Final Report
OPC UA Industrial Plant HUB

Final Report for the Distributed Systems Course [A.A 2025-2026]

Federico Capitanio
federico.capitanio@studio.unibo.it

February 2, 2026

This report illustrates **OPCUA Industrial Plant Hub**, a software developed to demonstrate the fundamental principles of Distributed Systems by integrating them with the world of industrial automation.

In the world of industrial automation, the issue of data collection from machines and machine lines is becoming increasingly important. Therefore, this project aims to provide a distributed data collection system for different production lines, which may also be distributed across the territory.

The project is based on the use of IoT devices, which are connected to the production line and collect data on the production process. The data collected is then sent to a central server, which stores it and allows it to be analysed. The data can also be sent to the machines themselves, allowing them to be controlled and optimised.

The underlying idea is that data is collected from the machine line using the **OPC UA** protocol, widely used in industry, saved and then retrieved via **HTTP API**.

Unlike traditional data collection architectures based on a single physical node that reads data, the architecture of this project consists of three HUB nodes that collect **OPC UA** data, synchronised via anti-entropy and gossip protocols, with an nginx load balancer that distributes client requests and implements data encryption via the **TLS** protocol.

The project emphasises the **AP** behaviour of the **CAP Theorem**, i.e. it focuses on the aspects of **Partition Tolerance** and **High Availability**, thus ensuring that clients can always access data even in the presence of network partitions or individual node failures. The **Strong Consistency** concept is therefore not guaranteed, favouring data that is not updated to the latest possible value over a longer wait time that would result from the application of **Strong Consistency** itself.

The system allows OPC UA sources to be added dynamically, by implementing an **on the fly** automation line connection mode, allowing individual

machines or entire lines to be connected and added without the system needing to be restarted.

Figure 1: This is an example image

Quick LaTeX suggestions

If you need to cite a reference, you can use the `cite` command, like providing the BibTex key of some entry in the `references.bib` file, e.g.: [1].

You can find pre-coocked BibTex entries for most Computer Science papers on DBLP.

If you need to include an image, let LaTeX decide where to put it by using the `figure` environment, with a `includegraphics` command inside it. If your need to reference a figure, use the `label` command to assign a label to the figure, and then use the `cref` command to reference it. Put figures in the `figures` folder. A complete example is shown in fig. 1.

Do **not** put any placement constraint on figures, such as `[h]` or `[h!]`. Same considerations apply for other floating elements (e.g. tables, algorithms, listings, etc.).

Do **not** use the \\ or `\newline` commands to break lines.

Just leave an empty line between two paragraphs to start a new one. The new line will be automatically indented. This is intended: it is the LaTeX way to separate capoverses.

1 Concept

Here you should explain:

- The type of product developed with that project, for example (non-exhaustive):
  - Application (with GUI, be it mobile, web, or desktop)
  - Command-line application (CLI could be used by humans or scripts)
  - Library
  - Web-service(s)
- Use case description:
  - *what* is the software doing?

  - *where* are the users?
  - *when* and *how frequently* do they interact with the system?
  - *how* do they *interact* with the system? which *devices* are they using?
  - does the system need to *store* user's **data**? *which*? *where*?
  - most likely, there will be *multiple* **roles**

- Why is distribution needed?
  - geographically distributed environments?
  - computation speedup?
  - resource sharing?
  - fault tolerance?
  - other reasons?

  In any case, please elaborate.

2 Requirements Elicitation and Analysis

- The requirements must explain **what** (not how) the software being produced should do.
  - you should not focus on the particular problems, but exclusively on what you want the application to do.
- Requirements must be clearly identified, and possibly numbered
- Requirements are divided into:
  - **Functional**: some functionality the software should provide to the user
  - **Non-functional**: requirements that do not directly concern behavioural aspects, such as consistency, availability, etc.
  - **Implementation**: constrain the entire phase of system realization, for instance by requiring the use of a specific programming language and/or a specific software tool
    - these constraints should be adequately justified by political / economic / administrative reasons...
    - ... otherwise, implementation choices should emerge *as a consequence of* design
- If there are domain-specific terms, these should be explained in a glossary
- Each requirement must have its own **acceptance criterion**
  - these will be important for the validation phase

2.1 Relevant Distributed System Features

Motivate which distributed system features are relevant for your project, and which are not.

- transparency
  - does your system need to hide distribution details from users or developers?
  - is it important that failures, location, or replication are invisible?
- fault tolerance, dependability – availability, reliability, integrity, maintainability, safety
  - what happens if a component fails? is uninterrupted service required?
  - is data loss or corruption unacceptable?
  - how quickly must the system recover from faults?
- scalability
  - will the system need to handle increasing numbers of users, requests, or data?
  - is it expected to grow over time?
- security, trust
  - is sensitive data being processed or stored?
  - are there multiple user roles with different permissions?
  - is authentication or authorization required?
- resource sharing
  - do multiple users or components need access to shared resources?
  - is coordination or synchronization needed?
- openness, interoperability, heterogeneity of components
  - Will your system interact with external systems or use components built with different technologies?
  - Is standardization or compatibility important?
- evolvability, maintainability
  - Will the system need to be updated or extended after deployment?
  - Is long-term maintenance a concern?
- performance, concurrency, computation / communication efficiency, bandwidth
  - Are there strict requirements on response time or throughput?
  - Will many operations happen in parallel?
  - Is network usage a concern?

- economy, costs
  - Are there budget constraints for development, deployment, or operation?
  - Is minimizing resource usage important?

3 Design

This chapter explains the strategies used to meet the requirements identified in the analysis. Ideally, the design should be the same, regardless of the technological choices made during the implementation phase.

> You can re-arrange the sections as you prefer, but all the sections must be present in the end

Important: try to motivate your design choices in relation to the requirements and features identified in section 2 and section 2.1.

3.1 Architecture

- Which architectural style?
  - why?

3.2 Infrastructure

- are there *infrastructural components* that need to be introduced? *how many*?
  - e.g. *clients*, *servers*, *load balancers*, *caches*, *databases*, *message brokers*, *queues*, *workers*, *proxies*, *firewalls*, *CDNs*, *etc.*
- how do components *distribute* over the network? *where*?
  - e.g. do servers / brokers / databases / etc. sit on the same machine? on the same network? on the same datacenter? on the same continent?
- how do components *find* each other?
  - how to *name* components?
  - e.g. DNS, *service discovery*, *load balancing*, *etc.*

Component diagrams are welcome here

3.3 Modelling

- which **domain entities** are there?
  - e.g. *users*, *products*, *orders*, *etc.*
- how do *domain entities* **map to** *infrastructural components*?

  - e.g. state of a video game on central server, while inputs/representations on clients
  - e.g. where to store messages in an IM app? for how long?
- which **domain events** are there?
  - e.g. *user registered*, *product added to cart*, *order placed*, *etc.*
- which sorts of **messages** are exchanged?
  - e.g. *commands*, *events*, *queries*, *etc.*
- what information does the **state** of the system comprehend
  - e.g. *users' data*, *products' data*, *orders' data*, *etc.*

Class diagram are welcome here

3.4 Interaction

- how do components *communicate*? *when*? *what*?
- *which* **interaction patterns** do they enact?

Sequence diagrams are welcome here

3.5 Behaviour

- how does *each* component **behave** individually (e.g. in *response* to *events* or messages)?
  - some components may be *stateful*, others *stateless*
- which components are in charge of updating the **state** of the system? *when*? *how*?

State diagrams are welcome here

3.6 Data and Consistency Issues

- Is there any data that needs to be stored?
  - *what* data? *where*? *why*?
- how should *persistent data* be **stored**?
  - e.g. relations, documents, key-value, graph, etc.
  - why?
- Which components perform queries on the database?
  - *when*? *which* queries? *why*?

  - concurrent read? concurrent write? why?

- Is there any data that needs to be shared between components?
  - *why*? *what* data?

3.7 Fault-Tolerance

- Is there any form of data **replication** / federation / sharing?
  - *why*? *how* does it work?
- Is there any **heart-beating**, **timeout**, **retry mechanism**?
  - *why*? *among* which components? *how* does it work?
- Is there any form of **error handling**?
  - *what* happens when a component fails? *why*? *how*?

3.8 Availability

- Is there any **caching** mechanism?
  - *where*? *why*?
- Is there any form of **load balancing**?
  - *where*? *why*?
- In case of **network partitioning**, how does the system behave?
  - *why*? *how*?

3.9 Security

- Is there any form of **authentication**?
  - *where*? *why*?
- Is there any form of **authorization**?
  - which sort of *access control*?
  - which sorts of users / *roles*? which *access rights*?
- Are **cryptographic schemas** being used?
  - e.g. token verification,
  - e.g. data encryption, etc.

4 Implementation

Please report here all the implementation (technology-dependent) choices you made while implementing your design.

> If you run out of time for this project, you may consider leaving some aspect unimplemented, and simply discuss how you would have implemented them. In this case, better would be to discuss unimplemented features in section 9.

- which **network protocols** to use?
  - e.g. UDP, TCP, HTTP, WebSockets, gRPC, XMPP, AMQP, MQTT, etc.
- how should *in-transit data* be **represented**?
  - e.g. JSON, XML, YAML, Protocol Buffers, etc.
- how should *databases* be **queried**?
  - e.g. SQL, NoSQL, etc.
- how should components be *authenticated*?
  - e.g. OAuth, JWT, etc.
- how should components be *authorized*?
  - e.g. RBAC, ABAC, etc.

4.1 Technological details

- any particular *framework* / *technology* being exploited goes here

5 Validation

5.1 Automatic Testing

- how were individual components ***unit*-test**ed?
- how was communication, interaction, and/or integration among components tested?
- how to ***end-to-end*-test** the system?
  - e.g. production vs. test environment
- for each test specify:
  - rationale of individual tests
  - how were the test automated
  - how to run them
  - which requirement they are testing, if any

recall that *deployment* **automation** is commonly used to *test* the system in *production-like* environment

recall to test corner cases (crashes, errors, etc.)

5.2 Acceptance test

- did you perform any *manual* testing?
  - what did you test?
  - why wasn't it automatic?

6 Deployment

- should one install your software from scratch, how to do it?
  - provide instructions
  - provide expected outcomes
- what software should be installed on the machines to run your project? which versions?
- should one set environment variables or configuration files?
- if you're using containerization (e.g. Docker), describe how you engineered your deployment scrips (e.g. `docker-compose.yml` files)

7 User Guide

- how to use your software?
  - provide instructions
  - provide expected outcomes
  - provide screenshots if possible

8 Self-evaluation

- An individual section is required for each member of the group
- Each member must self-evaluate their work, listing the strengths and weaknesses of the product
- Each member must describe their role within the group as objectively as possible.

It should be noted that each student is only responsible for their own section.

9 Future works

Discuss possible future works, improvements, extensions, optimizations, etc.

Also discuss here any unimplemented feature, and how you would have implemented it.

References

[1] D. Adams. *The Hitchhiker's Guide to the Galaxy*. San Val, 1995.