurden zur Vereinfachung in Abbildung 3 weggelassen.

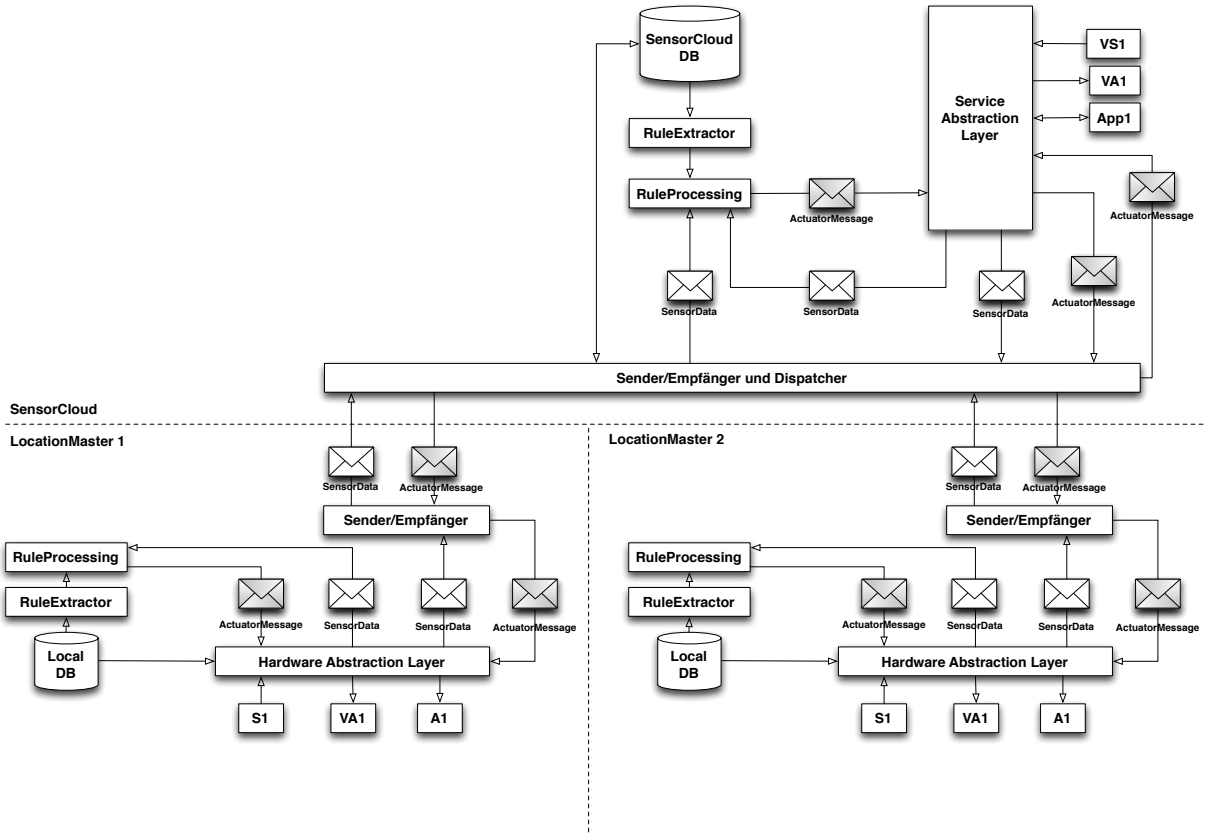

Abbildung 3: Nachrichtenfluss

Aktoren sind physische oder virtuelle Aktoren und werden über den HAL (auf dem LocationMaster) oder den SAL (in der Cloud) an das Gesamtsystem angebunden. Für die Übermittlung von Aktorsteuerbefehlen 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.

4. FDBS-Module

Abbildung 4 zeigt farblich unterlegt die in der Forschungsgruppe *Föderierte Datenbanksysteme* (FDBS) entwickelten und in diesem Kapitel vorgestellten Module. Weitere in dieser Übersicht auftretende Module (Hardware Abstraction Layer und Software Abstraction Layer) gehören in ihrer Entwicklung zum Testbett-System und sind dort beschrieben. Das Modul Trustpoint als zentrale Authentizitäts-/Verschlüsselungs-Komponente wurde von dem Konsortialpartner RWTH/Comsys entwickelt. Die von der FDBS-Forschungsgruppe für das Modul bereitgestellten Datenbankservices sind in [DBAP9] beschrieben. Auf die nachrichtenbasierte Steuerung von Aktoren wird in Kapitel 5 eingegangen. Das Konzept der virtuellen

Sensoren und Aktoren wird in Kapitel 6 erklärt.

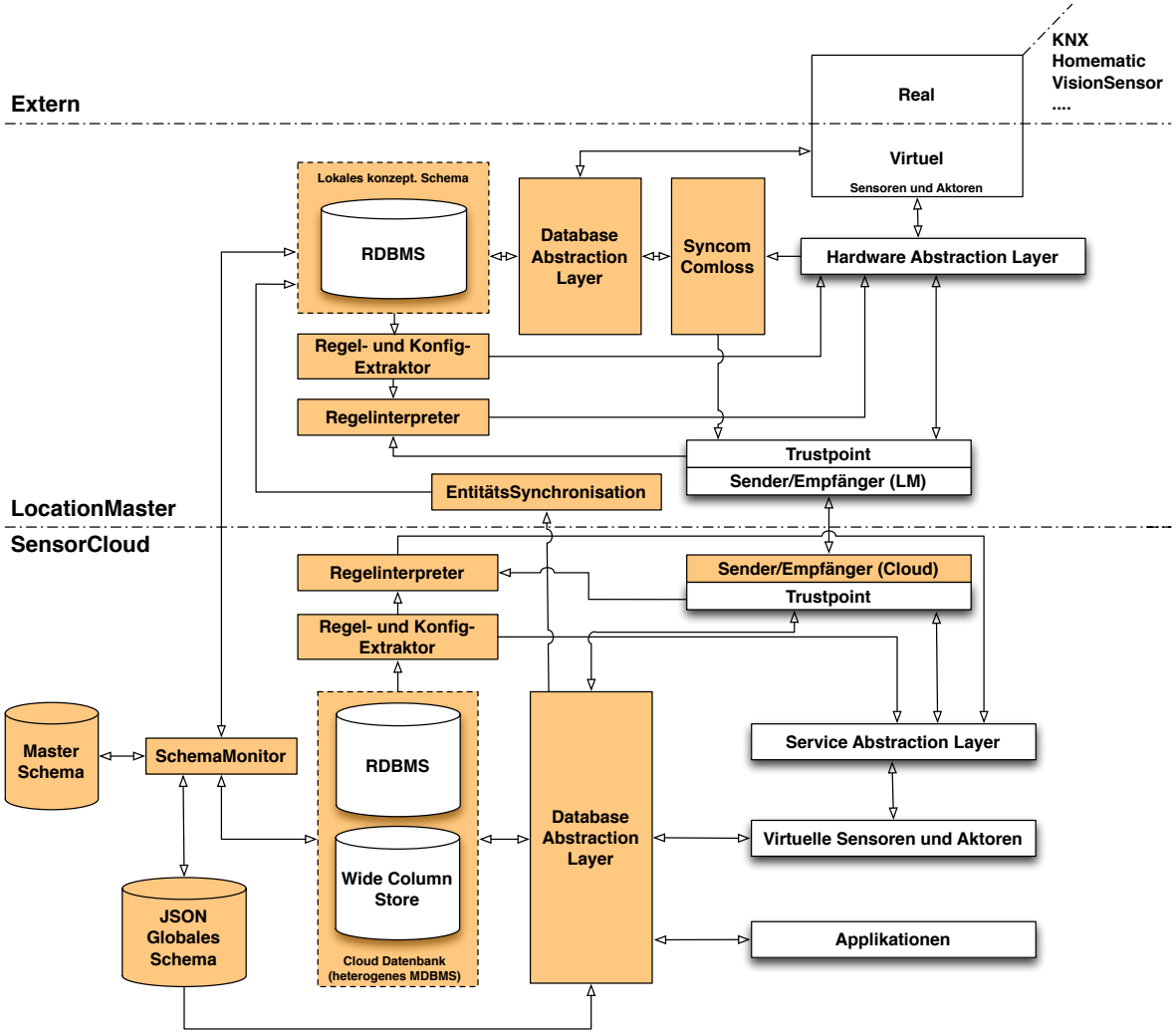

Abbildung 4: FDBS-Module des Gesamtsystems (FDBS-Module sind farblich hervorgehoben)

4.1 Data Abstraction Layer

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.

Das Modul „Database Abstraction Layer“ (DAL) (Abbildung 4) abstrahiert als Middleware den Zugriff von Sensordiensten auf die Datenbanken. Hierzu stellt der DAL [DBAP8] geeignete Dienste für den Zugriff auf die Entitäten des konzeptionellen globalen SensorCloud-Datenbankschemas [DBAP4] 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*.

4.2 Entitätssynchronisation

Das Modul „Entitätssynchronisation“ (Abbildung 4) implementiert die Cloud-Synchronisierung von Stammdaten und 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 4.1) 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 5 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 5 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.

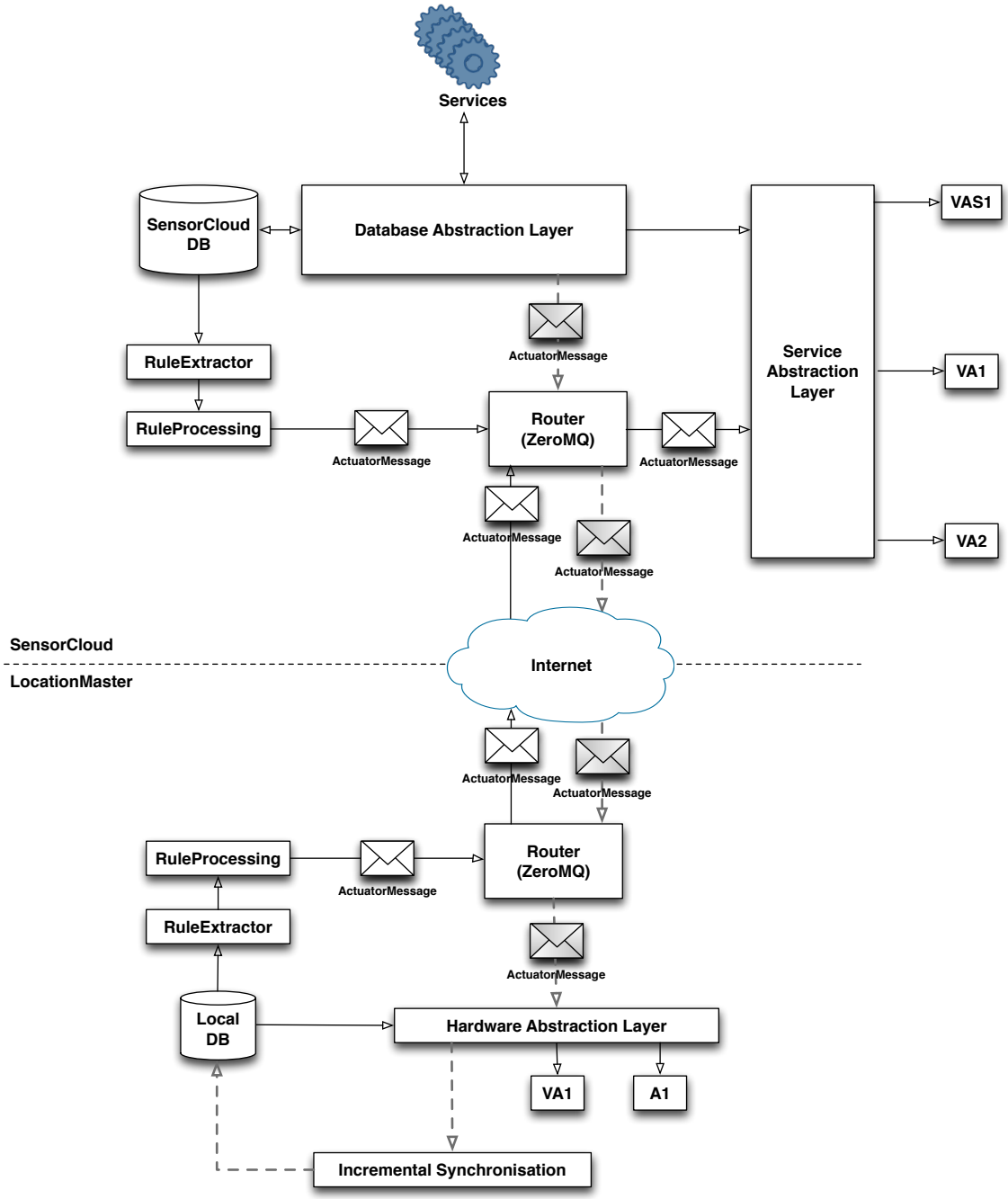

Abbildung 5: Inkrementelle Synchronisation via HAL (LocationMaster)

4.3 SyncomComloss

Bei bestehender Verbindung des LocationMasters zur SensorCloud werden Messwerte kontinuierlich als SensorData über einen XMPP-Client in Richtung SensorCloud versendet. Das Modul „SyncomComloss“ (Abbildung 4) implementiert die Cloud-Synchronisierung von Messwerten und synchronisiert die bei einem Kommunikationsausfall zwischenzeitlich auf dem LocationMaster angefallenen Messwerte mit der Cloud-Datenbank. Im Falle eines Kommunikationsausfalls beschreibt Abbildung 6 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).

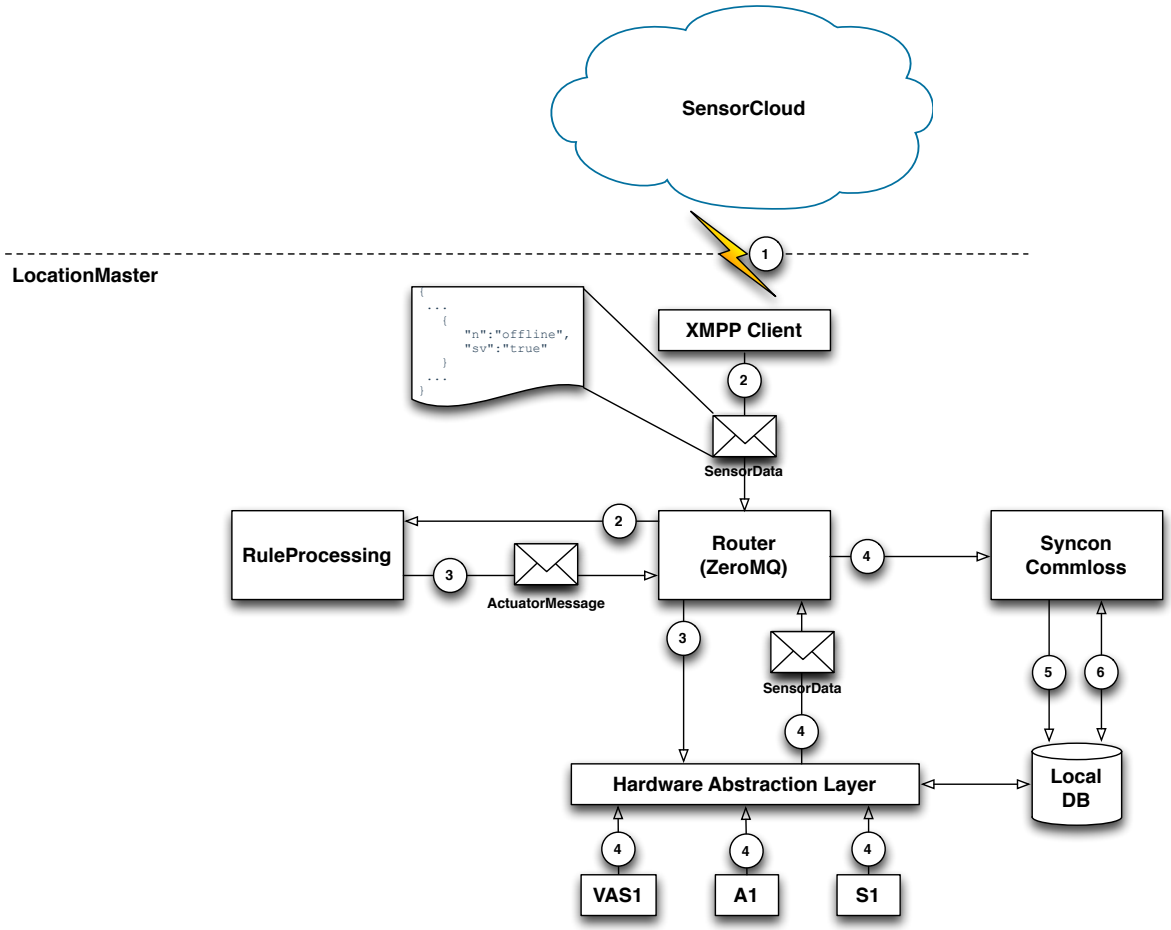

Abbildung 6: Nachrichtenfluss bei Kommunikationsausfall

Abbildung 7 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).

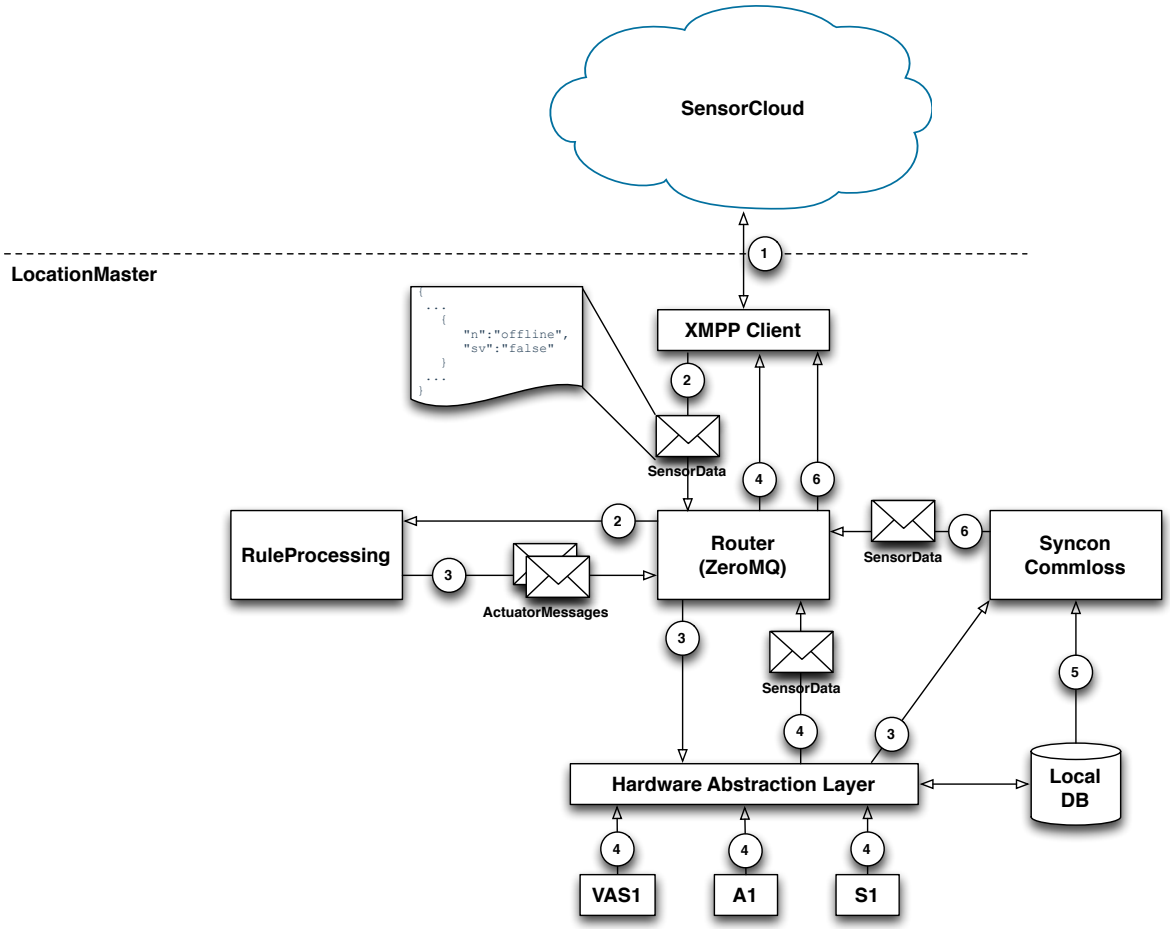

Abbildung 7: Nachrichtenfluss nach Verbindungsaufbau

4.4 Schema Monitor

Das Modul „Schema Monitor“ (Abbildung 4) überwacht Veränderungen im konzeptionellen Schema und der Schemata auf Gateway und Cloud Seite und verteilt Schemata-Änderungen an die Datenbanken des föderierten Datenbanksystems [DBAP5,PRAX-AK]. Der Schema Monitor arbeitet dabei mit einem zentralen Abbild des konzeptionellen Schema, dem sogenannten *JSON Globalen Schema*.

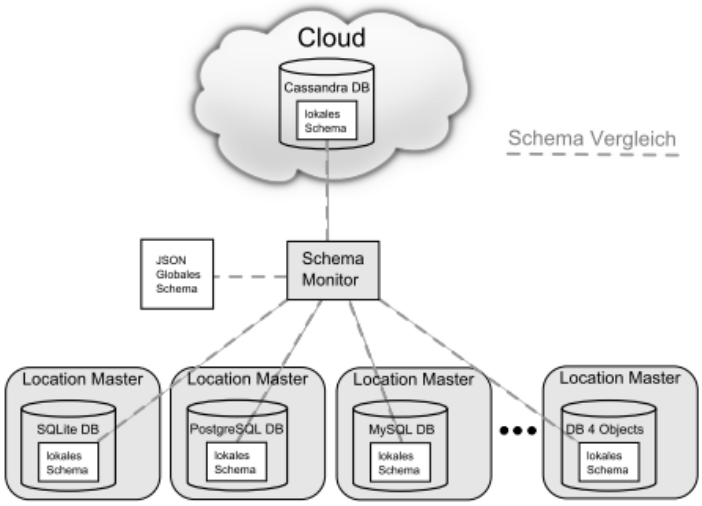

Abbildung 8: Schema Monitor Übersicht

Werden Änderungen an dem JSON Globalen Schema vorgenommen, so werden diese durch den Schema Monitor in den lokalen Schemata aller Location Master Datenbanken sowie der Cloud Datenbank übernommen. Zusätzlich können 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, übernommen werden.

4.5 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. 4.1).

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 der JSON globale Schema Eintrag für die Entität Sensor angegeben.

```
{
"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 9: 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.

4.6 Regel- und Konfig-Extraktor

Das Modul „Regel- und Konfig-Extraktor“ (Abbildung 4) implementiert 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 das 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 10: 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]

4.7 Sender/Empfänger (Cloud)

Das Modul „Sender/Empfänger (Cloud)“ mit integriertem Dispatcher empfängt eingehende SensorData-Nachrichten und verteilt ausgehende ActuatorMessages (Abbildung 11) an die LocationMaster des Gesamtsystems. Eingehende SensorData-Nachrichten werden über das XMPP-Protokoll von den LocationMastern und des cloudseitigen Service Abstraction Layers versendet und von dem Sender/Empfänger (Cloud)-Modul empfangen. Dieses persistiert die empfangenden Messwerte in der Clouddatenbank und übergibt die empfangenden SensorData-Nachrichten an das Cloud-Regelwerk.

Eingehende ActuatorMessages (über den SAL und des vorgelagerten Cloud-Regelwerk empfangen) verteilt die integrierte Dispatcher-Komponente anhand des enthaltenden Basenames (bn-Feld in der AcutatorMessage) (siehe Kapitel 5) an den entsprechenden LocationMaster. Dieser Mechanismus wird wie in Kapitel 5.2.2 beschrieben bei der Nutzung des cloudbasierten Regelwerks verwendet.

In einer späteren Ausbaustufe ist der Bezug von cloudbasierten Sensorwerten durch die LocationMaster angedacht, um z.B. Wetterdaten aus der Cloud zu abonnieren und mit diesen lokale Regeln auf einem LocationMaster auszuführen. Hierzu muss der Dispatcher um eine Publish-/Subscriber-Komponente erweitert werden.

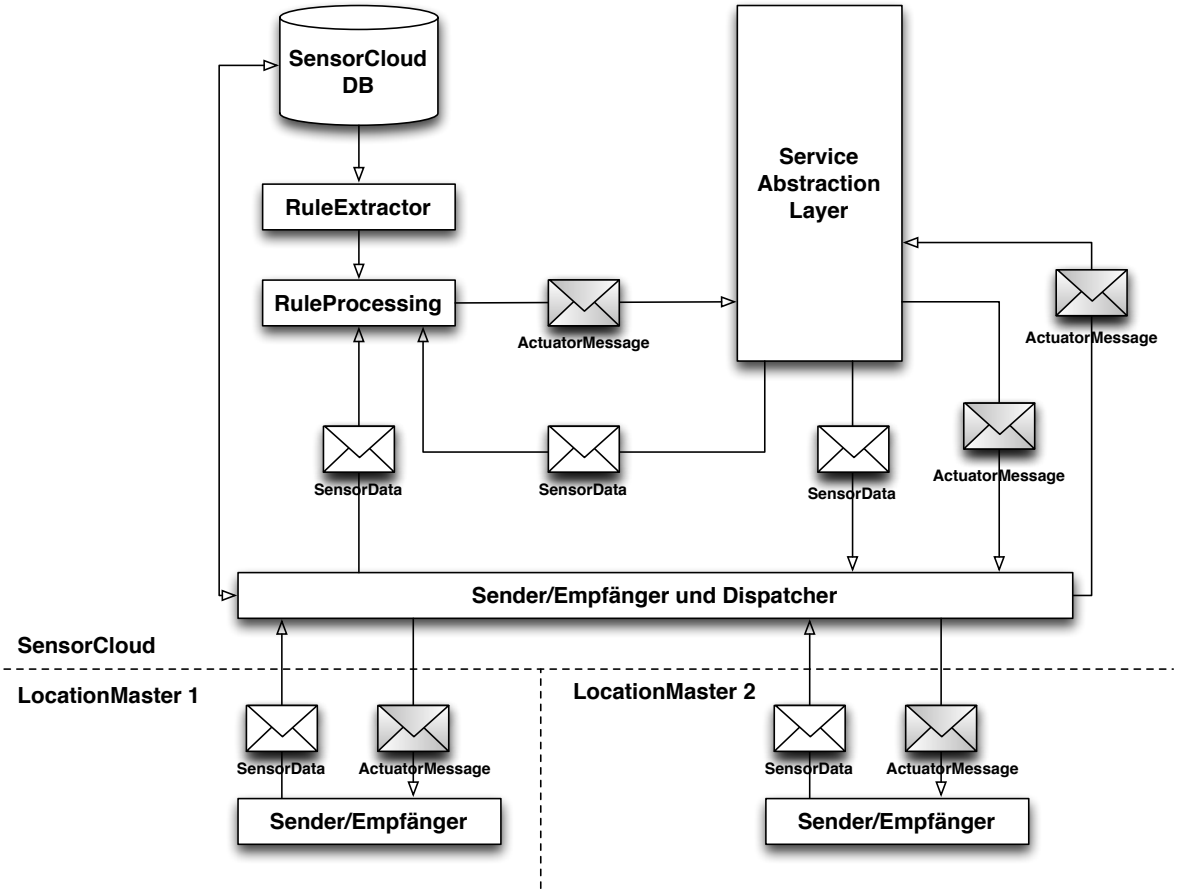

Abbildung 11: Nachrichtenfluss über Sender/Empfänger (Cloud) mit integriertem Dispatcher

5. Aktorsteuerung

Aktoren werden wie in Kapitel 3 beschrieben über ActuatorMessages (Nachrichtentyp 4) gesteuert (Listing 1). Eine Steuerung kann lokal über das Regelwerk erfolgen oder über eine entfernte Steuerung aus der Cloud. Der Nachrichtentyp 4 identifiziert den absendenden Service über das src-Feld der Nachricht und den zu steuernden Aktor über das bn-Feld. Das dst-Feld enthält den LocationMaster (Gateway) an welchem der zu steuernde Aktor angeschlossen ist.

```
{
      "typ":"4", /* type of the message payload */
      "src":"string", /* unique service ID */
      "dst":"string", /* unique gateway ID */
      "bn":"string", /* actuator device ID */
      "seq":"string", /* Sequence Number (optional) */
      "fn":"string", /* Function Name (optional) */
      "e":[{
            "n":"string", /* key ID */
            "sv":"string" /* value for key ID */
      }]
}
```

Listing 1: ActuatorMessage (Quelle: JSON_Darstellung.pdf)

Für eine asynchrone Kommunikation mit Rückantwort müssen Empfänger (Aktor) und Absender (Service)

eindeutig identifizierbar sein. Im Nachrichtentyp 4 (ActuatorMessage) ist dies mit den bn- und src-Feldern sichergestellt. Bietet ein Aktor verschiedene Funktionalitäten kann über das optionale fn-Feld die auszulösende Funktion spezifiziert werden.

Wird vom Initiator der ActuatorMessage eine Rückantwort erwartet, muss das optionale seq-Feld (Sequenznummer) in der ActuatorMessage gesetzt sein. Die Sequenznummer wird in die Antwort-Nachricht vom Nachrichtentyp 5 (Listing 2) übernommen, damit der auslösende Service die asynchrone Antwort einem Benutzer/Ereignis zuordnen kann. Die Rückantwort wird dann über die n- und sv-Felder des Nachrichtentyps versendet.

```
{
	"typ":"5", /* type of the message payload */
	"src":"string", /* unique gateway ID */
	"dst":"string", /* unique service ID */
	"bn":"string", /* actuator device ID */
	"seq":"string", /* Sequence Number */
	"fn":"string", /* Function Name (optional) */
	"e":[{
		"n":"string", /* name */
		"sv":"string" /* value as string */
	}]
}
```

Listing 2: ActuatorResponse (Quelle: JSON_Darstellung.pdf)

Der HAL dient als Interface zu der angeschlossenen Hardware. Er empfängt eingehende ActuatorMessages und übergibt diese an Funktionsbausteine, die je nach Verbindungstyp die Kommunikation mit der Hardware realisieren.

Die Funktionsbausteine des HALs übersetzen dann den Funktionsnamen des kontrollierten Vokabulars in eine herstellerspezifische Kommunikationsart und einen herstellerspezifischen Funktionsnamen und führen die Funktion des adressierten Aktors mit den Parametern aus dem Payload der ActuatorMessage aus.

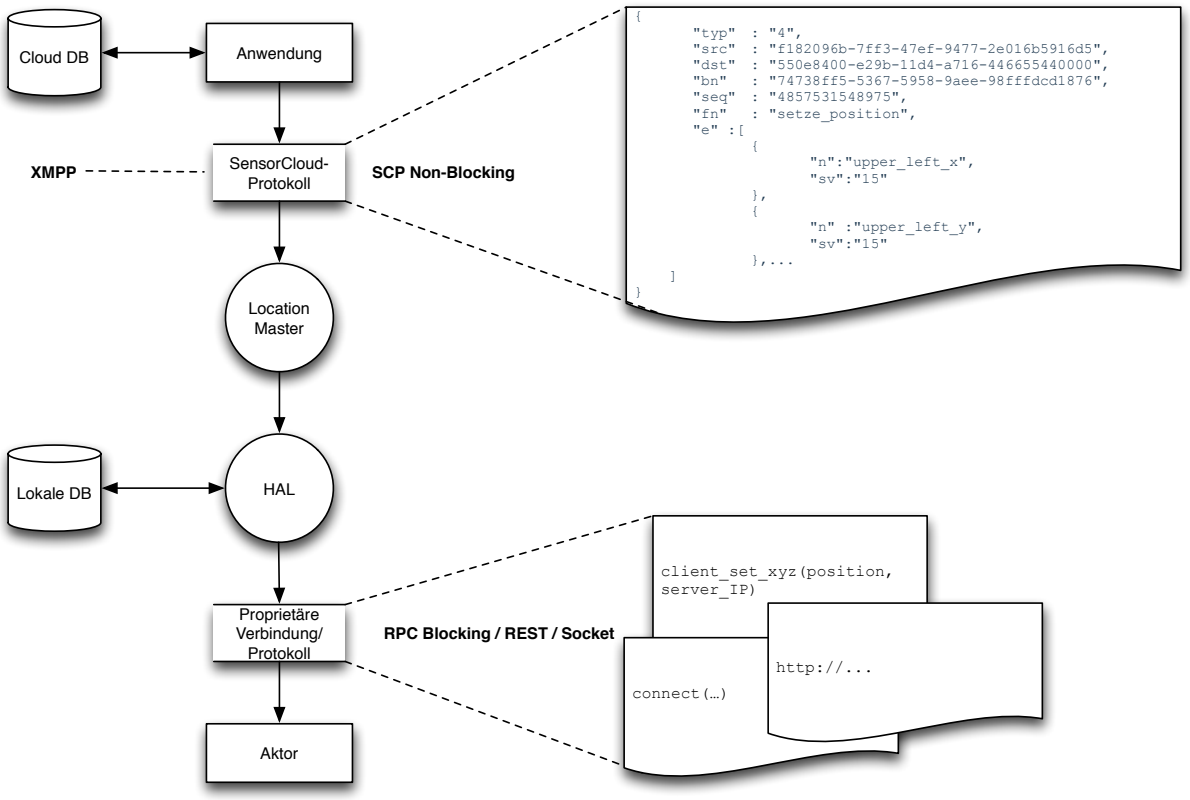

Abbildung 12: Datenflussdiagramm SensorCloud Richtung LocationMaster

Die Rückgabe der aufgerufenen proprietären Funktion wird durch den HAL über eine Nachricht vom Nachrichtentyp 5 an den ursprünglichen Absender gesendet. Der Absender ist durch die src-Angabe aus der eingegangenen ActuatorMessage bekannt und identifiziert zum Beispiel eine Cloud-Anwendung, einen RESTful Service oder einen WebDAV-Server. Über die Sequenznummer kann der Absender die Antwort der ursprünglichen Aktion zuordnen.

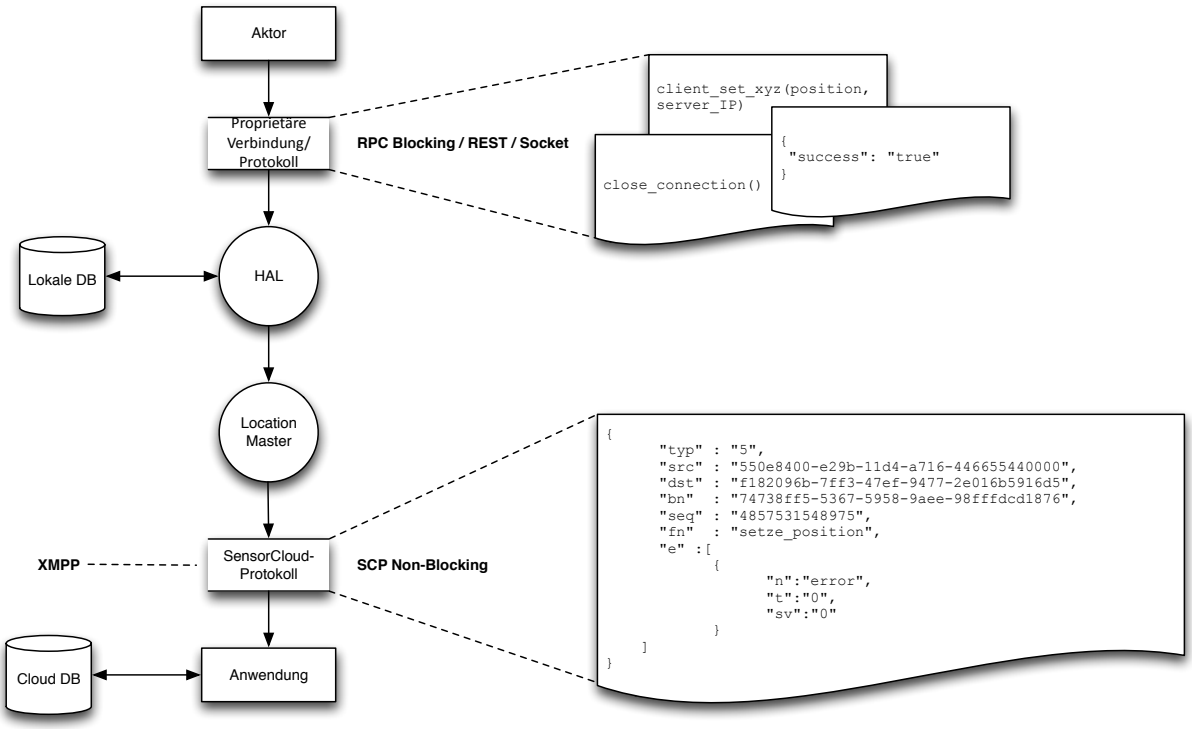

Abbildung 13: Datenflussdiagramm LocationMaster Richtung SensorCloud

5.1 Direkte Aktorsteuerung

Eine direkte Aktorsteuerung durch SensorCloud-Dienste oder lokale Dienste über den SAL, bzw. DAL mit ActuatorMessages, erfordert eine Implementierung einer Sicherheitsschicht in dem Service Abstraction Layer, um nicht autorisierte Aktorsteuerungen zu unterbinden.

Eine direkte Aktorsteuerung gleicht einer Steuerung aus einer Anwendung heraus (z. B. die Steuerung eines Aktors über eine Smartphone-App). Diese Art der Steuerung ist bei aktuell erhältlichen Heimautomatisierungssystemen sehr verbreitet. Sie ist nicht regelbasiert, was einen geringeren Funktionsumfang gegenüber der regelbasierten Steuerung der SensorCloud mit sich bringt. Die regelbasierte Steuerung der SensorCloud wird im folgenden Kapitel beschrieben.

5.2 Regelbasierte Aktorsteuerung

In [DBAP3] ist die Steuerung von Aktoren innerhalb der SensorCloud über ein Regelwerk beschrieben, 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.

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 }
```

Listing 3: Regelaufbau für eine regelbasierte Aktorsteuerung

5.2.1 Lokales Regelwerk

Messwerte von Sensoren fließen über den Hardware Abstraction Layer über SensorData-Nachrichten des SensorCloud-Protokolls in das lokale Regelwerk, wo diese ausgewertet werden. Zutreffende Regeln erzeugen zur Steuerung von Aktoren ActuatorMessages. Hierbei enthält das lokale Regelwerk ausschließlich Regeln mit nur lokal angebundenen Sensoren und Aktoren.

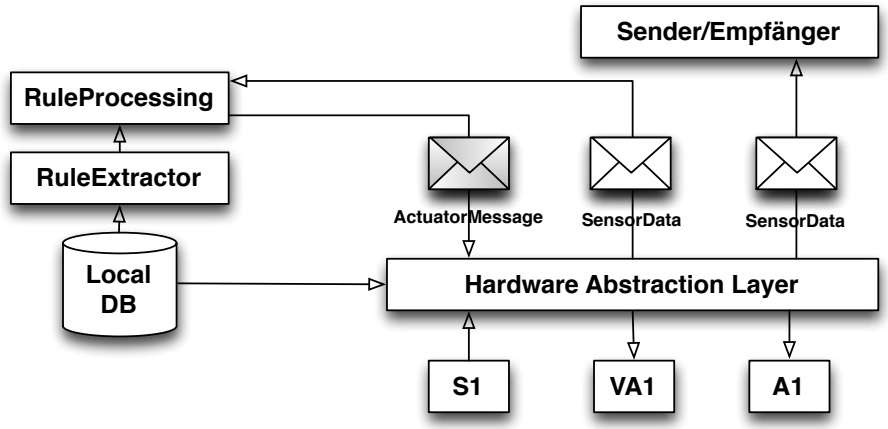

Abbildung 14: Aktorsteuerung über lokales Regelwerk

5.2.2 Globales Regelwerk

Messwerte werden über den Hardware Abstraction Layer in SensorData-Nachrichten zusammen mit dem Versand in das lokale Regelwerk in Richtung Cloud-Knoten der SensorCloud versendet. Die in der Cloud eintreffenden Messwerte werden in den Cloud-Datenbanken persistiert und zudem in das Cloud-Regelwerk

eingeleitet. Das Cloud-Regelwerk enthält LocationMaster übergreifende Regeln für z.B. eine Aktorsteuerung von an anderen LocationMastern angeschlossenen Sensoren. Wenn die Wirkung einer Regel durch einen Messwert ausgelöst wird, versendet das Cloud-Regelwerk analog der LocationMaster-Regelwerke eine ActuatorMessage an den Service Abstraction Layer (bei LocationMastern an den Hardware Abstraction Layer). Handelt es sich bei dem zu steuernden Aktor um keinen lokalen (virtuellen) Aktor, wird die ActuatorMessage über den Dispatcher an den LocationMaster mit dem angeschlossenen Aktor weitergeleitet. Der empfangende LocationMaster leitet die ActuatorMessage an den lokalen Hardware Abstraction Layer, der den entsprechenden Aktorbefehl ausführt.

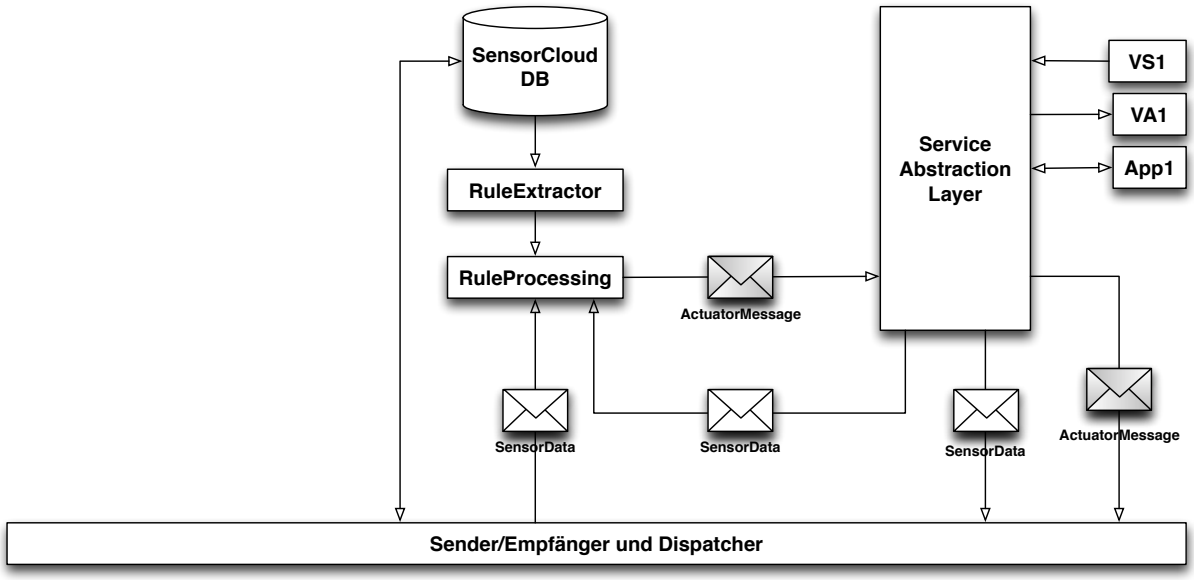

Abbildung 15: Aktorsteuerung über Cloud-Regelwerk

6. SensorCloud-Dienste

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 HAL, bzw. SAL in das Regelwerk einbinden (siehe Abbildung 15). Hierbei kann sowohl auf Nachrichten von SensorCloud-Diensten im Regelwerk reagiert werden, als auch SensorCloud-Dienste über das Regelwerk ausgelöst werden.

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 (Abbildung 16). Weiterhin sind auch SensorCloud-Dienste für die Aggregierung von Messwerten oder der Lösung von linearen Optimierungsaufgaben denkbar. Diese Dienste empfangen und senden SensorData-Nachrichten.

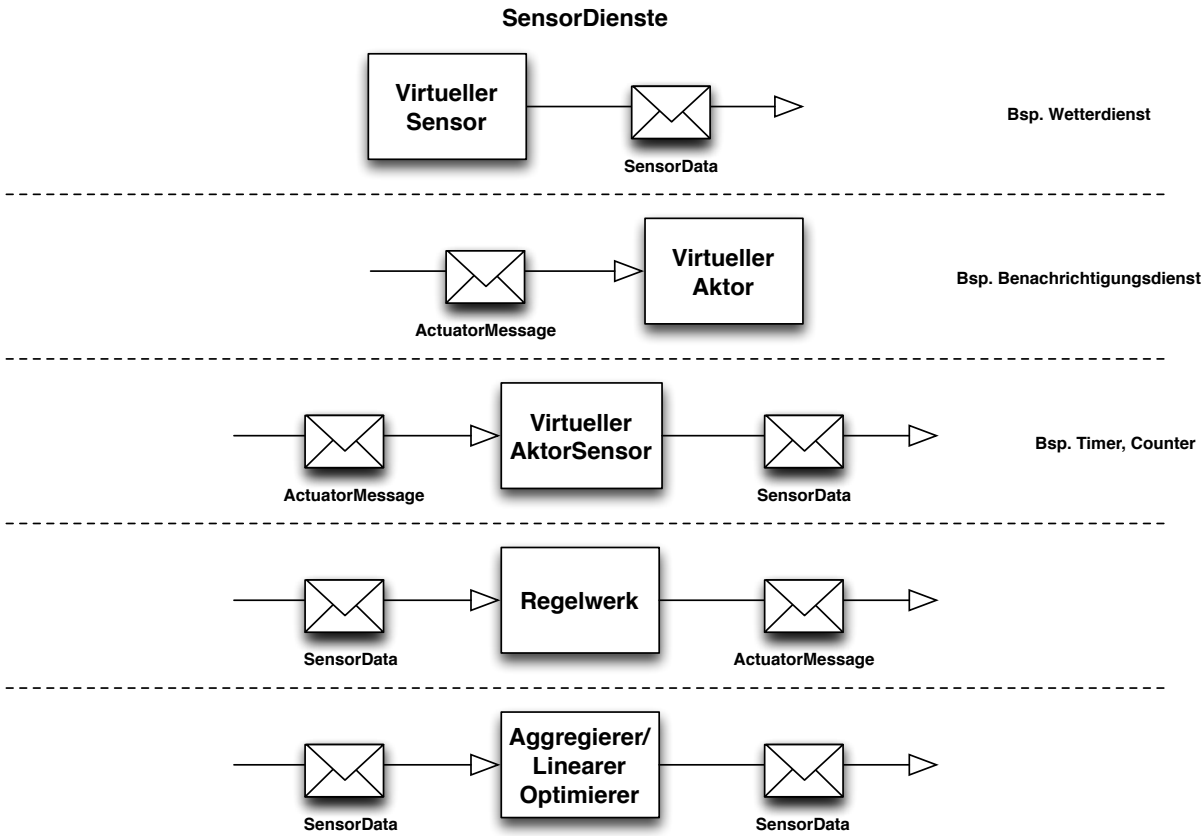

Abbildung 16: SensorCloud-Dienste in der Übersicht

Für die Anbindung von SensorCloud-Diensten an den SAL, bzw. bei einer lokalen Anbindung über den HAL ist eine RESTful Schnittstelle in beide Schichten implementiert. Über die Schnittstelle können Dienste SensorData- und ActuatorData-Nachrichten in das System einleiten. Letzteres dient der in Kapitel 5.1 beschriebenen direkten Aktorsteuerung.

6.1 Anbindung virtueller Sensoren

SensorData wird an den in Listing 4 aufgeführten URI über einen PUT-Request an den SAL/HAL übermittelt (bei der Anbindung an den HAL wird ein lokaler Hostname in der URI verwendet). Die Source ID in Listing 4 entspricht der in der Entität Sensor angebenden Source ID (Attribut SenSouID) des virtuellen Sensors. Da ein virtueller Sensor sich in das System wie ein realer Sensor einbindet, müssen alle notwendigen Entitäten des virtuellen Sensors in den SensorCloud-Datenbanken hinterlegt sein.

```
http://babeauf.nt.fh-koeln.de:81/SAL/SourceID/<SourceID>
```

Listing 4: URI für die Anbindung von virtuellen Sensoren

SensorData wird im einfachsten Fall im JSON-Format über ein Array von (n, t, sv)-Tupel eingeliefert. Die Übergabe kann auch in einem anderen Format an den SAL erfolgen (z.B. einfacher String mit mehreren hexadezimalen kommaseparierten Messwerten). Dies erfordert dann analog zu der Anbindung von realen Sensoren das Schreiben eines extra Treibers im SAL/HAL.

```
[{
      "n" : "Temperatur",
      "t" : "1418813375000",
      "sv": "20"
}, ...]
```

Listing 5: (n, t, sv)-Tupel für virtuelle Sensoren

Der SAL/HAL versendet den empfangenen Messwert über eine SensorData-Nachricht (Listing 6) gemäß des

Nachrichtenflusses aus Abbildung 3 in das Gesamtsystem.

```
{
	"typ":"1", /* type of the message payload */
	"src":"string", /* unique gateway ID */
	"bn":"string", /* sensor device ID */
	"bt":"number", /* base time of sensor readings */
	"e":[{
		"n":"string", /* sensor ID */
		"t":"number", /* optional time value */
		"sv":"string" /* sensed value as string */
	}]
}
```

Listing 6: SensorData (Quelle: JSON_Darstellung.pdf)

6.2 Direkter Versand von ActuatorMessages

Für einen wie in Kapitel 5.1 diskutierten direkten Versand von ActuatorMessages wird diese über einen PUT-Request an den in Listing 7 aufgeführten URI an den SAL/HAL übermittelt (bei der Anbindung an den HAL wird ein lokaler Hostname in der URI verwendet). Die Actuator ID in Listing 7 entspricht der in der Entität Aktor angebenden Actuator ID (Attribut AktID).

```
http://babeauf.nt.fh-koeln.de:81/SAL/ActuatorID/<ActuatorID>
```

Listing 7: URI für die Anbindung von virtuellen Sensoren

ActuatorCommands werden im JSON-Format eingeliefert (Listing 8).

```
{
	"fn":"string", /* Function Name (optional) */
	"seq":"string", /* Sequence Number (optional) */
	"src":"string", /* unique service ID */
	"e":[{
		"n":"string", /* key ID */
		"sv":"string" /* value for key ID */
	},...]
}
```

Listing 8: ActuatorCommand bei Anbindung an den SAL/HAL

Der SAL/HAL versendet das empfangene ActuatorCommand über eine ActuatorMessage-Nachricht (Listing 1) gemäß des Nachrichtenflusses aus Abbildung 3 in das Gesamtsystem.

Die in Kapitel 5 beschriebene asynchrone Rückantwort bei der Aktorsteuerung kann bei einem direkten Versand von ActuatorMessages über einen Push- oder Pull-Mechanismus erreicht werden. Beide Mechanismen sind in den SAL zu implementieren. Ein Pull-Mechanismus kann über einen durch den SAL bereitgestellten RESTful-Service implementiert werden, den der ursprüngliche Service periodisch abfragt. Bei einem Push-Mechanismus muss der Service einen Server über eine Kommunikationsart (z.B. Socket, REST, RPC, ZeroMQ) bereitstellen, über den der SAL die von den LocationMastern eintreffenden Rückantworten der ursprünglichen ActuatorMessages dem Service übergeben kann. Hierzu ist, wie für die in dem folgenden Abschnitt beschriebene Anbindung von virtuellen Aktoren, eine Treiber-Implementierung für den SAL notwendig.

6.3 Anbindung virtueller Aktoren

Virtuelle Aktoren werden analog physikalischer Aktoren an das Gesamtsystem angebunden. Ein Aktor bietet eine unterschiedliche Menge an Funktionen, die die physikalischen Aktivitäten eines Aktors beschreiben. In [DBAP3] wird eine AktorSemantik eingeführt, die Aktorfunktionen und deren Übergabeparameter beschreibt. Die eingeführte AktorSemantik wird auch für virtuelle Aktoren verwendet. Dementsprechend wird ein virtueller Aktor analog eines physikalischen Aktors in den Entitätstypen des in [DBAP4] beschriebenen konzeptionellen Schemas angelegt.

Jeder virtuelle Aktor kann, wie in Kapitel 5 für physikalische Aktoren beschrieben, über eine unterschiedliche Kommunikationsart gesteuert werden (RPC, REST, Socket). Die Kommunikationsart ist in den Entitäten des virtuellen Aktors abgespeichert. Für jede verwendete Kommunikationsart muss als Gegenstück ein Treiber des SALs implementiert werden. Jede Kommunikationsart muss nur als ein Treiber im SAL implementiert sein. Zum Vergleich des Treiberkonzepts sei auf die Dokumentation der Forschungsgruppe Testbett verwiesen.

6.4 Authentifizierung an dem Service Abstraction Layer

Eine Authentifizierung an den Diensten des Service Abstraction Layers (SAL) kann analog der in [DBAP8] beschriebenen DAL-Authentifizierung implementiert werden.

Services der SensorCloud-Serviceplattform verbinden sich zum SAL und prüfen über dessen Serverzertifikat die Authentizität des Endpunktes (1). Nach dem Aufbau eines über Transport Layer Security (TLS) gesicherten Kommunikationskanals authentifizieren sich die Services anhand ihrer Zertifikate an dem SAL (2). Der SAL prüft das Zertifikat der Gegenstelle und die Zugriffsrechte (zur Verwaltung der Zugriffsrechte von Nutzern und Gruppen vgl. [DBAP9]) auf den aufgerufenen Dienst des SALs. Im Erfolgsfall erhält die Gegenstelle Zugriff auf den Dienst (3). Nur authentifizierte Services können über den mittels TLS gesicherten Kommunikationskanal die für sie zugelassenen Dienste des SALs verwenden (4). Die Dienste des SALs prüfen über das verwendete Zertifikat (die Identität der Gegenstelle) die Zugriffsrechte auf die angeforderten Daten. Dadurch wird ein Zugriff auf Daten fremder LocationMaster und Services unterbunden. Die Gegenstelle erhält eine entsprechende Mitteilung (5).

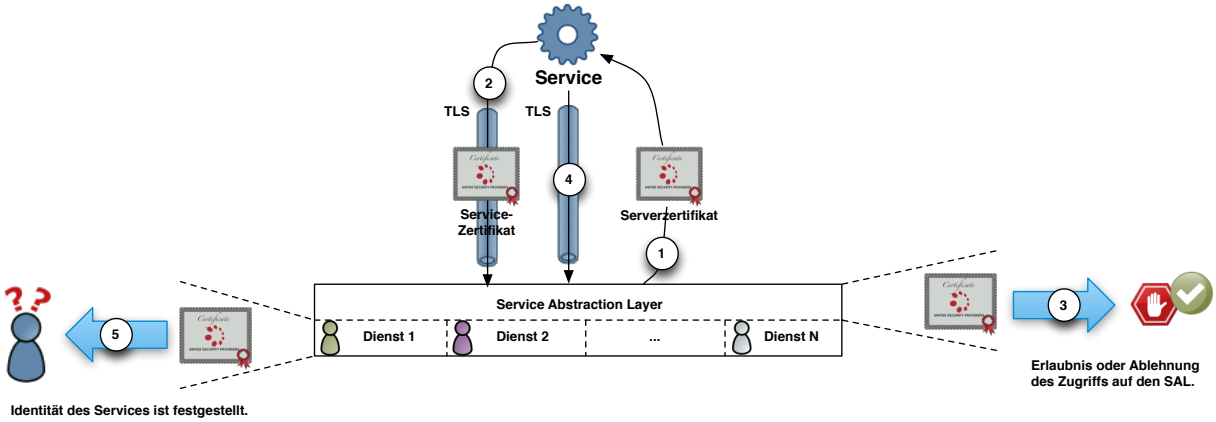

Abbildung 17: Authentifizierung über Zertifikate

6.5 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 (SensorData) in das Cloud-Regelwerk gesendet, um z. B. entfernte Aktoren von Fenstern eines Hauses zu steuern.

```
{
	"typ": "1",
	"src": "0d7f4e18-d2cf-4ae5-b34c-5ed57474a061",
	"bn":  "015dacf0-93ef-11e3-baa8-0800200c9a66",
	"bt":  "1418813375000",
	"e": [
		{
			"n": "Orkan",
			"t": "0",
			"sv": "true"
		}, …
	]
}
```

Listing 9: 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).

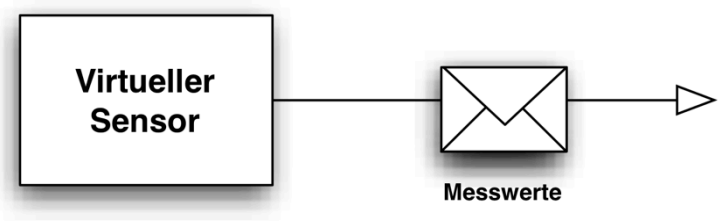

Abbildung 18: Virtueller Sensor

6.6 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 (ActuatorMessages).

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",
	"dst" : "fe249d66-ff5d-421a-87f9-79308a1bbba3",
	"fn"  : "sendeEmail",
	"e" :[
		{
			"n":"Email",
			"sv":"empfanger@domain.tld"
```

```
            },
            {
                  "n" :"Nachricht",
                  "sv":"Die Fenster im Gebäude 13 wurden geschlossen."
            },...
    ]
}
```

Listing 10: ActuatorMessage eines Benachrichtigungsdienstes

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

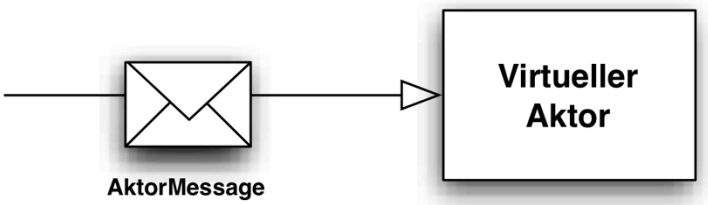

Abbildung 19: Virtueller Aktor

6.7 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 20: 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 [DBAP3] im Datenmodell wie folgt realisieren:

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

Tabelle 2: Entität SensorEvent

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.

EveMitID	EveMitEveID	EveMitSenEveID	EveMitRei
103	2030	4711	1
104	2031	4712	1
105	2031	4713	2

Tabelle 3: Entität EventMitglieder

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.

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

Tabelle 4: Entität Event

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

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",	1

						"sv":"9999" }	

8005	2031	Licht	A1	Aktor	LichtAusFunktion	{
						"n":"Licht",
						"sv":"ausschalten"
						}
							1

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

Tabelle 5: Entität EventAktion

In der Entität 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.

Quellen

[BA-SW] S. Wieben: „*Wetterberichte für die SensorCloud*“, Februar 2013,

http://bscw.fh-koeln.de/bscw/bscw.cgi/d3951089/BachelorThesis.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
[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

[DBAP14] T. Büchel et al.: „*Abschlussbericht der FDBS Gruppe FH Köln im Projekt SensorCloud“,* Fachhochschule Köln, Köln, 2015

[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).

[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

[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

[PRACAS] Bradberry, R., & Lubow, E. (2014). *Practical Cassandra.* Upper Saddle River, NJ, USA: Pearson Education.

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

[XENVIR] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., et al. (2003). Xen and the Art of Virtualization. *Proceedings 19th ACM Symposium Operating Systems Principles* (S. 164-177). ACM Press.