PArADISE S&A-Level

Felix Wächter, Martin Haufschild

Gliederung

Aufgaben
Teilnehmer
HMC, Appliance, Sensor
Anmerkungen und Festlegungen
Protokoll
Abläufe
Anforderungen
Nachrichtenformat
Code
Fragen

Aufgaben

Aufgaben

Anfragen von oben
HMC → Appliance
Umschreiben und Verteilen von Anfragen
Appliance → Sensoren
Prozessieren von Anfragen
Sensoren
Appliance
Zusammenführen von Ergebnissen

Teilnehmer

Teilnehmer - Home Media Center

ortsfest über längeren Zeitraum

im Netzwerk

Verbindung mit Local Server

relativ hohe Leistungsfähigkeit

lange Uptime

<SQL-Standard hier einfügen>

Bsp.:

PC

NUC

Spielekonsole

Teilnehmer - Appliances

mobil

“kommen und gehen”

begrenzte Leistungsfähigkeit

unvorhersehbare Uptime

unterstützen BerkeleyDB/ SQLite, MongoDB, U1DB, …

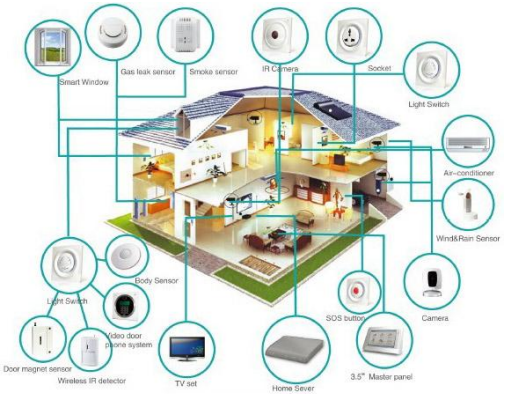
Bsp.:

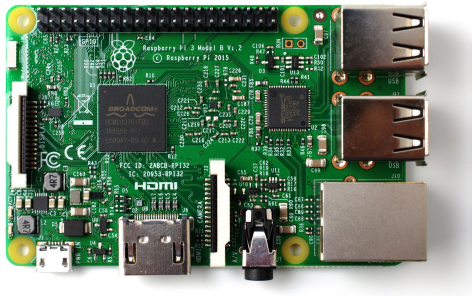
Android/ iOS/ Tizen/ …

Phones, Tablets, Wearables, TVs

teilweise nur dokumentenorientierte DBs (Ubuntu Phone)

Teilnehmer - Sensoren





Geräte zur Datenerfassung
Aktivität über größere Zeiträume
TinyDB (oder gar keine DB)
Bsp.:
Kameras
Rasberry Pi (“DIY-Geräte”)
Heizungssteuerungen
Lichtschalter
Bewegungssensoren
...

Teilnehmer - Anmerkungen

haben vier Hierarchiestufen (LS, HMC, A, S)
weitere Differenzierung auf jeder Stufe notwendig
Bsp.: HMC
NUC → eher stationär
PS4 → wird schon eher aus dem Netzwerk entfernt
Bsp.: Appliances
Android → SQLite
Ubuntu Phone → U1DB (dokumentenorientiert, JSON)

Teilnehmer - Anmerkungen

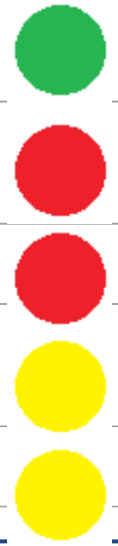
Kriterien für die Einteilung in Geräteklassen:

- Rechenleistung
- verfügbarer SQL-Standard/ DB-Technologie
- Fähigkeiten (z.B. verfügbare Sprachen)
- Vertrauen in Plattform/ Gerät
- Erweiterbarkeit
- Verweildauer im Netzwerk
- durchschnittliche Uptime
- Belastung (durch andere Anwendungen)

Teilnehmer - Anmerkungen

Plattform	Sprachen	Datenbanken
Android 	Java, C/C++	SQLite
iOS 	Objective-C, Swift	SQLite, Core Data
Tizen 	Javascript, C	SQLite, ActiveRecord.js, MiniMongo (kommt mit Blaze Library)
Ubuntu Phone 	HTML5, QML	U1DB
Firefox OS 	Javascript	keine (?) LocalStorage

Teilnehmer - Anmerkungen

Geräteklasse	Rolle	Mobilität im Haus	Verweildauer im Netzwerk	Kontrolle über Daten
Laptop	HMC, Appliance	gering	mittel	
Spielekonsole	HMC, Appliance	gering	lang	
SmartTV	Appliance	gering	lang	
Phone/ Tablet	Appliance, Sensor	hoch	mittel bis kurz	
SmartWatch	Sensor*	hoch	kurz	

* Wearables können u.U. direkt in die DB des Companion-Devices schreiben

Teilnehmer - Festlegungen

Beschränkung auf Einteilung in Sensoren und Appliances

Sensor

TinyDB
Python3
Raspberry Pi

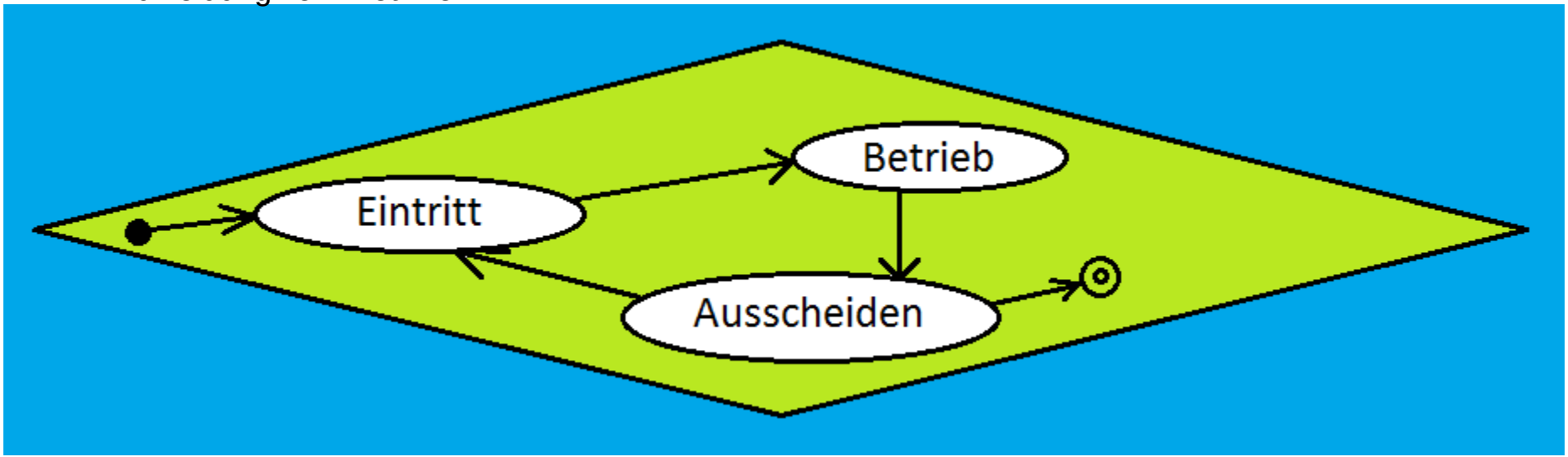
Appliance

BDB, SQLite
Python3, Java
NUC, Laptop oder Smartphone

Teilnehmer - Festlegungen

Einbettung in einfachen Lifecycle

- Anmeldung im Netzwerk
- Betrieb (Anfrageverarbeitung)
- Abmeldung vom Netzwerk



Protokoll

Protokoll - Gliederung

Annahmen
Anforderungen
Logik/ Handlungssequenzen
Protokollspezifikation
Kommentar und Ausblick

Protokoll - Annahmen

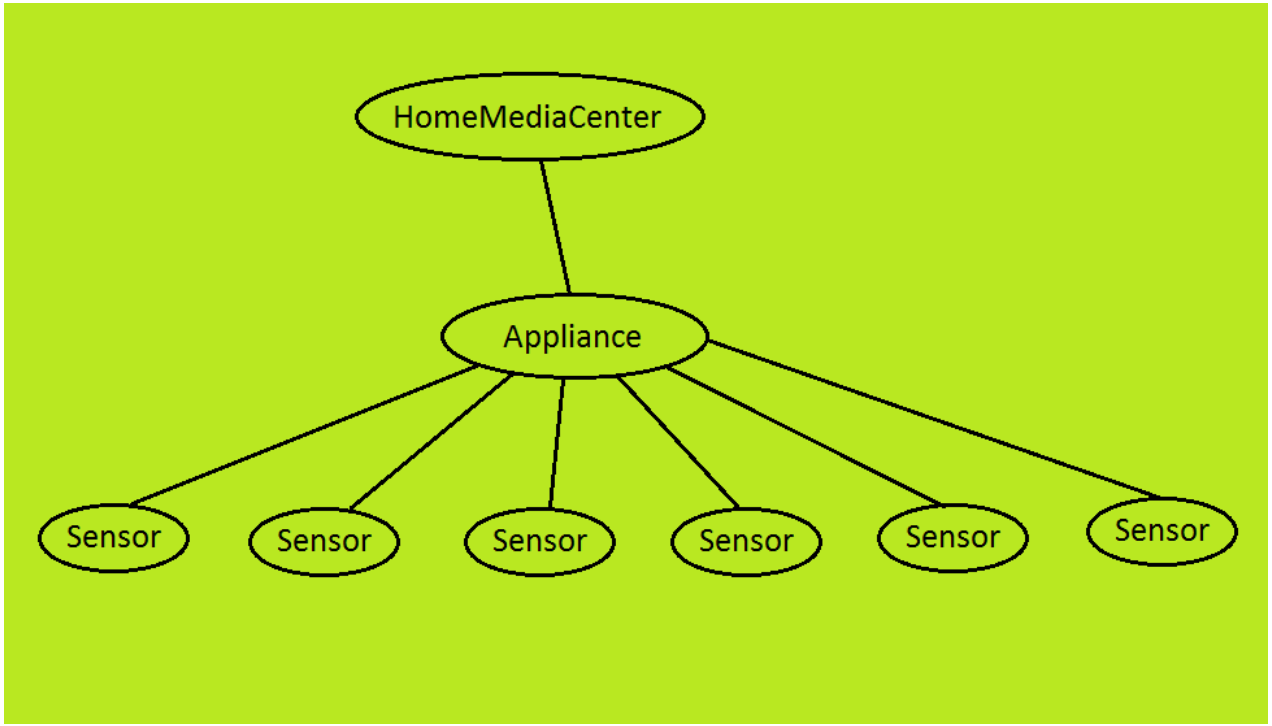
einfacher Lifecycle
Sensor, Appliance, HMC wie bereits beschrieben
bestehendes (TCP/IP) Netz
kein Handover
Geräte verschwinden nicht einfach
→ Sensor und Appliance führen ihr individuelles Setup durch und treten anschließend miteinander in Kontakt. Eine eventuell durchzuführende Discovery-Phase wird übersprungen.

Protokoll - Anforderungen

Sensor-Appliance-Communication (SAC):
Anmeldung des Sensors bei der Appliance
Abmeldung des Sensors von der Appliance
Deassoziierung des Sensors von Seiten der Appliance
Anfrage von Appliance an Sensor
Übermittlung von Sensordaten an die Appliance

Appliance-HMC-Communication (AHC):
De-/ Assoziierung von Appliance und HMC
Query Transmission
Result Set Transmission

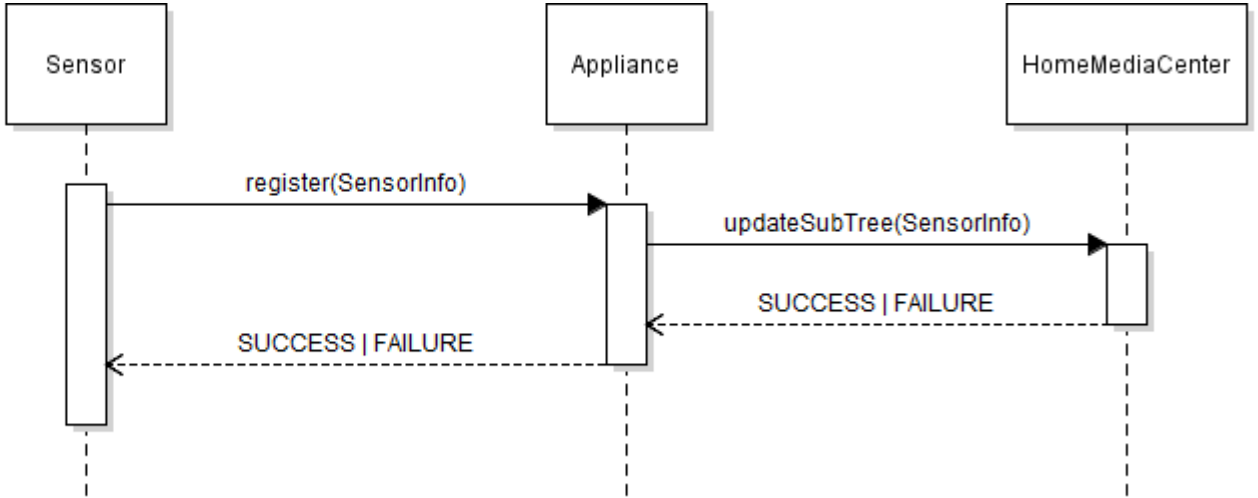
Protokoll - Logik

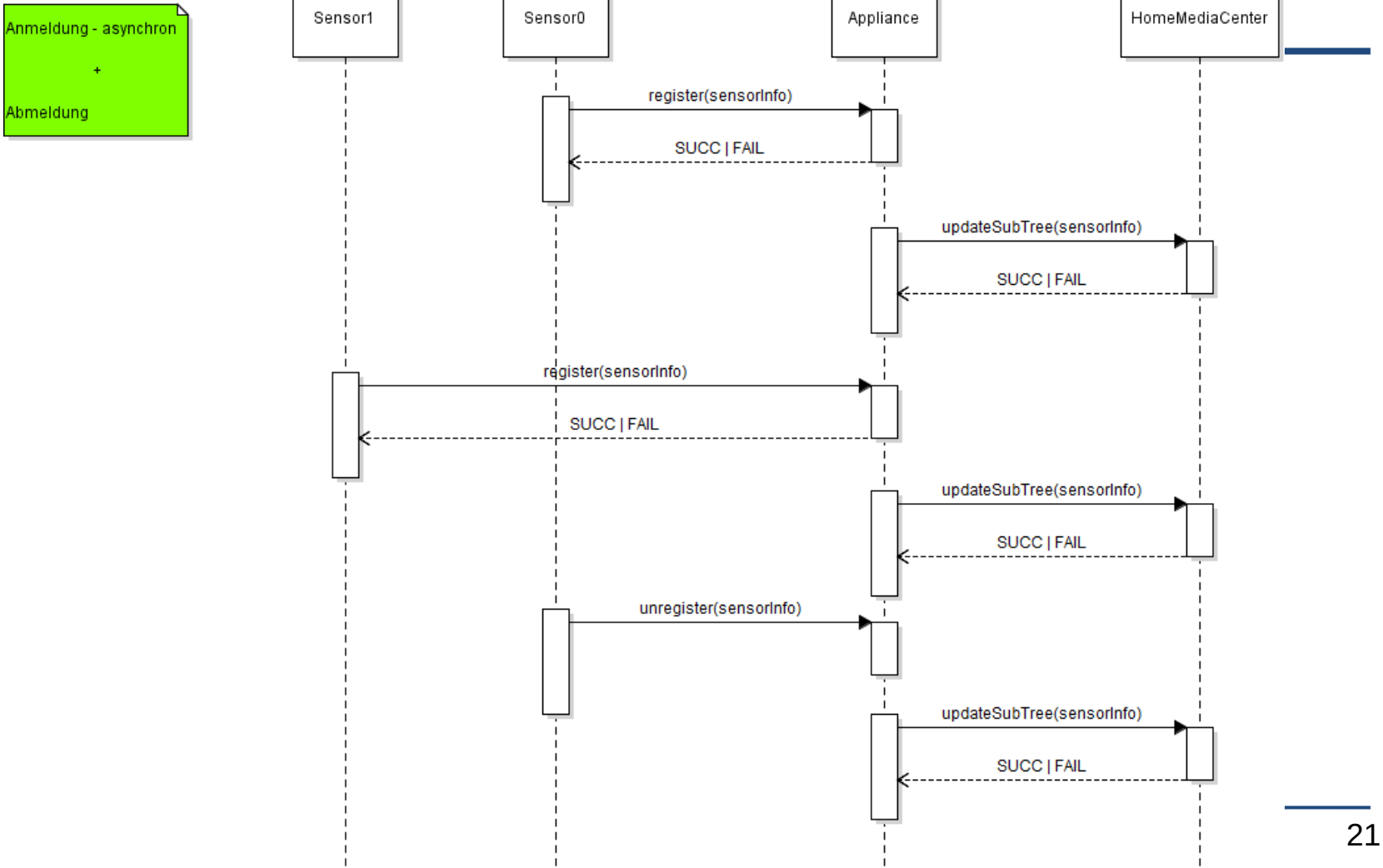


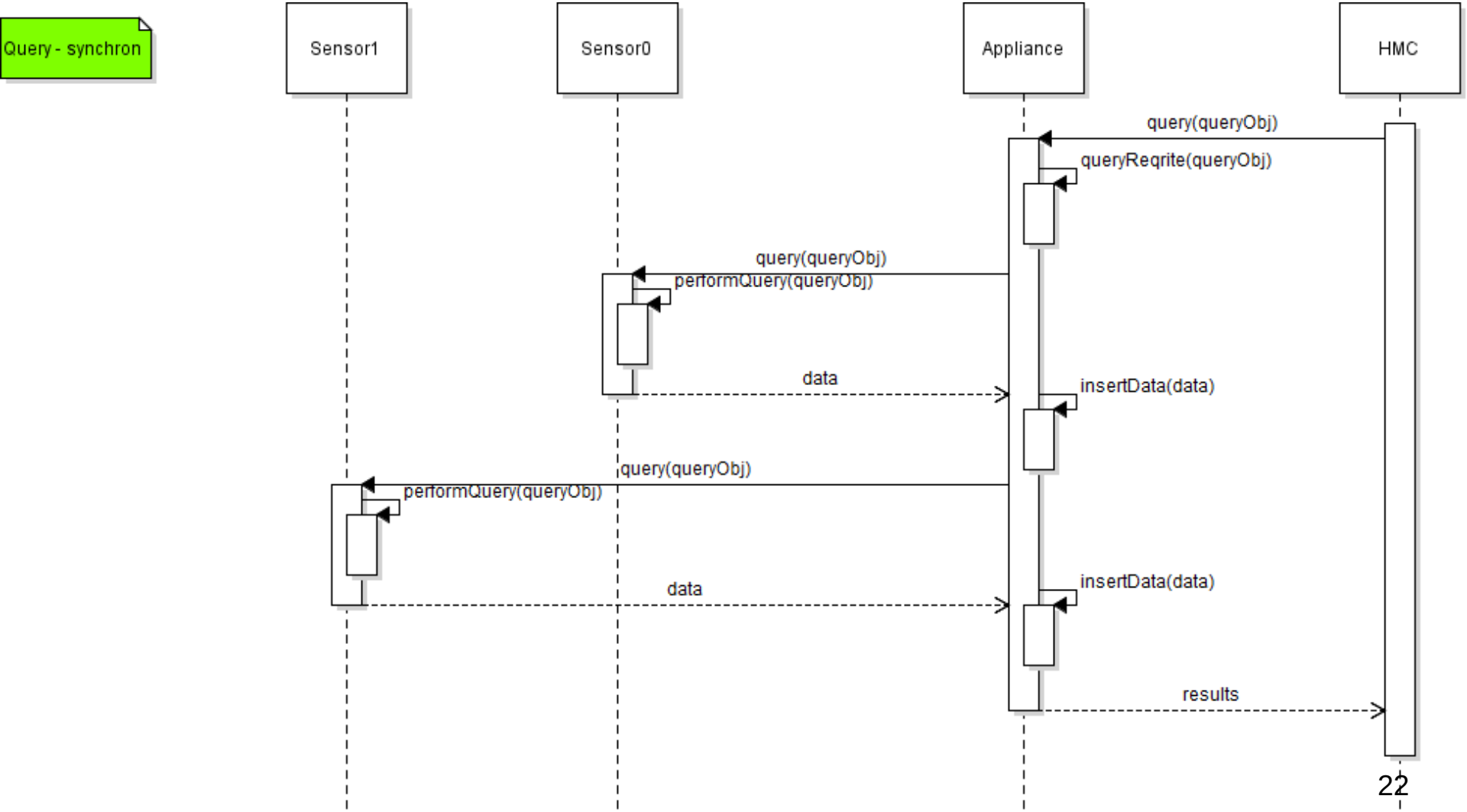
Protokoll - Logik

Sequenzdiagramm zur Visualisierung der Netzaufbauphase

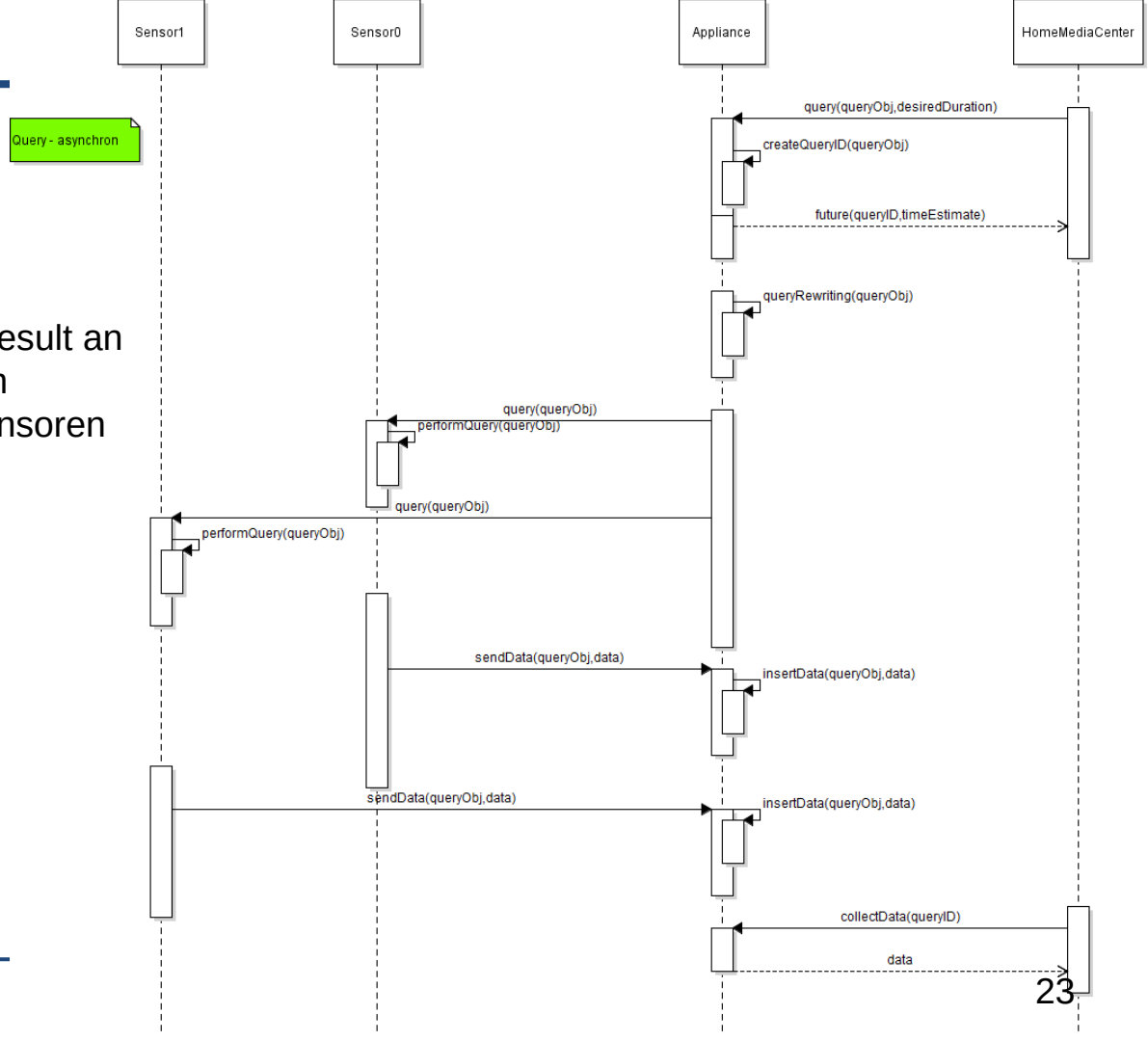
-> alle Objekte im LAN
-> Namesnauflösung bereits erfolgt
-> HMC und LocalServer analog




Anmeldung - asynchron
+
Abmeldung
Sensor1
Sensor0
Appliance
HomeMediaCenter
register(sensorInfo)
SUCC | FAIL
updateSubTree(sensorInfo)
SUCC | FAIL
register(sensorInfo)
SUCC | FAIL
updateSubTree(sensorInfo)
SUCC | FAIL
unregister(sensorInfo)
updateSubTree(sensorInfo)
SUCC | FAIL

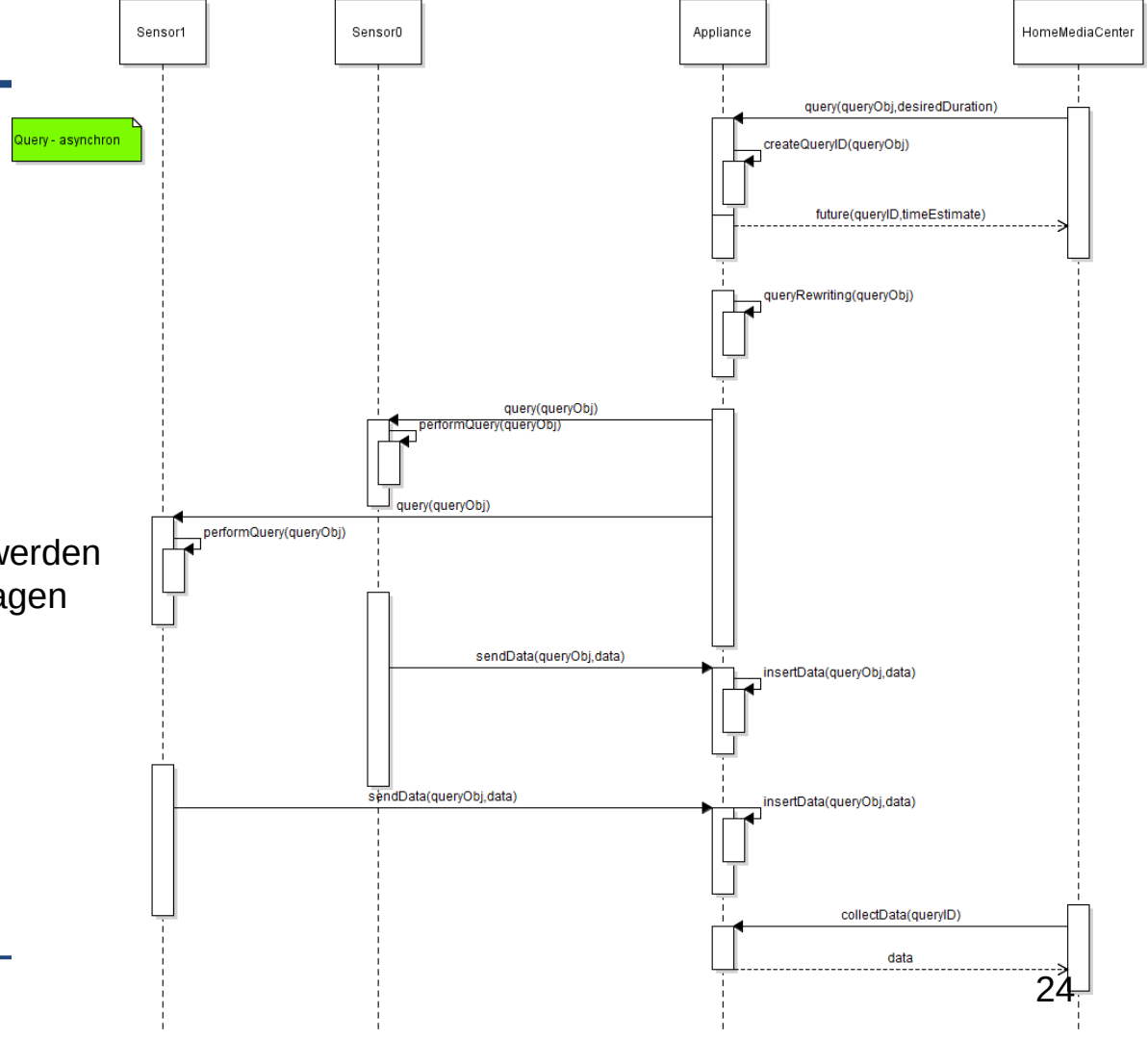

Query - synchron
Sensor1
Sensor0
Appliance
HMC
query(queryObj)
queryReqrite(queryObj)
query(queryObj)
performQuery(queryObj)
data
insertData(data)
query(queryObj)
performQuery(queryObj)
data
insertData(data)
results

1) HMC sendet Query
2) HMC erhält Token
3) Appliance legt Tabelle für QueryResult an
4) Appliance schreibt die Anfrage um
5) Appliance sendet Anfragen an Sensoren
6) Sensoren führen Anfragen durch
7) Sensoren liefern Ergebnisse bei Appliance ab



Begründung:

- HMC blockiert nicht
- Ausfall/ lahmer Sensor wenig problematisch
- weniger Last auf Appliance
- HMC kann auch schon Teile des ResultSets holen
- Daten können schon verarbeitet werden (Appliance, HMC) bevor alle Anfragen durch sind



Protokoll - Spezifikation

Nachrichten im JSON-Format
zweiteilig

1. Nachrichtentyp
2. Daten

Protokoll - Spezifikation

Nachrichtentyp:
beschreibt Vorgang/Aktion
“QUERY”, “REGISTRATION”, “CONFIRMATION”, ...

Daten:
für den jeweiligen Vorgang notwendige Informationen
Anfrage an Sensor → QueryID & Anfrage

Protokoll - Spezifikation

```
/** Registratration
 *
 * Sensor -> Appliance
 */
{
    "MSG_TYPE" : "REGISTRATRION",
    "DATA" : {
                "ID" : "S0",
                "TYPE" : "SENSOR"
                "ATTR" : {
                    "A" : "INTEGER",
                    "B" : "FLOAT",
                    "C" : "STRING"
                },
                "ADDR" : {
                    "HOST" : 0.0.0.1, // filled by appliance
                    "PORT" : 4711
                }
            }
}
```

```
/** Dissociation
 *
 * Sensor -> Appliance
 */
{
    "MSG_TYPE" : "DISSOCIATION",
    "DATA" : {
                "ID" : "S0"
            }
}
```

Protokoll - Spezifikation

```
/** QUERY
 *
 * Appliance -> Sensor
 */
{
    "MSG_TYPE" : "QUERY",
    "DATA" : {
                "QUERY_ID" : "A012-2016-09-34-09-06-01-123",    // Appliance-ID + Timestamp
                "QUERY" : "<insert-valid-tinydb-statement-here",
            }
}
```

Protokoll - Spezifikation

```
/** Query-Response
 *
 * Sensor -> Appliance
 */
{
    "MSG_TYPE" : "QUERY_RESPONSE",
    "DATA" : {
                "QUERY_ID" : "A012-2016-09-34-09-06-01-123",     // Appliance-ID + Timestamp
                "HEADER" : ["INTEGER", "FLOAT"],                 // tuple types (header)
                "RESULT_SET" : [
                                    [213132,2.22],
                                    [655820,2.13],
                                    [646465,2.01],
                                    [646455,2.02],
                                    [845220,2.30],
                                    [796845,1.23]
                                ]
            }
}
```

Protokoll - Spezifikation

```
/** HMC-Query
 *
 * HMC -> Appliance
 */
{
    "MSG_TYPE" : "HMC-QUERY",
    "DATA" : {
                "QUERY" : "<insert-valid-query-here>"
             }
}

/** HMC-Query-Response
 *
 * Appliance -> HMC
 */
{
    "MSG_TYPE" : "HMC_QUERY_RESPONSE",
    "DATA" : {
                "TOKEN" : "Sx546Yl464", // identifies the query on appliance level
                "DURATION" : 6000       // estimated duration in ms
             }
}
```

Protokoll - Spezifikation

```
/** HMC-Data-Collection
 *
 * HMC -> Appliance
 */
{
    "MSG_TYPE" : "HMC_DATA_COLLECTION",
    "DATA" : {
                "TOKEN" : "Sx546Yl464", // identifies the query on appliance level
             }
}
```

Protokoll - Spezifikation

```
/** HMC-Data-Collection-Response
 *
 * Aplliance -> HMC
 */
{
    "MSG_TYPE" : "HMC_DATA_COLL_RESP",
    "DATA" : {
                "TOKEN" : "Sx546Yl464",
                "HEADER" : ["INTEGER" , "FLOAT"],
                "RESULT_SET" : [
                                    [213132,2.22],
                                    [655820,2.13],
                                    [646465,2.01],
                                    [646455,2.02],
                                    [845220,2.30],
                                    [796845,1.23]
                                ]
            }
}
```

Implementierung

SQL-TO-TinyDB-Wrapper

- Sensor-Ebene nur mit TinyDB-DBMS ausgestattet
- TinyDB mit JSON-Format und nur mit Python-Schnittstelle
- Konvertierung der SQL-Anfragen von Appliance-Ebene in die Python-Schnittstelle nötig
- Wrapper deshalb in Python programmiert unter Nutzung von Lamda-Ausdrücken
- Fähigkeiten des Wrappers:
  - nur SELECT-Anfragen für eine Relation verarbeitbar
  - nur Attribut-Wert-Vergleiche im WHERE-Teil durch .search()-Aufruf von TinyDB. Auch kein WHERE-Teil möglich.
  - beliebig tiefe Klammerungen und logische Verknüpfungen (and, or, not) im WHERE-Teil
  - Projektionen auf beliebige und beliebig viele Attribute (ohne Rel.) durch map. Auch Wildcard (*) möglich.
  - Bsp.: SELECT a, b FROM tabelle WHERE a<2 or (a>5 and b==4)

SQL-TO-TinyDB-Wrapper

- Wrapper erweiterbar um
  - Attribut-Attribut-Vergleiche durch filter
  - Aggregationen (SUM, AVG), MIN, MAX durch map und reduce
  - Projektionen auf Operationen von Attributen, z.B. SELECT 2*Col_A FROM Tabelle

- Wrapper nutzt sqlparse-Modul von Python und traversiert dann dessen Syntaxbaum

- Wrapper-Aufruf:
  - execfile(SQLToTinyDB.py) (existiert nur in Python2)
  - SQLToTinyDB(<SQL_String>) gibt String für TinyDB-Tabelle zurück
  - eval(SQLToTinyDB(<SQL_String>)) führt TinyDB-Anfrage aus
- Ergebnis des Wrapper-Aufrufs im JSON-Format. Noch Konvertierung zu ResultSet für JDBC-Treiber nötig.

Fragen ?