SDD-VA-03

Virtual Apartment Team

IoT Platform for Senior Homecare

Software Design Document
Phase 3

Kilian Bosse, Marina Aoki

Date: 2023/10/23

1 INTRODUCTION

1.1 Purpose

This software design document (SDD) builds on the specifications set out by the Virtual Apartment Team during Phases 1 and 2 of the IoT Lab Course at the Chair of Computer Architecture and Parallel Systems at the Technical University of Munich. Now, in Phase 3, the teams are tasked with developing a minimum viable product of their respective subsystems. This SDD will detail how we expect our solution will work and the metrics used to evaluate it. For more details on the Virtual Apartment's overarching goals and design ideas for the entire project, see SDD-VA-01.

1.2 Scope

The MVP we develop during Phase 3 of this project is not expected to have the full polish but most of the functionality of the final product. Instead, we aim to provide a somewhat "bare-bones" implementation of the system that can showcase almost the full functionality even if lacking polish.

1.3 Overview

In this document, we will attempt to specify the features we wish to include in this MVP version of the Virtual Apartment and concretise its system architecture. To this end, we will first start by providing a general system overview in Section 2. In Section 3, we will describe and explain the architectural design of the Virtual Apartment along with the decomposition of its subsystems. The user interface for the system developed in Phase 2 will be specified in Section 4. Next, in Section 5, we will provide a description of the non-functional requirements of the Virtual Apartment. A proposed timeline and a risk assessment for the second phase of this Lab Course can be found in the Appendix.

2 SYSTEM OVERVIEW

2.1 Includes & Excluded functionality

Since Phase 3 delivers a Minimal Viable Product (MVP) and not the full version, we need to separate which functionality will be included versus what will be delayed for the final product.

2.1.0 Functionality present in the prototype

- Reading CSV files, separating the data per sensor
- Serving separate sensors via their MQTT topics
- Rudimentary message reformatting

2.1.1 MVP Included Functionality

- Reading Scenario and Sensor configs from files allowing for easy and repeatable scenario execution as well as easy editing for anyone
- Hardcoded reformatting on a per-sensor-type basis
- Graphical User Interface that allows for
    - Selecting scenarios, sensors
    - Selecting the data source folder
- Readme file that explains the usage to the other teams

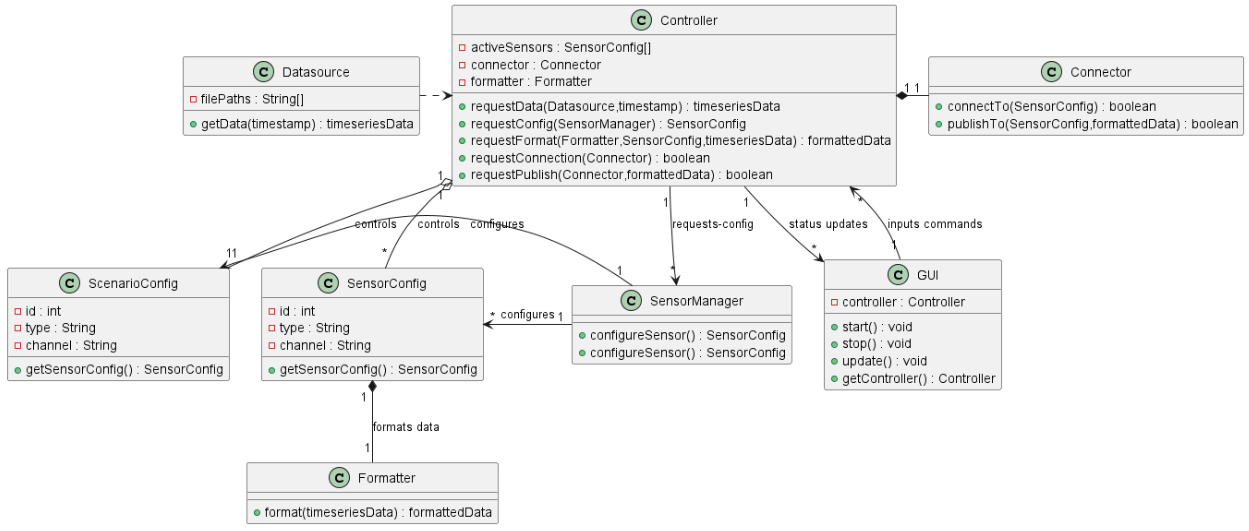
2.1.2 Not necessarily in MVP included Functionality

- in-GUI Sensor & Scenario config manipulation
- Timestamp offsets / manipulation
- Regex-based dynamic formatting for the data per sensor
- CSV-file preview
- Selecting part of the timeseries dynamically
- Multi-csv scenarios (if deemed helpful by the digital twin team)
- Multithreading the CSV file access & reformatting

3 SYSTEM ARCHITECTURE

3.1 Architectural Design

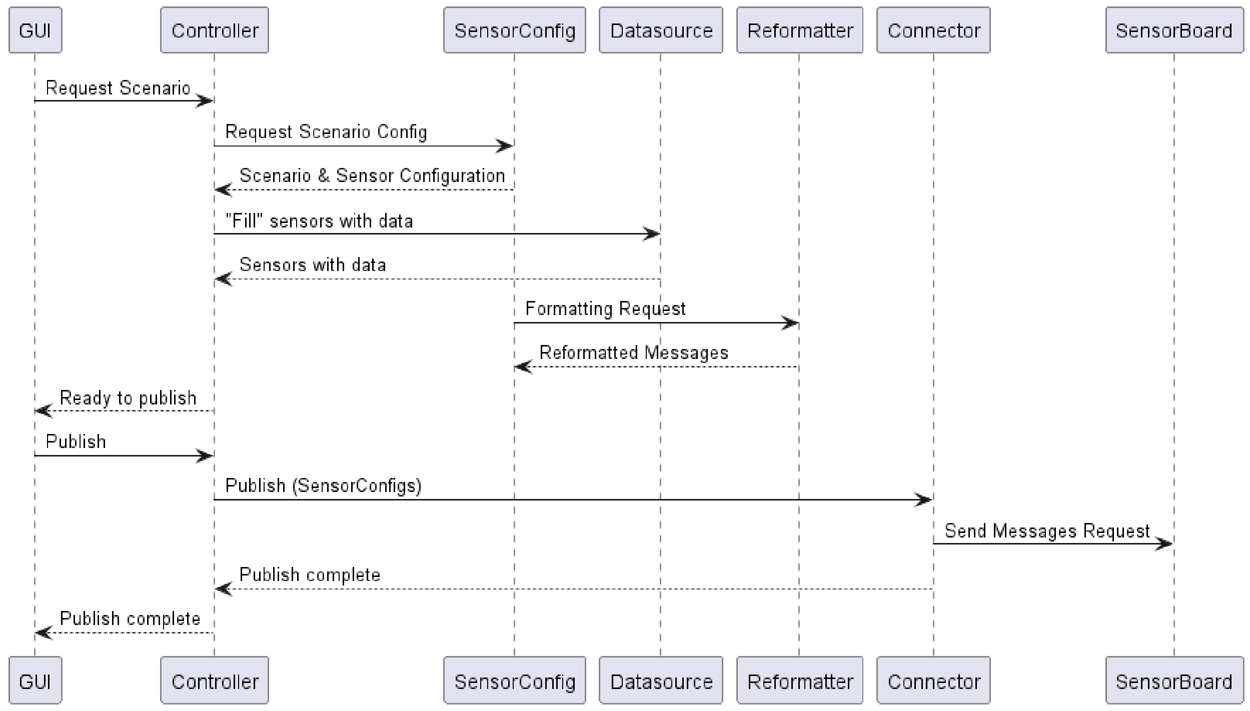
The Virtual Apartment architecture will consist of six main components and will communicate with the rest of the monitoring system platform through the temperature sensor board provided by the Sensor Team. The six main components are the Datasource, the Controller, the Connector, the SensorManager, the SensorConfig and the Reformatter. The GUI will be written with the JavaFX framework and will be superimposed on the architecture below without changing much of the technical workflow.



Controller
activeSensors : SensorConfig[]
connector : Connector
formatter : Formatter
requestData(Datasource,timestamp) : timeseriesData
requestConfig(SensorManager) : SensorConfig
requestFormat(Formatter,SensorConfig,timeseriesData) : formattedData
requestConnection(Connector) : boolean
requestPublish(Connector,formattedData) : boolean
Datasource
filePaths : String[]
getData(timestamp) : timeseriesData
Connector
connectTo(SensorConfig) : boolean
publishTo(SensorConfig,formattedData) : boolean
1 1
ScenarioConfig
id : int
type : String
channel : String
getSensorConfig() : SensorConfig
SensorConfig
id : int
type : String
channel : String
getSensorConfig() : SensorConfig
SensorManager
configureSensor() : SensorConfig
configureSensor() : SensorConfig
GUI
controller : Controller
start() : void
stop() : void
update() : void
getController() : Controller
Formatter
format(timeseriesData) : formattedData
controls
controls
configures
requests-config
status updates
inputs commands
configures
formats data
11
1
1
*
1
1
*
1
*
*
1
*
1
1
1

The Controller comprises the core of the system and administers the timing and execution of methods in the other components. In particular, it requests the configuration of the required scenario (and thereby sensors) from the ConfigurationManager.The ConfigManager reads Scenario- and Sensor config files to properly configure the requested scenario.
Then each sensor gets it’s data from the Datasource, which is passed the selected CSV file.

The sequence diagram below further illustrates the interactions between each component including the external sensor boards.



4 USER INTERFACE/EXPERIENCE

For the MVP we develop in Phase 3, the GUI will be quite basic.
Upon startup, the path to the data folder (where the sensor and scenario configuration files as well as the corresponding CSV data is) can be set and then a scenario selected.
Then individual sensors can be excluded from the scenario execution in the UI.
Once the setup is completed, the sensors are prepared with data and the scenario is executed.

5 REQUIRED INPUTS FROM OTHER TEAMS

5.1 Sensor Team Interaction and Requirements

For full-on testing, we need sensor boards which have the relay functionality enabled.

5.2 Digital Twin Team Interaction and Requirements

Since data generation was removed from the VA scope, all that is needed from the DT team are the scenario configs (or the relevant data) and the sensor setup necessary for testing any scenario wanted.

6 EVALUATION METRICS

The software fulfills at least the stated goals for the MVP phase and is useable with minimal introduction by the other teams to test their setup.
This includes the ability of a “layperson” from the DT team to set up a desired scenario via writing sensor and scenario config files and then execute the desired scenario.
This sounds simplistic but perfectly encapsulates a MVP and would enable the other teams to test their MVPs.
Ideally an early version of a GUI based MVP is already functional in the described sense before the deadline to enable practical use for the other teams.

7 APPENDICES

A - Example Config Files

default.scenario

- The filetype has to be .scenario
- The sensors are delimited by ; and consist of the type, id, mqtt-topic delimited by &. The mqtt-topic is optional (if left out a default will be generated from the sensorconfig and the id

```
mqttBrokerURL =  ws://broker.emqx.io:8083/mqtt
start_time = 2011-01-01
end_time = 2013-12-31
sensors =
temperature&1&testing_virtual_apartment/temperature1;temperature&2&testing_virtual_apartment/te
mperature2;humidity&3&testing_virtual_apartment/humidity
description = scenario description, this is the default scenario
```

temperature.sensor

```
#A config file for the temperature sensor
formatting_regex = .*
no_formatting = true
formatting_charset = UTF-16
default_mqtt_topic = temperature
```

B - Risk Assessment

- The dynamic sensor handling cannot handle a specific sensor type
- Sensor team not delivering their deliverables again

- Uncontrollable human factors (team members may get sick or be busy with other courses and exams, inevitably leading to getting less work done than planned)