Thermal Grace - VHUB
Comfort-as-a-Service IoT System

Hardware Architecture Design

Artur Kraskov
Semester 7
ICT & OL, Delta
Fontys 2025-2026

Contents

Introduction

Within the scope of VHUB Comfort-as-a-Service (CaaS) IoT system project a MVP prototype was designed and realized. This document covers the design of a distributed computer system including determining microcontrollers, sensors, timing, resource use and performance.

Background

A project plan comprising objectives, problem, literature review, scope and risk analysis was compiled [1]. User-stories were obtained based on literature review and allowed to define the requirements for the system [2].

Hardware Architecture

The following architecture was designed to collect sensor data and process it locally based on the system requirements and project objectives. It wires sensors to microcontrollers and one microcomputer, which stream sensor values over MQTT to the central microcomputer for processing.

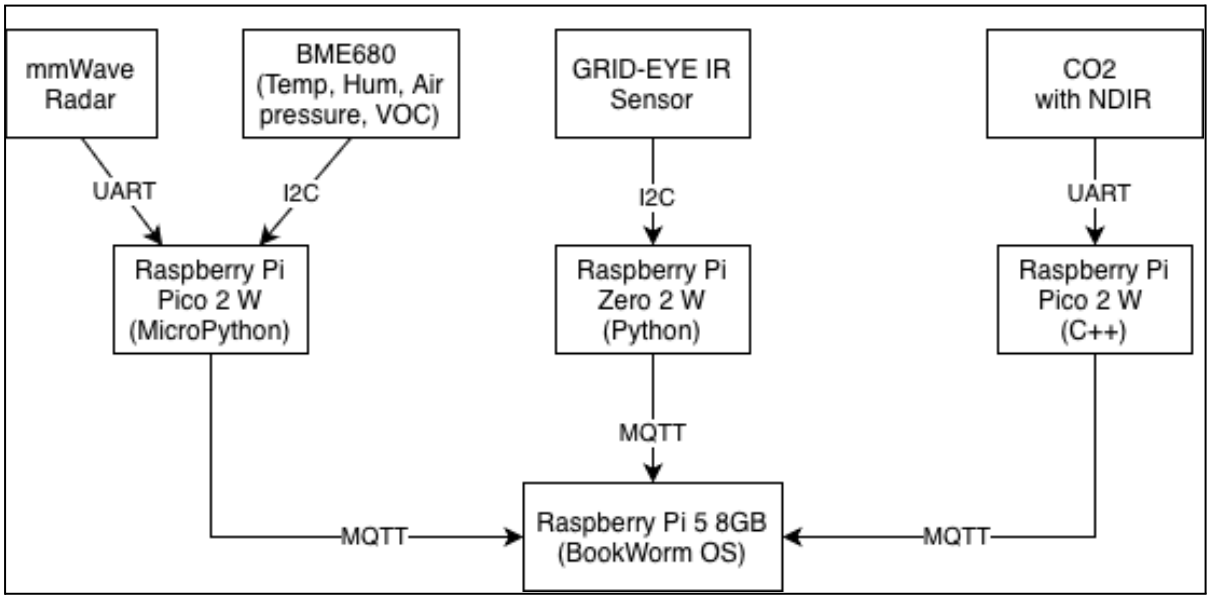

Image 1: hardware architecture diagram

To meet semester time constraints, sensors were split across wireless microcontrollers and microcomputers: one sensor required C++ (others had MicroPython libraries), so two different microcontroller setups were used instead of building a new C++ stack; a Python-only sensor was run on a Raspberry Pi Zero 2 W (Linux + full Python).

Pi Pico 2 W and Pi Zero 2 W (both with Wi-Fi/Bluetooth) communicated with the central computer via MQTT; wireless networking enabled a modular kit where users can attach/detach sensors and relocate them around the office for experiments or personalized setups.

For sensor interfaces, I2C was chosen as the primary hardware bus for its reliability and broad support; UART was used for the sensor that lacked I2C.

Hardware Components

Table 1: hardware components

#	Role	Name	Link	Price
1	Central microcomputer	Raspberry Pi 5 8GB	https://pip-assets.raspberrypi.com/categories/892-raspberry-pi-5/documents/RP-008348-DS-4-raspberry-pi-5-product-brief.pdf?disposition=inline	75€
2	Microcontrollers (2x)	Raspberry Pi Pico 2 w	https://datasheets.raspberrypi.com/pico/pico-2-product-brief.pdf	7€
3	Microcomputer	Raspberry Pi Zero 2 w	https://datasheets.raspberrypi.com/rpizero2/raspberry-pi-zero-2-w-product-brief.pdf	17€
4	mmWave Radar (location & human detection, people counter)	Ai-Thinker Rd-03D 24GHz Radar Sensor Module	https://www.tinytronics.nl/product_files/007073_rd-03d_v1.0.0_specification20240103.pdf	7.5€
5	CO2 Sensor	MemsFrontier MTP40-F CO2 Sensor	https://www.tinytronics.nl/product_files/004660_MTP40-F-CO2-sensor-module-single-channel.pdf	17.25€
6	Temperature, Humidity, Air Pressure, VOC level	BME680 Sensor Module with Level Converter	https://www.tinytronics.nl/en/sensors/air/pressure/bme680-sensor-module-with-level-converter-air-pressure-air-quality-humidity-temperature	10.5€
7	GRID-EYE (Thermal heatmap)	Grove - AMG8833 8x8 Infrared Thermal Temperature Sensor Array	https://www.seeedstudio.com/Grove-Infrared-Temperature-Sensor-Array-AMG8833.html	26€

Sensor usage

PMV/PPD Calculation with PyThermalComfort

The selection of sensors is minimal to provide initial data for the PyThermalComfort PMV/PPD calculation method.

Source: https://pythermalcomfort.readthedocs.io/en/latest/#quick-start

Example

```
# Calculate PMV and PPD using ISO 7730 standard

result = pmv_ppd_iso(
    tdb=25,  # Dry Bulb Temperature in °C
    tr=25,  # Mean Radiant Temperature in °C
    vr=0.1,  # Relative air speed in m/s
    rh=50,  # Relative Humidity in %
    met=1.4,  # Metabolic rate in met
    clo=0.5,  # Clothing insulation in clo
    model="7730-2005"  # Year of the ISO standard
)
```

Implementation

Model: `pythermalcomfort.models.pmv_ppd_iso` with `model="7730-2005"` (ISO PMV/PPD). Returns `pmv` (thermal sensation index) and `ppd` (percent people dissatisfied).

Table 2: PMV/PPD inputs and assumptions

value	**source**
dry-bulb air temperature, C	from MQTT `temperature_c` (live)
mean radiant temperature, C	assumed equal to `tdb` unless future payloads provide it (static assumption)
relative humidity, %	from MQTT `humidity_pct` (live)
air speed, m/s	hard-coded to 0.1 m/s to approximate still indoor air (static default)
metabolic rate, met units	from user feedback app, estimated from the most recent feedback row via `estimate_met` mapping; falls back to 1.2 if not found (derived/static)
clothing insulation, clo units	from user feedback app, estimated from upper/lower garment labels via `estimate_clo`; falls back to 0.7 if not found (derived/static)

PMV/PPD computation

PMV balances human heat gains and losses under steady-state conditions. The ISO model solves for skin heat loss via convection, radiation, sweat evaporation, respiration, and diffusion, then maps the heat balance to the PMV scale (-3 cold to +3 hot). PPD is derived from PMV with the ISO exponential relation, meaning even at PMV = 0 at least 5% are dissatisfied.

CO2, VOC and Air Pressure

These sensors provide additional data for analysis, which doesn't go directly into the PyThermalComfort method. However, future development can integrate it into a more advanced algorithm/model. Currently it is fed together with PMV/PPD into LLM for analysis and advice.

mmWave & GRID-EYE

mmWave Radar is used for indoor location tracking. It recognises targets and provides coordinates, which can be visualized as a 2D radar. Currently the sensor is limited to 3 people with the default library.

GRID-EYE Infrared Thermal Temperature Sensor Array provides an 8*8 (64 pixel) 2D heatmap. It can be used for people recognition and also for temperature measurements. The current implementation is experimental. Potentially it can allow people to measure peoples' temperature or overall heatmap of the room if properly calibrated and set.

System Specifications

Pico 2 W & mmWave radar & environmental sensor (BME680)

Distributed edge node on Raspberry Pi Pico 2 W running MicroPython, pushing mmWave radar targets and BME680 environmental data to MQTT. Covers sensor/IO configuration, timing, resource use, performance expectations, and MQTT contract for downstream consumers.

Sensors and Interfaces

RD03D mmWave radar

Interface: UART0 at 256000 baud on GP0 (TX) / GP1 (RX); initialized in multi-target mode (up to 3 targets) during startup. Frame command bytes set on init; buffer cleared after mode switch.

Frame format: 30 bytes, header 0xAA 0xFF, up to 3 targets, tail 0x55 0xCC. Each target provides x, y (mm), speed (cm/s), pixel_distance (mm); distance and angle are computed per target before publishing.

Update loop waits up to 120 ms for a valid frame; polls UART with 10 ms sleep between checks, trimming noise until header alignment.

BME680 environmental sensor

Interface: I2C0 on GP5 (SCL) / GP4 (SDA) created at startup.

Metrics read on each publish: temperature (°C/°F), humidity (%), pressure (hPa), gas resistance (kΩ). Read wrapped in try/except; failures return null env block.

Timing and Scheduling

Wi-Fi association attempts up to 30 times with 0.5 s spacing (≈15 s max) before giving up.

Main loop base cadence: 20 ms sleep per iteration.

Radar polling: `radar.update()` scans UART until a valid frame or a 120 ms timeout; only on success does the loop proceed to publish.

Publish guard: minimum spacing `PUBLISH_INTERVAL_MS = 100` (10 Hz max) measured from last successful publish, gated on radar data availability.

BME680 read occurs only when a publish is due, keeping sensor sampling aligned with publish cadence.

Error handling: on MQTT/Wi-Fi exceptions, client is dropped, a 2 s pause is taken, and reconnect attempted next loop.

Resource Use and Performance

Latency: worst-case detection-to-publish path ≈120 ms (radar frame wait) + up to 100 ms (publish guard) + MQTT send; target end-to-end <220 ms for radar-triggered packets.

Throughput: up to 10 publishes/sec when radar frames stream continuously; bounded by radar availability and Wi-Fi/MQTT health.

UART load: each radar frame 30 bytes; at 256000 baud transfers in ≈1 ms, well below the 120 ms frame wait budget.

I2C load: single BME680 read per publish; negligible bus occupancy relative to 100 ms interval.

MQTT payload size: JSON with timestamp + up to 3 targets + 5 BME fields; typical 200–600 bytes/publish → ~6 kB/s at 10 Hz plus MQTT overhead.

Memory/CPU: stores at most 3 target objects and one payload dict; main loop includes 20 ms idle sleep and lightweight math, fitting comfortably within typical MicroPython heap and CPU headroom on Pico 2 W.

MQTT Interface

Broker: 192.168.x.x, TCP port 1883, no TLS, no auth (user/password None).
Client ID: `pico-mmwave-air`; keepalive 60 s; QoS 0 (default); retain flag not set.
Topic: sensors/pico/air_mmwave (bytes literal in code).

Payload structure (JSON):

`timestamp_ms`: MCU tick count (ms).

`radar`: `target_count` (0–3), `targets` list with per-target `id`, `angle` (deg), `distance_mm`, `speed_cms`, `x_mm`, `y_mm` as published in `build_radar_payload()`.
`environment`: object with BME680 readings or null if read failed.

Session behavior: reconnects if Wi-Fi drops or MQTT errors occur; Wi-Fi rejoin clears the MQTT client so a fresh session is established on the next loop.

Assumptions and Options

BME680 oversampling/filter settings use library defaults; actual sample time depends on those defaults but is typically <100 ms.

If network reliability is critical, consider QoS 1 and retained status beacons; current code uses QoS 0 for lowest latency.

TLS is not configured; broker must be on trusted LAN. Add TLS and credentials if required by security policy.

Publish rate can be lowered by increasing `PUBLISH_INTERVAL_MS` to reduce network use or power draw; radar still runs at UART frame rate.

CO2 NDIR Node (MTP40F on Pico 2 W)

Implemented with C++ firmware.

Sensor: MTP40F NDIR CO2 module on UART Serial1 at 9600 baud, pins GP6 (RX from sensor TX) / GP7 (TX to sensor RX).

Timing: CO2 sample pulled every 2.5 s and published immediately; LED strobes 50 ms per sample for operator feedback. Base loop otherwise free-runs; Wi-Fi status reported every 5s.

Connectivity: Wi-Fi station joins with 10s deadline and 200ms poll interval; MQTT connect attempted after Wi-Fi success. No TLS or credentials.

MQTT: broker 192.168.50.176:1883, client ID `pico2w-mtp40f`, topic sensors/pico/mtp40f/co2, QoS 0/retain false (PubSubClient default), payload JSON `{"co2_ppm":<int>}` sized <64 bytes.

Performance: throughput ~0.4 Hz (one publish per 2.5 s), payload <100 bytes → <0.1 kB/s; network latency dominated by Wi-Fi/MQTT but bounded by 2.5 s sensor cadence. CPU/memory load minimal (single integer payload, small stack buffer).

Reliability: If Wi-Fi disconnects, reconnect is attempted and MQTT session re-established before next publish; publishes are skipped when not connected. Consider adding retained heartbeat or QoS 1 if broker-side reliability is required.

Notes: MQTT and Wi-Fi timings are network-dependent; payload size (200–600 B) is an estimate but aligns with the JSON fields built in main.py:36-49 and the environment block. The Pico 2 W has ample headroom for this load, and nothing in the code exceeds typical RAM/CPU limits.

Conclusion

Described hardware architecture meets the requirements for a modular sensor kit at the prototyping stage. Sensors are easily mountable and can be swapped. The connection between different nodes is wireless and doesn't depend on the distance unless wi-fi is there. To avoid the use of the network for sensor nodes, they can connect to Raspberry Pi 5 as an Access Point, but then it is recommended to use Pi 5 without internet or use ethernet. Separate document covers software architecture and implemented perceived thermal comfort sensing and predicting solution.

Reference

1. Kraskov A., Kaszuba B., (2025), Vitality HUB Perceived Thermal Comfort | Project Plan, Thermal Comfort-as-a-Service (CaaS) IoT system, FHICT Delta, [Online], Available: https://drive.google.com/file/d/1BXHXlD4_SPatpJrU1mDwF0ZKnESl2W_U
2. Kraskov A., Kaszuba B., (2025), Vitality HUB Perceived Thermal Comfort | Requirements, Thermal Comfort-as-a-Service (CaaS) IoT system, FHICT Delta, [Online], Available: https://drive.google.com/file/d/1NooyuNM97BFz3jYOX2aPSRkbFZ7f5yJF