Virtual Power Plant design document

Ubbe Welling
ubbe@eng.au.dk

October 21, 2015

Contents

1 Considerations and requirements **2**
1.1 Overview 2
1.2 Users and privileges 2
1.2.1 User categories 2
1.2.2 Privileges requirements 3

2 Design **4**
2.1 Database design 4
2.1.1 Schema: core 4
2.1.2 Schema: Users, groups and privileges 7
2.1.3 Subject to change 8
2.1.4 Rolling window 8
2.1.5 Data warehouse 8
2.2 Application design 8

Chapter 1

Considerations and requirements

1.1 Overview

The purpose of the Virtual Power Plant (VPP) platform is to:

- receive measurements from sensors deployed in a building.
- receive current and forecast information from external sources on grid load, electricity price and more.
- process measurements and external information to make decisions on power usage.
- actuate devices deployed in a building to implement above mentioned decisions.

The above functionality is already implemented in a number of individual applications and scripts. This system should provide the same functionality in an integrated application.

1.2 Users and privileges

1.2.1 User categories

An initial listing of envisioned users and their access to the system is shown below:

Residents

- Actuate in own home
- See data from own home

Janitor

- Should be able to do anything he/she can already do before the system
- Actuate anything not in private
- Add/remove actuators

Administrative staff (ie. housing association office staff)

- See data on some level of aggregation. Maybe just reports?

Aggregator

- multiple buildings
- specific read/actuate permissions, granted by janitor/admin. staff

System administrator

- full access

1.2.2 Privileges requirements

With outset in the above user categories and tasks, the following distinct privileges have been identified:

Global system privileges

- Access to reports
- User access
- System admin access

Building-specific privileges

- Add new devices
- Remove devices

Device-specific privileges

- Read device status and data (measurements/actions)
- Configure device (configure parameters, disable)
- Remove device and data
- Actuate controller

Chapter 2

Design

The platform will consist of one main server application with an attached database.

2.1 Database design

The database will reside in a PostgreSQL DBMS.

2.1.1 Schema: core

The central part of the database schema is shown in figure 2.1 and explained below:

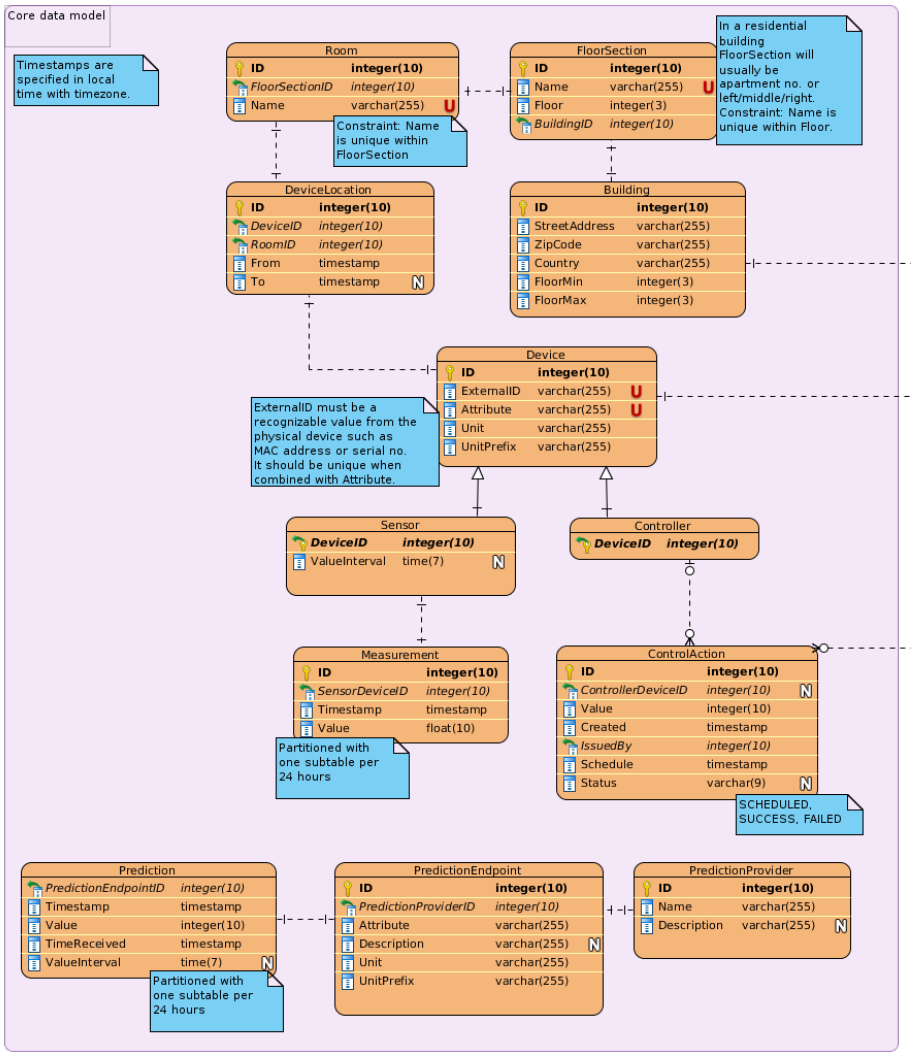

Figure 2.1: Database core schema

Devices and measurements

Device Entries in `Device` correspond to physical control and sensor devices, with the modification that we store one logical device for each function of the physical device. Whether a device is a sensor or a controller is specified by its presence in table `Controller` or `Sensor`. Columns `Attribute`,`Unit` and `UnitPrefix` (such as milli) are for sensors specification of the incoming measurement values, while they for controllers specify the format of values to send to the controller when actuating it.

Sensor Table `Sensor` specifies an optional property `ValueInterval` which is used when values are aggregated over a limited time interval (such as 15 minutes).

Measurement This table will contain a row for each measurement received from a physical sensor. It will simply consist of a `Value`, a `Timestamp` and a reference into `Sensor`, which enables interpretation of the value. The `Measurement` table is expected to grow very large and will therefore be partitioned into subtables that will each contain 24 hours of measurements and can be discarded on the fly according to the rolling window strategy explained in section 2.1.4.

ControlAction Table `ControlAction` will contain scheduled and past commands for controllers. An action is simply specified by a `Value` which can be interpreted via the reference into table `Controller` and `Device`. Column `Schedule` specifies the time for carrying out the action, and `Status` indicates if execution is still pending or has been completed. Finally, `IssuedBy` specifies which user scheduled the action. We might consider partitioning and discarding of old data in this table in the same way as for table `Measurement`.

Building, FloorSection, Room The physical properties of a building are modeled in these tables. A building consists of an integer range of floors. Each floor consists of `FloorSection`s which in most cases will be equivalent to apartments. The generalized term FloorSection is intended to support other types of buildings where designations such as "South wing" or other may be desired. Finally, a floor consists of named `Room`s. We do not expect to obtain device locations with a higher degree of accuracy than individual rooms.

DeviceLocation This table maps `Device`s to `Room`s for specified time periods, indicating that devices may be moved around.

Predictions

Predictions of a wide range of values (power consumption, grid load, price, CO_2 emissions, ...) will be received from external data providers and will in addition be generated by our own application logic.

While the data stored for predictions is quite similar to those for measurements, we have chosen to store them separately because of the fundamentally different semantics.

PredictionProvider An entity providing predictions, such as energinet.dk or the system itself.

PredictionEndpoint A logical source of predictions of one type. Specifies the `Attribute`,`Unit` and `UnitPrefix` of the incoming values and provides an optional `Description`.

Prediction Actual prediction values. `Timestamp` indicates the time for which the value applies, while `TimeReceived` indicated when the prediction was received from the provider. This is relevant since multiple predictions for the same future point in time may be received over time. Some values actually cover an interval (for instance predicted power consumption for a given day of 24 hours), which is specified in column `ValueInterval`. This table is also expected to grow quickly, motivating the same partitioning and rolling window strategy as for table `Measurement`.

2.1.2 Schema: Users, groups and privileges

The database schema for storing users and privileges has been developed to meet the requirements from section 1.2.2. This is shown below:

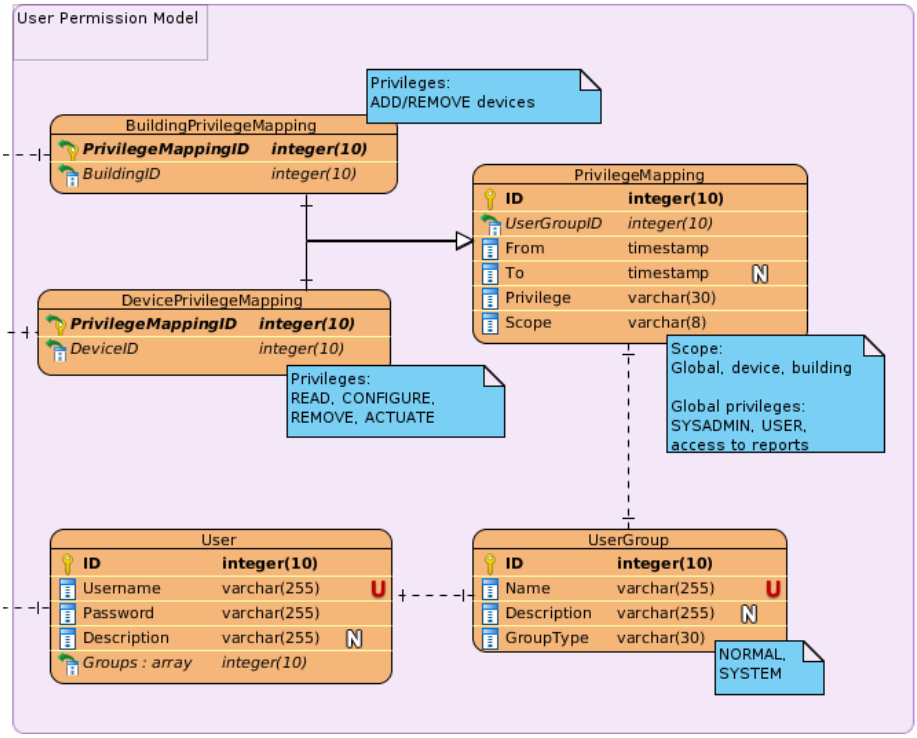

Figure 2.2: Users and privilege schema

Users and UserGroups `Users` have a username and a password (which will be hashed with a suitable one-way hash function) and can be member of a number of `UserGroups`.

PrivilegeMapping This table maps `UserGroup`s to specific privileges. A mapping includes a time period which may be open-ended. In order to have privileges for specific devices and buildings as well as globally valid ones, a `Scope` must be set. In case of a device or building-specific privilege, the concerned `Device` or `Building` must be looked up in `DevicePrivilegeMapping` or `BuildingPrivilegeMapping` respectively. These tables are intended to have corresponding subtypes in the object-oriented implementation.

It will be up to application logic to interpret the semantics of the concrete privileges.

2.1.3 Subject to change

We can already foresee that configuration of controllers as well as other settings that must be persisted will most likely require extensions to the schema proposed above. The basic structure of the core schema should however not change.

2.1.4 Rolling window

Since the `Measurement` table will grow very quickly, a the partitioning and data discarding scheme will be employed. The table will be partitioned in time intervald, having one subtable for every 24 hours. Furthermore, subtables older than one week will be dropped. The time limits can naturally be configured. The same scheme might be applied to tables `ControlAction` and `Prediction`. This is done in order to accommodate the VPP server on a desktop size machine with limited disk space.

2.1.5 Data warehouse

In order to retain data, the VPP will periodically forward data to an external database (data warehouse) that can accommodate a larger volume of data for longer periods. When forwarding data, measurements may be averaged over limited time intervals to reduce data size. The data warehouse can then be used for statistics and historical analysis. While the data warehouse schema was initially planned to be identical to the VPP rolling window DB, the presence of users, privileges and most likely various other configuration indicates that probably only the core schema as shown in figure 2.1 should be present in the data warehouse.

2.2 Application design

The application will be programmed in object-oriented Python, using Python processes to enable concurrent processing.

Key classes have been identified to form a static structure which is shown in figure 2.3 and elaborated below:

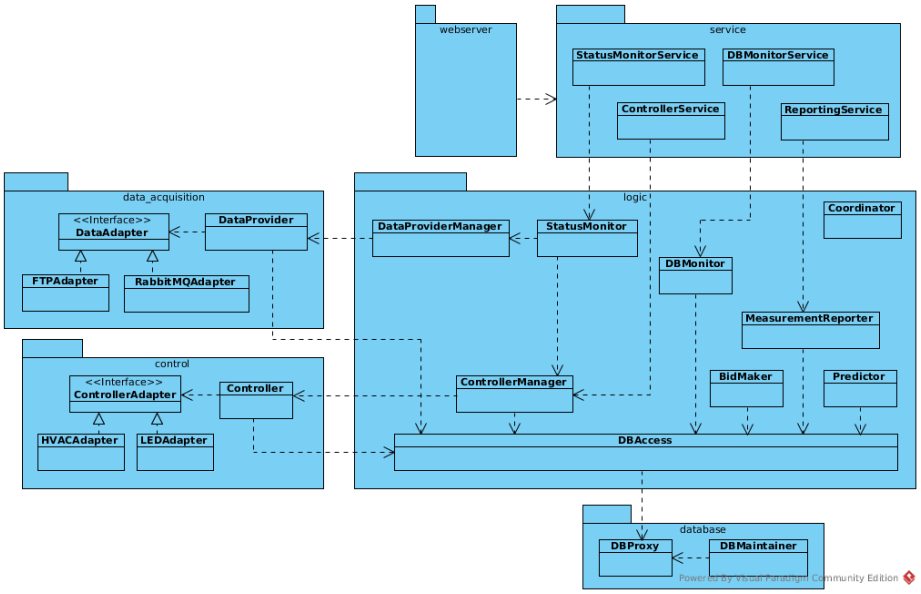

Figure 2.3: Class diagram

The runtime creation of various threads is shown below:

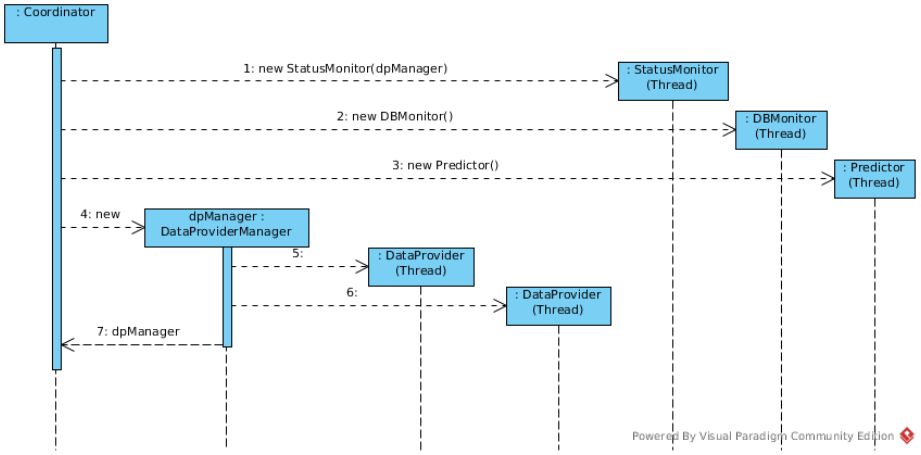

Figure 2.4: Sequence diagram of thread creation