1. Record architecture decisions

Date: 2025-02-27

Status

Accepted

Context

We need to record the architectural decisions made on this project.

Decision

We will use Architecture Decision Records, as described by Michael Nygard.

Consequences

See Michael Nygard's article, linked above. For a lightweight ADR toolset, see Nat Pryce's adr-tools.

1. Microservice architecture

Date: 2025-03-04

Status

Accepted

Context

To tackle this project we needed to decide on an architecture that allows us to flexibly meet the requirements of the remainder of the semester. As this course is focussed on event-driven architectures, we needed to decide on an architecture that allows us to incorporate different

Decision

We will use a microservice architecture to integrate the different components to be able to demonstrate the interplay of different aspects of an event-driven architecutre.

Consequences

Micro-service architecture will be used.

3. Using Kafka for communication

Date: 2025-03-04

Status

Accepted

Context

For communication between the microservices a system in needed that adhers to event-driven architecutres.

Decision

As the course requires and suggest the use of kafka, kafka will be the main event-driven system for use.

Consequences

Implementing kafka for communication between the micro-services. It also has the consequence of planning and defining the different kafka topics, along with the process flow of which micro-services needs to be producing and listening to which topic.

4. Using protobuf for messages

Date: 2025-03-04

Status

Accepted

Context

To make sure we have good compability and are congruent with the messages that are sent in kafka a decision needed to be made to which standard would be used throughout this project.

Decision

The decision was made to use protobuf due to its compact and efficent way of serializing structured data.

Consequences

Protobuf will be used for all communication between our services.

5. mixing orchesrtation and choreography

Date: 2025-03-04

Status

Accepted

Context

In the context of event-driven architectures two different approaches are discusssed. These are orchestration and choreography. While orchestration can simplify the overall flow by providing a single point of control, it also introduces a single point of failure and potential bottlenecks.A choreography can provide greater flexibility and scalability by allowing services to react independently to events, but the lack of a central coordinator can make the system more complex to manage and troubleshoot.

Decision

To evaluate and become familiar with the up- and down-sides of both of these flows, both will be present in different aspects of the project.

Consequences

Determin the process that will be replicated and decide which steps will orchestrated and which will be implemented in a choreography. The goal here is to use both in scenarios where it makes sense. As the flow is not fully determined at this stage, the decision on which exact services will use which flow will be documented later on.

6. seperating out services from robots

Date: 2025-03-04

Status

Accepted

Context

In the current implementation some features, such as the conveyor belt is attached to one of the robots (and so is the colour sensor). This means, at this time, these services are tightly coupled to this robot. This may not be desirable in an event driven architecture.

Decision

Peniding technical feasiblity all of these services are to separated out, into seperate microservices so that these services are decoupled and processes may happen in parallel.

Consequences

Technical feasiblilty will need to be checked. Especially the independence of the API. Questions that need to be revisited before fully implementing seperate service: Can the robot and theconveyer be called at the same time or does the robots current task need to be complete before calling to theconveyer belt. On an architectual level the consequence for this is more services that allow for seperation of concern. This is also important as especially during error scenarios having the conveyer belt seperate may allow the system to clear the workspace/conveyer so that other processes can continue to complete different task without the whole system coming to a standstil.

7. using camunda 7

Date: 2025-03-17

Status

Accepted

Context

To move forward without BPMN model, and its implementation we needed to decide which version of Camunda to use. The options were Camunda7, which runs locally, but for which support is running out, or Camunda8, its new version that is cloud-based. One of the upsides of Camunda8 are the wide array of adapters that are already supplied, therefore reducing the need for self-written java adapters. One upside of Camunda7 is its local apporach which may make integrating hardware easier.

Decision

Due to the local Camunda7 being able to be deployed locally, we decided to go with Camunda7. It will make the process of integrating the java services a little bit more manual, however it will save us from finding some new implementation of our Kafka servers for the robots in the lab to pick up commands.

Consequences

Our BPMB will be created and deployed using Camunda7. It will result in us needing to do some extra implementations to call the services, while making it eaier when sending commands to the robots.

8. messageType in Protobuf header

Date: 2025-03-17

Status

Accepted

Context

We need a structure to discern between different events/commands/sensordata so that services know what they are listening to and complete the tasks only meant for the respective service.

Decision

To ensure that services can liste to the correct commands for them we implemeted a messageType in the header of the protobuf messages.

Consequences

We have a system that allows us to more accuratly listen and emit events and commads to ensure that the correct messages are received by the services.

9. Three Kafka topics

Date: 2025-03-18

Status

Accepted

Context

To move forward with the implementation, how and what topics for Kafka will be used, need to be decided.

Decision

To strike a balance between order and chaos we decided on implementing and working with three topics for Kafka. These three topics are:

Commands - here all commands are sent for execution which the services listen too. Typically these are 1:1

Events - here events are posted by different services that may have multiple listeners

Sensor - there the sensors that may emit a lot of messages post their sensor data, this was deliberatly excluded from the events topic so it does not get spammed

Consequences

Workig with these three topics in Kafka for the purpose of this project. For every service that is implemented it has to be decided which of these three topics the services listens too and where it emits messages.

10. implementing mock server

Date: 2025-03-18

Status

Accepted

Context

The hardware for this porject is not readily available online for us to use and needs to be accessed via the local network in the lab. This makes testing our system and devloping for it more difficult.

Decision

To facilitate easier trials and aid implementation a mock server was implemented to imitate the robot and convoy services.

Consequences

This allows us to already implement some services and make sure they are working correctly before testing it in the lab and work with the physical hardware.

11. orchestrated workflow with single camunda woorkflow

Date: 2025-03-18

Status

Accepted

Context

As we have seen in the lecture, multiple instances of Camunda may be run depending on the set up and workflow of the application. In the example from the lecture, the workflows for payment and order fullfilment were seperated out into seperate instances in Camunda. This allows for further decoupling. For our project we need to decide if multiple workflows make sense and if they do how they would be split and implemented.

Decision

Only a single workflow of camunda will be run and implemented.The reason for this is the tightly integrated hardware that executes a workflow in a specific order. Only one workflow can be run at a single time with a single workpiece. There is no opportunity to one workpiece to “work around” or “overtake” another workpiece. As the physical execution in sequential and coupled it makes sense to reflect this in the Camunda workflow too.

Consequences

Only a single BPMN will be created and deployed that depicts the whole workflow for this application.

12. Human Intervention for missing Events

Date: 2025-03-21

Status

Superceded by 13. Stateful Resilience Pattern-for-missing-Events

Context

Within our event-driven and process-oriented workflow architecture, certain events may be lost or fail to trigger due to system faults, network issues, or as we expect, hardware issues. This missing event scenario can halt the progression of our workflow, creating risks to the overall system's reliability and throughput. At the moment we expect this to happen at three stages, where it can be detected automatically. 1. Picker robot: Block is moved to the conveyor belt 1. Event from right sensor missing, that a block was detected. 2. Conveyor belt: Block moved to color robot pickup point 1. Event from left sensor missing, that a block was detected. 3. Color robot: Block is moved to color sensor 1. Event from color sensor service missing.

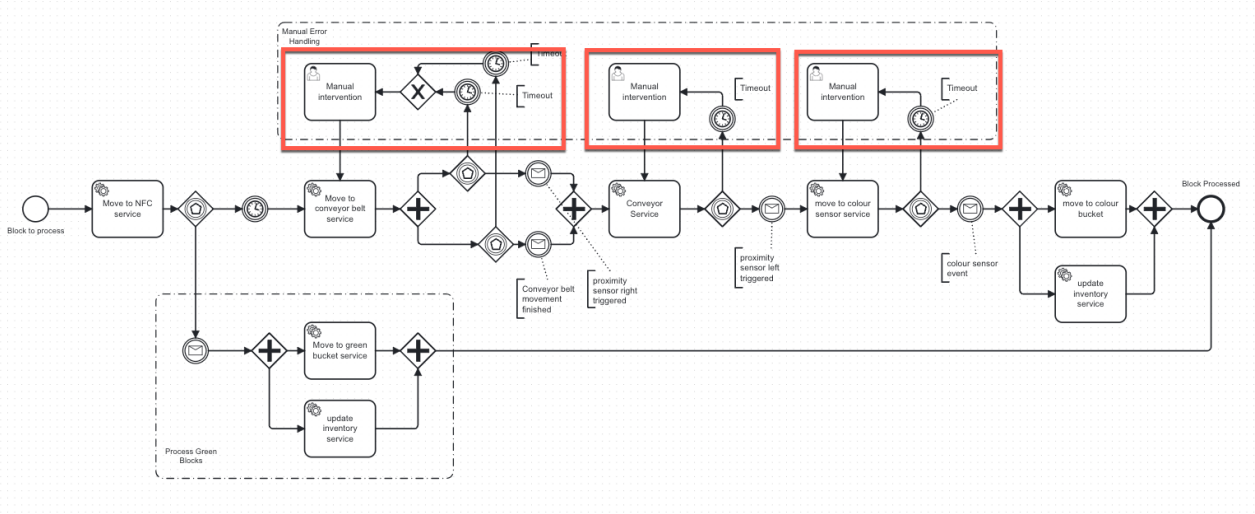

Figure 1: img.png

Decision

We will implement human intervention as a stateful resilience pattern within our workflow. This means when an expected event is not received within a predefined timeout, the system transitions into a state awaiting human interaction. The system's current state, at the point of failure, will be persisted to allow manual recovery or intervention. After human intervention, the last action will be repeated.

Rationale

- Human intervention provides a controlled method to handle exceptional cases, avoiding indefinite suspension of workflow processes.
- Maintaining a stateful context allows human operators to easily understand the situation and intervene effectively.
- Without seeing the robots and conveyor belt in action it is not determinable, if a stateful retry could be applicable.
-

Consequences

Positive

- Improved resilience to unexpected scenarios or failures.
- Reduced risk of indefinite process blockage.
- Enhanced observability and controllability in failure scenarios.

Negative

- Introduction of manual interventions could lead to increased operational overhead.
- Potentially slower recovery times compared to automated solutions.

Conclusion

If future tests with the actual hardware indicate errors that could be handled more appropriately with a stateful automated retry, this decision will be reconsidered. For example, if a robot fails to grab a block and the block remains in place, the robot could automatically retry grabbing it without requiring human intervention.

13. Stateful Resilience Pattern for missing Events

Date: 2025-03-25

Status

Accepted

Supercedes 12. Human Intervention for missing Events

Context

Within our event-driven and process-oriented workflow architecture, certain events may be lost or fail to trigger due to system faults, network issues, or as we expect, hardware issues.
This missing event scenario can halt the progression of our workflow, creating risks to the overall system's reliability and throughput.
At the moment, we expect this to happen at three stages, where it can be detected automatically:

1. **Picker robot: Block moved to conveyor belt**
   - Event from picker robot missing (block positioned on NFC reader).
   - Event from right sensor missing (block detection).
   - Event from robot missing (block placed on conveyor belt).
2. **Picker robot: Block sorted in green basket**
   - Event from picker robot missing (green block sorted).
3. **Conveyor belt: Block moved to color robot pickup point**
   - Event from left sensor missing (block detection).
   - Event from right sensor missing (block removal).
   - Event from conveyor belt missing (block movement).
4. **Color robot: Block moved to color sensor**
   - Event from color sensor service missing.
   - Event from left sensor missing (block removal).
5. **Color robot: Block sorted in non-green basket**
   - Event from color robot missing (block sorted).

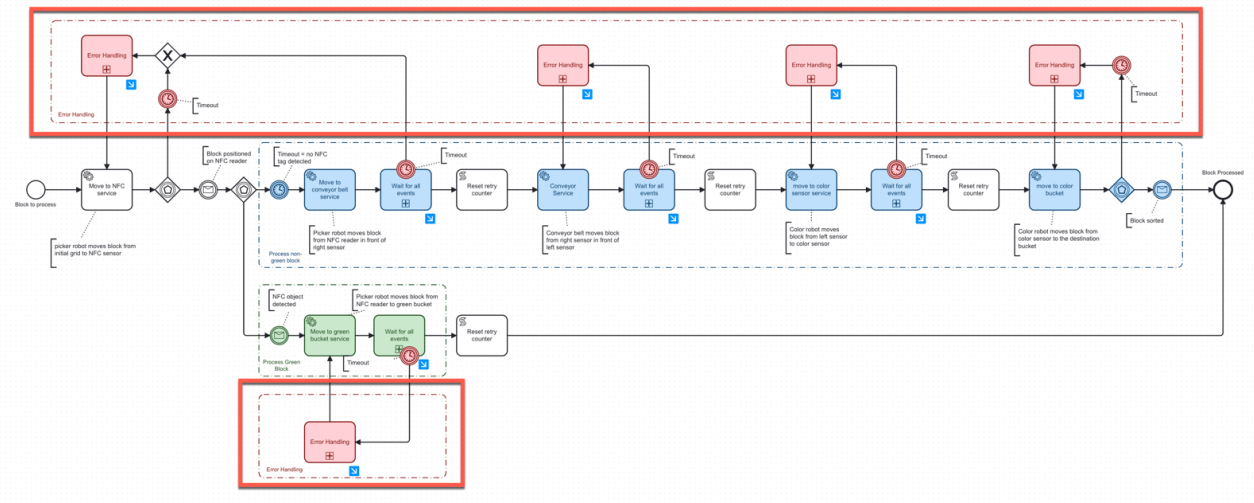

Figure 2: process_error_handling.png

Decision

We will implement stateful retry and human intervention in combination as a stateful resilience pattern within our workflow. This means when an expected event is not received within a predefined timeout, the system transitions into an error-handling state.
The system's current state, at the point of failure, will be persisted to allow automatic retry or manual recovery or intervention.

Implementation

It is configurable via `application.properties`, defining which error handling blocks will attempt an automatic retry and which rely solely on manual intervention.
This allows configuring the error handling based on tests with the actual hardware.

If the automatic retries fail, the process will always end in manual intervention.

Rationale

- Robots can automatically retry simple steps, such as grabbing the block from the grid.
- Human intervention provides a controlled method to handle exceptional cases, avoiding indefinite suspension of workflow processes.
- Maintaining a stateful context allows human operators to easily understand the situation and intervene effectively.

Consequences

Positive

- Improved resilience through automated retries of simple actions (e.g., robotic grabs).
- Reduced human involvement in frequent, easily resolvable scenarios, freeing resources for more complex tasks.
- Enhanced reliability and predictability of workflow executions due to clearly defined state transitions.
- Easier debugging and operational transparency by maintaining persistent states at failure points.
- Improved overall operational efficiency and responsiveness, avoiding unnecessary workflow delays.

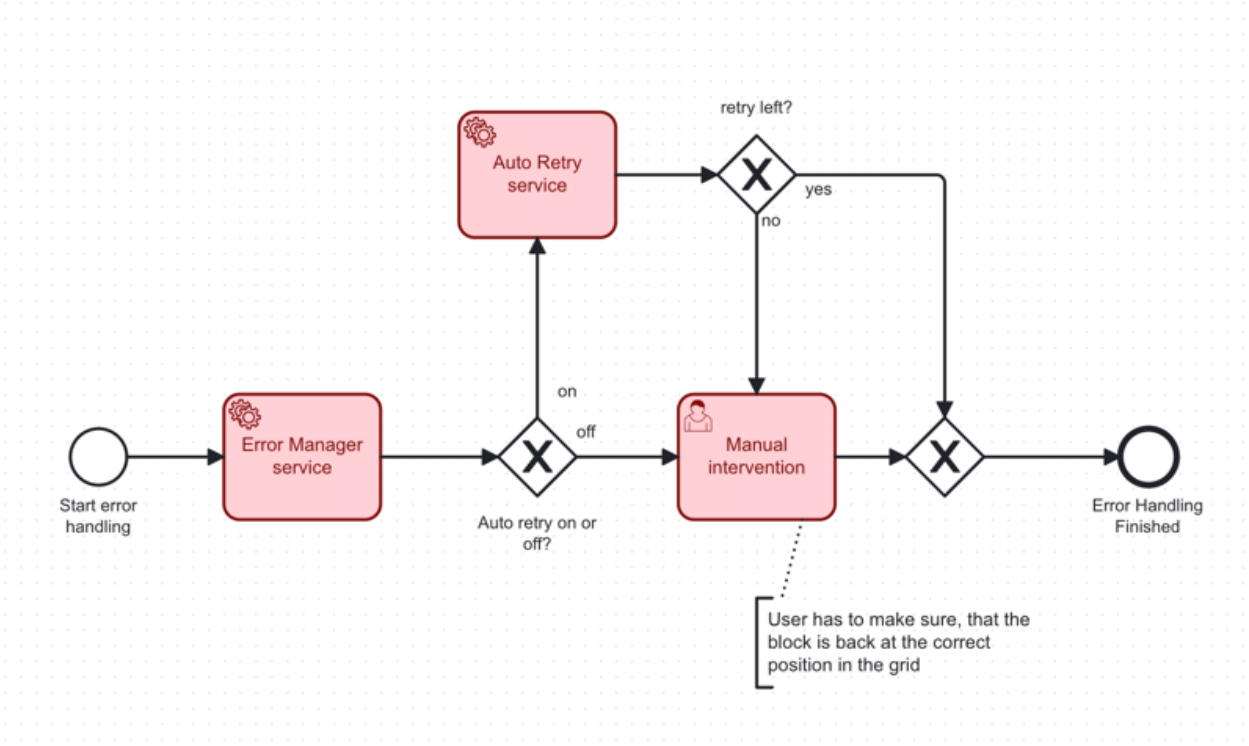

Figure 3: process_error_handling_detail.png

Negative

- Increased system complexity due to additional error-handling logic.
- Potentially larger message payloads or additional data storage required to persist state information.
- Configuration complexity to correctly set thresholds for automated retry versus manual intervention.
- Risks of repetitive automated failures if underlying issues aren't promptly resolved.

Conclusion

The introduction of a stateful resilience pattern combining automatic retries and human intervention provides improved fault tolerance and operational efficiency in handling missing or delayed events.
This approach balances operational efficiency and human control by automating simple retries and reserving human intervention for exceptional or complex error scenarios.

If future hardware test runs indicate errors that could be handled more appropriately with additional automated stateful retries, this is already fully configurable.