SDD-VA-02

Virtual Apartment Team

IoT Platform for Senior Homecare

Software Design Document

Phase 2

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 Phase 1 of the IoT Lab Course at the Chair of Computer Architecture and Parallel Systems at the Technical University of Munich. Now, in Phase 2, the teams are tasked with developing a viable prototype 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 prototype we develop during Phase 2 of this project is not expected to have the full functionality described in the previous SDD. Instead, we aim to provide a prototypical "bare-bones" implementation of the system that can send data to one specific sensor type. This is to prove general feasibility of the VA concept and to allow the entire group to test their dataflow. Some coding shortcuts may be used and some components may utilise hardcoded data/information. We strive to produce a prototype that can be evolved on in later phases, but the goal is providing a proof-of-concept.

1.3 Overview

In this document, we will attempt to specify the features we wish to include in this prototypical 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 prototypical 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

The prototype should have the following minimal functionalities:

- Provide hard-coded test scenarios
- Provide hard-coded configurations for the relevant sensor board
- Connect to the sensor board provided by Sensor Team
- Publish messages to the correct topic for the provided sensor board (assuming MQTT is chosen by the sensors team as a protocol)

Within the whole system architecture of the IoT Project, the Virtual Apartment is responsible for providing the data that is sent to the sensor boards implemented by the Sensor Team. This data is then made available to the Digital Twin Team, which attempts to analyse habits based on the data and detect possible anomalies within the data. Finally, the Senior Monitor team generates alarms based on further analyses of the information forwarded by the Digital Twin Team.

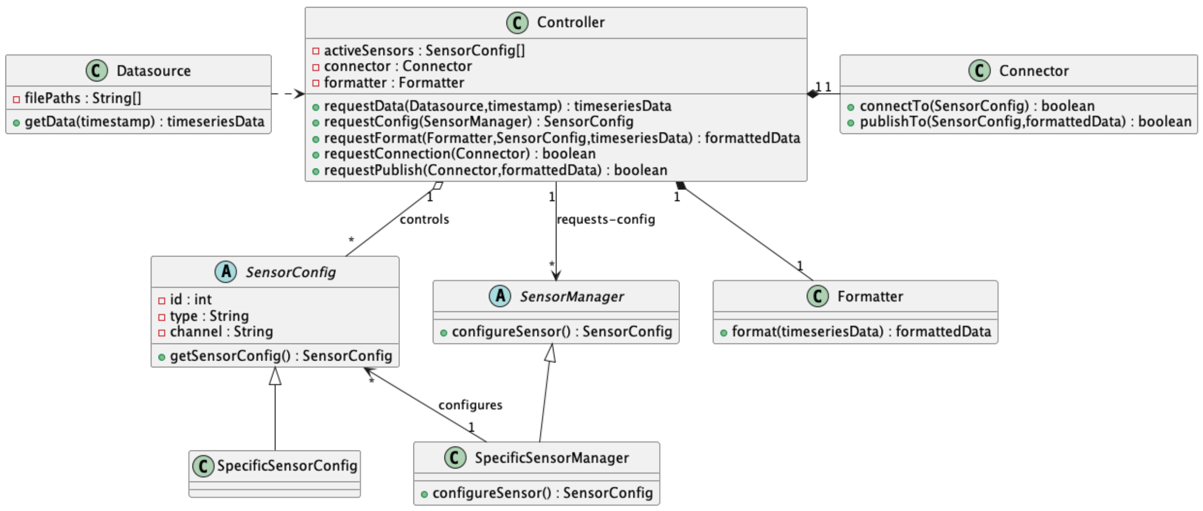
We have already established that the Virtual Apartment system should have an early working minimal setup because the other teams depend on the Virtual Apartment to test their own subsystems. Keeping this in mind, we designed a system architecture that will ensure a minimal working prototype that can be easily extended to include more complex functionalities.

After discussing possible Phase 2 scenarios with the other teams, we agreed to implement a system for temperature anomaly detection, in which drastic or sudden changes in temperature are reported by the monitoring system. It was established that the Sensor Team plans to use a temperature and humidity sensor along with a camera for this scenario. For the prototypical sensor, our team has decided to implement only one sensor for now: the temperature sensor.

3 SYSTEM ARCHITECTURE

3.1 Architectural Design

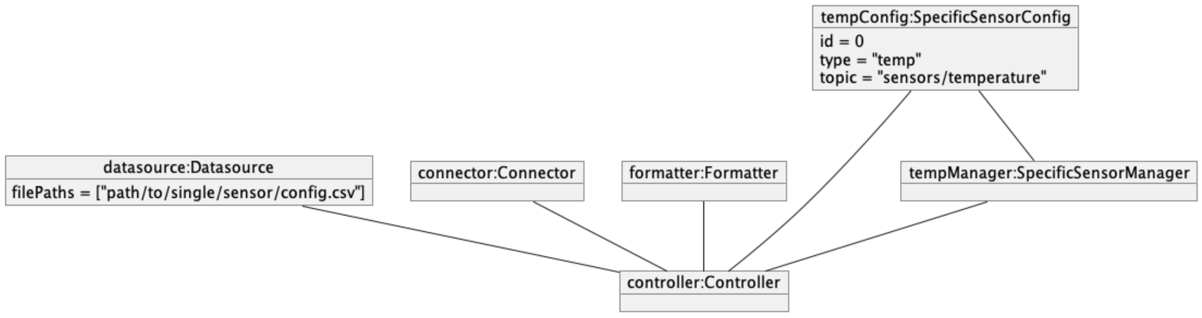
The Virtual Apartment architecture will consist of six main components and will communicate with the rest of the monitoring system platform through the prototype temperature sensor board provided by the Sensor Team. The six main components are the Datasource, the Controller, the Connector, the SensorManager, the SensorConfig and the Reformatter. Whether the abstract classes will remain as an intermediary will be decided late during implementation if enough functionality rests within each.


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
controls
1
*
requests−config
1
*
1
1
SensorConfig
id : int
type : String
channel : String
getSensorConfig() : SensorConfig
SensorManager
configureSensor() : SensorConfig
Formatter
format(timeseriesData) : formattedData
configures
*
1
SpecificSensorConfig
SpecificSensorManager
configureSensor() : SensorConfig

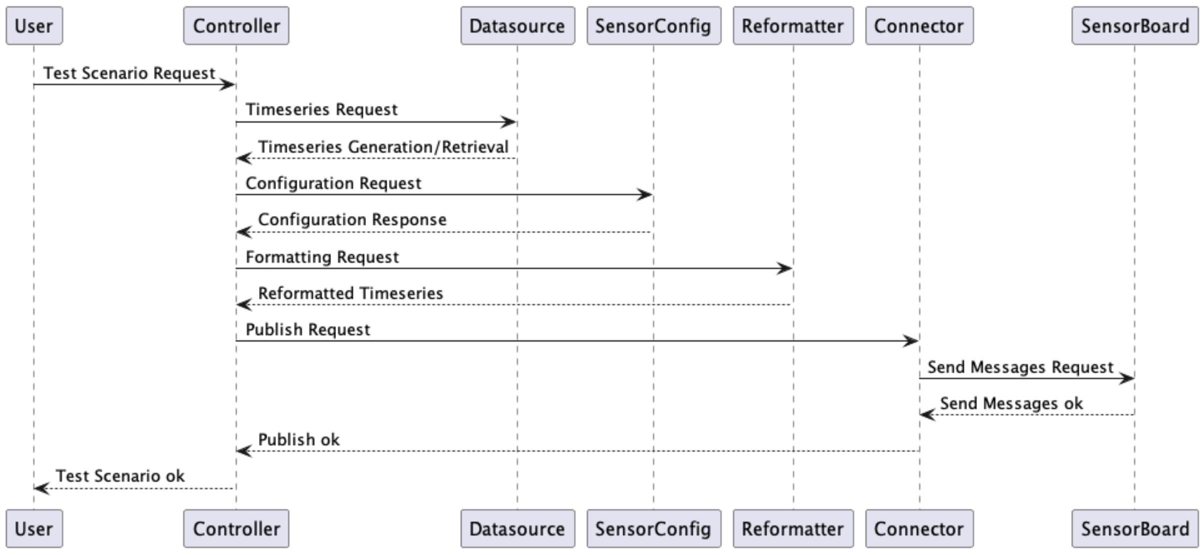
The Controller comprises the core of the system and administers the timing and execution of methods in the other components. In particular, it requests the configuration of the required sensor from the SensorManager, which in turn configures SensorConfig objects that the

Controller can control. The remaining components will be kept quite minimal in terms of functionality for the prototypical system we will develop for Phase 2: The Datasource should be composed of simple data provided in a format such as CSV that can then be retrieved by the Controller. Initially however some hardcoded data will be provided instead. For the Refomatter, we will provide hard-coded message formats and sensor configuration settings for a single sensor. Similarly, the Connector will have the basic functionality of establishing a connection to a single sensor board and publishing data messages to that specific sensor's topic, likely utilising MQTT unless the sensors team chooses a different protocol.

For the specific Phase 2 setup, we present the UML object diagram below. Note that in the minimal case, the Datasource will simply use hard-coded data rather than reading in data from files.


tempConfig:SpecificSensorConfig
id = 0
type = "temp"
topic = "sensors/temperature"
datasource:Datasource
filePaths = ["path/to/single/sensor/config.csv"]
connector:Connector
formatter:Formatter
tempManager:SpecificSensorManager
controller:Controller

The sequence diagram below further illustrates the interactions between each component including the external sensor boards. It is important to note that the sensor board itself should send a confirmation back to the Connector once it has received its published messages. In addition, in the ideal case, we would like to allow the user to request test scenarios when they choose, as opposed to the request being sent as soon the program is executed.



We propose the following minimal, good and ideal component features that may be developed during the course of this phase: (ideal component features are mostly plans for the next phase)

Component	Minimal Features	Good Features	Ideal Features
Controller	Manage & initialise other components upon execution.	Request sensor data from the Datasource upon user request and manage other components based on it.	Extract specific time slices of sensor data from the data received from the Datasource.
Datasource	Provide hard-coded data to the Controller.	Read a file (e.g. CSV) and provide it to the Controller upon request.	Same as in recommended case.
SensorManager	Provides hard-coded configuration for a single sensor and generates a SensorConfig object based on this.	Actually reads or constructs configuration for a single sensor from a file.	Same as in recommended case.
SensorConfig	Provides hard-coded sensor information to the Controller.	Same as in minimal case.	reads configuration files instead of hardcoded values
Formatter	Receives pre-formatted data that is simply passed back to the Controller.	Reformats data to be used as message upon Controller request.	Uses the message formatting in the SensorConfig object provided
Connector	Connection can be established and a message can be sent to one specific sensor board.	Same as in minimal case.	Same as in minimal case.

3.2 Design Rationale

Given the necessity to have an early working prototype as highlighted in Section 2, we provided three tiers of functionalities the system may possess, ensuring reliability over complexity. This also allows us to be more flexible with setting our goals depending on our own and the other teams' requests and progress. The Minimal features already constitute a working prototype usable by the other teams while the “good features” constitute an evolution towards an absolutely minimal MPV.

4 HUMAN INTERFACE DESIGN

For the prototype we develop in Phase 2, the minimal system will simply run when we execute the program akin to a script. Additionally, for the ideal and good versions, we hope to incorporate some basic command line interactivity that will allow the user to specify when and which data they want to be retrieved and published to the sensor board.

5 REQUIRED INPUTS FROM OTHER TEAMS

5.1 Sensor Team Interaction and Requirements

We need to send data to the temperature sensor that is then relayed as if the sensor had measured the temperature itself. To execute this, we need to reach a consensus with the Sensor Team regarding the message formatting for this specific use case sensor. We would propose the JSON format for its widespread use and availability of libraries in any tech stack, but are willing to adapt to their requirements, and MQTT for the protocol. In addition, the Sensor Team must provide us with information regarding the sensor they would like us to implement for this phase. Any labels or structures they require must also be communicated. For this prototypical version, it suffices for the Virtual Apartment to supply fictional min and max values of the virtual temperature readings.

5.2 Digital Twin Team Interaction and Requirements

In terms of the Digital Twin Team, we must agree on the definition of noteworthy events. These would set apart relevant data from “normal” readings so that the Digital Twin Team can use the data to perform adequate anomaly detection. We also need to reach a consensus on specific scenarios, ideally underpinned by realistic data. However, since we are still in the prototypical phase, we plan to work with rough approximations or estimations of realistic and relevant data. The data do not have to be realistic as long as there is sufficient variety within the data to distinguish “normal” readings from noteworthy events.

6 EVALUATION METRICS

Our general expectations for this phase: The VA component exists primarily to enable the other teams to test their parts due to a lack of real apartments and seniors to experiment with. Hence our expectations are to provide this basic testing functionality for the other teams. Additionally, prototyping could also encompass testing the general architecture for the later phases, but the top priority remains the functionality that the other teams rely on (i.e., sending data-messages).

Therefore, the absolutely necessary goals for a barebones minimal prototype would be as follows:

- Connection can be established to the temperature sensor board
    - This metric requires success on the Sensor Team’s end
- A message can be sent to the temperature sensor board (and then relayed if the Sensor team is successful)
    - For this, the pre-formatted message and the sensor configuration can be hardcoded
- The system is run upon execution without any command line interactivity

Building up on this, the optional achievements would be as follows:

- The sensor configuration is actually read from file or constructed by the SensorManager
- A CSV file (or similar) is read as the data source
- The data is reformatted to be published as a message of the relevant sensor’s topic
- The system is run upon execution without any command line interactivity
- Timeslice selection from the read data
- Basic command line interactivity

7 APPENDICES

A - Tentative Timeline for Virtual Apartment Phase 2

WP	TASK TITLE	START DATE	DUE DATE	DURATION	PHASE TWO													
					WEEK 3							WEEK 4						
					F	S	S	M	T	W	R	F	S	S	M	T	W	R
2	**Prototype Development**																	
2.1	Research and Design	19.10.2023	20.10.2023	1	🟩	🟩	🟩											
2.1.1	Feedback Incorporation	19.10.2023	19.10.2023	0	🟩													
2.1.2	Form Consensus with ST and DT	21.10.2023	21.10.2023	0		🟩												
2.2	SDD	20.10.2023	23.10.2023	3	🟩	🟩	🟩											
2.2.1	SDD Revisions and Submission	23.10.2023	23.10.2023	0	▨	▨	▨	🟩										
2.3	Prototype Development	24.10.2023	30.10.2023	6		▨	▨	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩			
2.3.1	Sparse Definition of Specs for Single Sensor	24.10.2023	26.10.2023	2		▨	▨	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩			
2.3.2	Definition of Noteworthy Events for Single Sensor	24.10.2023	26.10.2023	2		▨	▨	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩			
2.3.4	Environment Setup	24.10.2023	25.10.2023	1		▨	▨	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩			
2.3.3	Sparse Implementation of Single Sensor	25.10.2023	30.10.2023	5		▨	▨	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩			
2.4	Prototype Evaluation	30.10.2023	31.10.2023	1		▨	▨	▨	▨	▨	▨	▨	▨	▨	🟩	🟩		
2.4.1	Comparison with SDD Goals	31.10.2023	31.10.2023	0		▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	🟩	🟩	
2.4.2	Updates to Project Goals	01.11.2023	01.11.2023	0											▨	▨	🟩	
2.5	Demonstration	02.11.2023	02.11.2023	0											▨	▨	▨	🟩
2.5.1	Scenario Preparation	30.10.2023	01.11.2023	2											🟩	🟩	🟩	
2.5.2	Presentation Document	30.10.2023	01.11.2023	2											🟩	🟩	🟩	
2.6	Phase 2 Retrospective	02.11.2023	02.11.2023	0											▨	▨	▨	🟩

B - Risk Assessment

- Architecture assumptions may prove wrong (major risk of any medium to large software project, especially with such vague initial requirements)
- Late but major sensor style changes (rather minor risk since the configuration system should be rather universal but very exotic sensor/data types may cause issues)
- Assumed requirements differ from actual ones from Sensor Team and Digital Twin Team (a lot of assumptions had to be made for this document, not all of these will necessarily hold, potentially requiring major redesign)
- Little time to correct problems that may come up, especially when testing (Sensor Team deliverables are planned quite tightly in the hopes of earlier delivery of prototypes)
- 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)