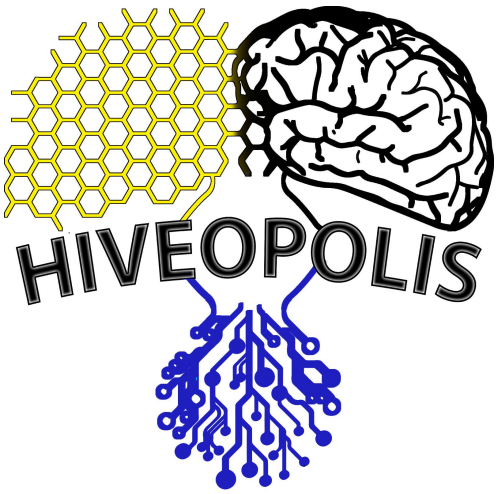

FUTURISTIC BEEHIVES FOR A SMART METROPOLIS

Deliverable D3.3

Data warehouse with implemented DSS

Lead Beneficiary	LLU
Delivery date	31.03.2022
Dissemination Level	PU
Version	1.0
Project website	www.hiveopolis.eu

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 824069

DELIVERABLE SUMMARY SHEET

Project number	824069
Project Acronym	HIVEOPOLIS
Title	FUTURISTIC BEEHIVES FOR A SMART METROPOLIS
Deliverable No	D3.3
Due Date	Project month M36
Delivery Date	31.03.2022
Name	Data warehouse with implemented DSS
Description	Data warehouse for online and offline data collection, storage and analysis will be developed. Several algorithms and models will be integrated for Decision support system development. Based on DSS recommendations, decisions for beekeepers will be provided.
Lead Beneficiary	LLU
Partners contributed	BST, EPFL
Dissemination Level	Public

Introduction

Purpose and scope of the document

The deliverable D3.3 presents the per-HIVEOPOLIS unit software architecture, including the data warehouse and the decision support system. The aim of the on-hive software infrastructure is to provide a pipeline for data acquisition, processing, and subsequent action recommendation and decision-making. This report provides an overview of the designed data warehouse architecture, and accompanies three demonstrator videos that showcase several different elements of the data warehouse and the decision support system.

Overview of the document

This report starts by outlining the operational requirements of the core data warehouse before describing the designed architecture and key elements, in chapter 1. Chapter 2 focuses on the kernel of each decision-support pathway: the models that perform state identification, anomalous condition detection, and make predictions for the future evolution of the colony or infrastructure. The final chapter provides some additional information regarding each of the three accompanying videos, which together illustrate data collection and storage, processing including predictive models, and action recommendation in an automated mode by controlling the hive actuators.

Acronyms and Abbreviations

Acronyms and Abbreviations	Definition
CDW	Core Data Warehouse
CLI	Command-line interface
DSS	Decision Support System
DB	Database

Chapter 1: Central core software architecture

1.1 Operational requirements for the core data warehouse

The core data warehouse (CDW) is an on-hive infrastructure for convenient data collection, storage, processing, analysis and output to external systems. Operation requirements for the CDW include the following considerations:

- The system is deployed on the embedded on-hive processing unit. It has limited storage and processing capabilities, and is based on ARM CPU architecture.
- The system is capable of supporting and running components of variable complexity. This includes components ranging from threshold-based validation on sensory information and ending with fully-fledged modelling and simulation packages.
- The identified patterns of use are defined by extensive data-in operations (writing incoming values from various sensors) with subsequent data-out operations including a live stream of the latest values and access to aggregated data.
- Based on embedded hardware limitations the data retention periods are defined to be 7 days for raw unprocessed data and 30 days for pre-aggregated data. Note that the total development of a worker honeybee lasts approximately 21 days, representing one of the longer biological processes in the colonies. Thus a 30-day data retention enables analysis to consider data from the most recent cycle with some margin, even in the case of extended connectivity outages.
- The system has connectivity to the online services for data archiving, external data providers, inter-hive interaction scenarios and user interfaces.

1.2 Software architecture implementation details

The developed CDW employs a modular architecture, as shown in Fig 1.1. The architecture is designed to operate on an embedded Linux platform (see D3.1), and designed in general to function with internet connectivity to access remote services including augmented maps, archiving, other HIVEOPOLIS units, and user interfaces. The architecture is also capable of stand-alone functioning in the case of a loss of external connectivity.

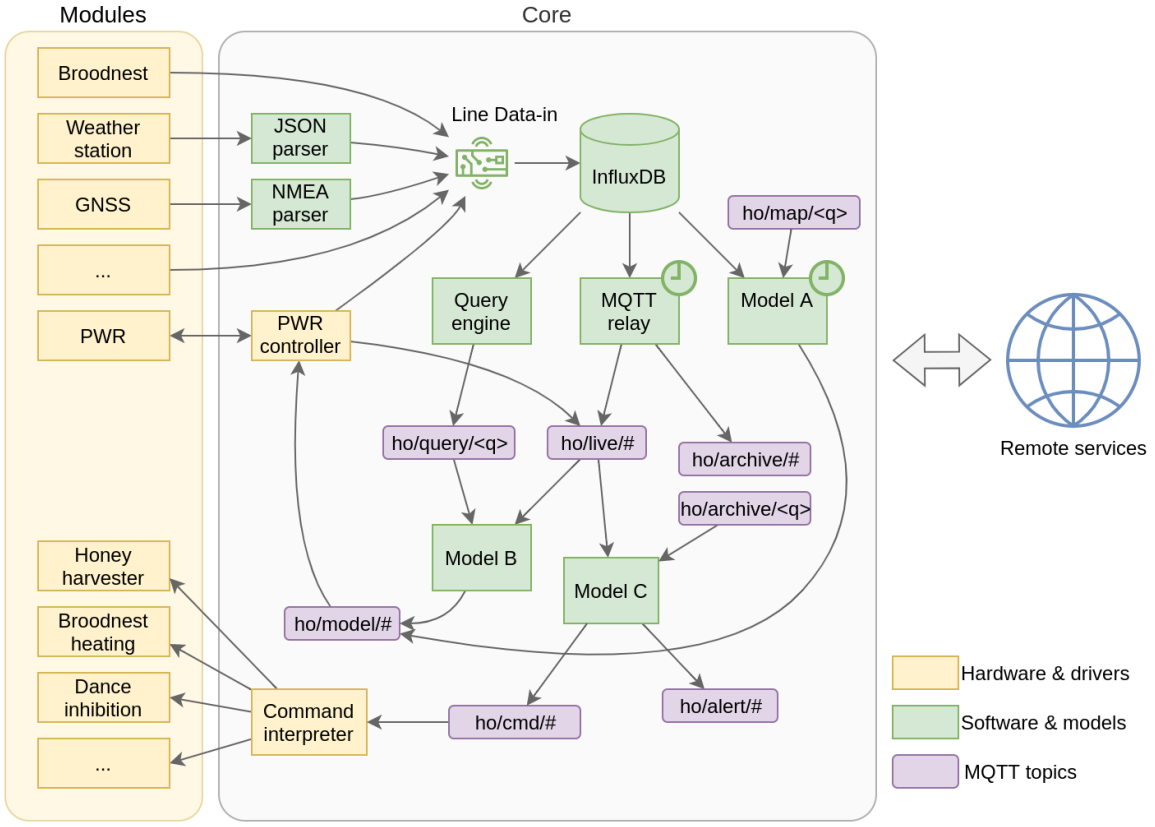

Fig 1.1 Architecture of the on-hive core data warehouse

MQTT protocol is used as a means of interaction between various components of the CDW, including data and command exchange. In addition, *MQTT* is used to implement a transparent and secure connection to the remote services via bridging and topic mapping (Fig 1.2). Our design makes use of multiple MQTT brokers: each HIVEOPOLIS unit has a local broker, which makes use of bridging in order to communicate with the remote services and other HIVEOPOLIS units.

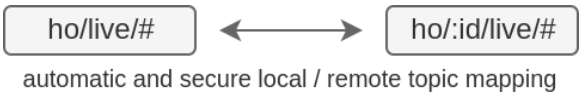

Figure 1.2 *MQTT* local-remote topic mapping

InfluxDB time series database is used as a central data storage with implemented data retention policies. The choice of a time-series oriented database matches well the stream-like nature of the sensory data from the various modules of the hive, as well as the remote services including the augmented maps (under development within WP1). Moreover, this organisation matches typical queries made by the models, which perform analysis or predictions for specific periods of time. The *Line Data-In* component acts as a proxy between various hardware modules and the database and provides an interface for convenient data input via *line protocol* (native for *InfluxDB*).

Two types of data-out mechanisms are implemented:

- *MQTT relay* periodically (once a minute) fetches most recent values from the data storage and propagates them to the dedicated MQTT topics for all subscribers. It implements live streaming of values from the data storage. This mode of data access is crucial for certain models such as those continually detecting abnormal states in the bee colony or in the hive infrastructure.
- *Query engine* provides access to the data on demand via a set of predefined queries. This mode of data access is more suited for models that are executed infrequently, and potentially human stakeholders who may perform ad-hoc inspections.

Specific algorithms and models are implemented in a form of stand-alone services, integrated into the hive core via CDW infrastructure. Chapter 2 provides more details, including models employing periodical and query-based data access, with online (streams of live data, acquired from on-hive sensors and systems, as well as remote services) and offline data sources (retrieving historical values from the data warehouse, in raw or aggregated form). These models form the kernel of the decision-support system (DSS). Depending on the pipeline of data processing, analysis, and predictive modelling, recommended actions can be made. While many conventional DSSs generate recommendations for a human operator to then make the final decision and actions, here, the HIVEOPOLIS system can be configured to act directly on some of the recommendations, by using actuator mechanisms in the modules. One model sequence, illustrated in the video described in Sec 3.3, exemplifies one automated pathway through the DSS.

All CDW software components, models and algorithms are deployed as *Docker* containers and are orchestrated via *Docker Compose* tool. This usage of containers ensures consistent deployment and software dependency support, including where different models or other components might require conflicting versions or packages.

Chapter 2: Models executed in the core

The models are key elements of the core data processing architecture described in Chap. 1. In our conceptualisation, data can be processed in a streaming fashion (immediately upon receiving new data or events from modules), periodically, or on-demand (e.g. from a beekeeper, via a user interface). The software blocks that perform this data processing are considered as *models* that can be chained together, a framing that permits the integration of new models as they are developed. This chapter describes several models that we used as examples during the architecture development. Some have been developed in other research and adapted here, while others have been developed specifically for the decision support tasks of the HIVEOPOLIS project. Each subsection below outlines the motivation, provenance, and functionality of the integrated models.

The models vary significantly in their complexity, ranging from if-then rules (see Sec 2.2) up to multi-agent simulations (see Sec 2.6). To cater flexibly for all these needs - but especially the most complex models - we have established a regime that breaks the model into preparation and post-processing stages besides the main computational stage. All communication to parameterise models is done through files, which are produced during the preparation stage, and enable database queries to be verified more easily, using the query engine. Each stage can in principle be executed in independent containers, which enables dependency management and code distribution. This regime is outlined in Fig. 2.1, illustrated with a NetLogo-based colony dynamics model developed within WP5 (see D5.2, Chap. 4).

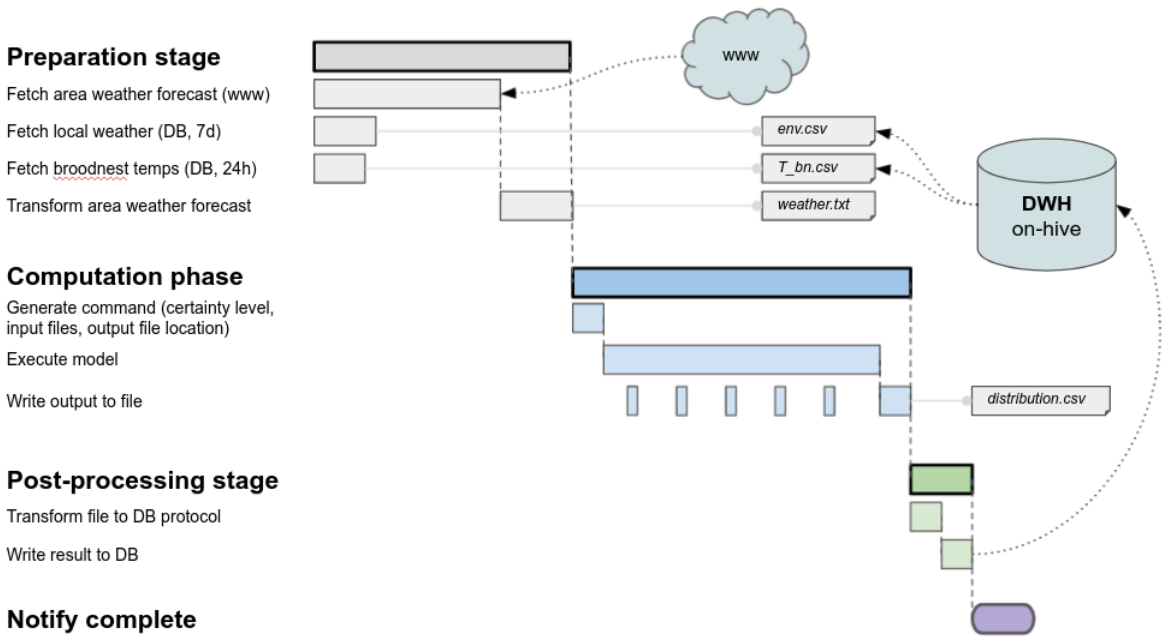

Fig 2.1. Scheduled execution of models, illustrated with a colony dynamics model. It is designed as a three-stage process, and communication to the model is via one or more files containing model input data or model output results. Each stage can have one or more sub-steps depending on the complexity of the model.

2.1 Recognition of colony states

During the annual bee colony cycle, different states (e.g., brood rearing, intensive nectar flow, broodless periods,, swarming) that affect colony development in a positive and negative way, can be observed during various seasons (Zacepins et al. 2015). Besides that, a collapse of a colony can also happen that breaks and (in worst case) ends the colony's life cycle. These states can be recognized by monitoring specific parameters, like temperature inside the beehive, weight of the colony, sound and also making video recordings of bee activity or brood development.
Several models focusing on temperature and weight data were integrated within the CDW to recognize typical states (brood rearing (spring, summer), nectar flow, broodless period (autumn, winter)) of the honeybees colony during specific seasons. The system can therefore recognize abnormal activities, like swarming, death of a colony, and the possibility of some kinds of disease. The recognition of these states are a crucial part of the DSS that provides recommendations for the beekeeper if an abnormality is detected and beekeeper's action is required.

To actively analyse temperature data inside the hive, a model based on fuzzy logic (Kviesis et al. 2020) was integrated within the embedded environment. The model is capable of classifying the bee colony state in three categories – "ok" (the state is as it should be, based on the temperature data), "extreme" (the state can't be exactly determined, therefore additional models need to be run), "colony death" (at different times of the year the detection of a colony death by temperature data can vary, therefore this state might fall also within "extreme" category). Basically, the integrated fuzzy logic model, in this case, can be considered as a filter to check for possible problems in an early stage.
The model takes 5 parameters (temperature (in/out), current month, temperature difference between in/out and temperature difference inside the same hive (change during a specified period) as inputs (retrieved from the on-hive database), processes them via the defined membership functions and runs through an inference engine (applying defined rules/knowledge). The result of this model is considered as a simplified health assessment (an assumption) of the colony in % (ranging from 0 to 100%, where 0-40% corresponds to "colony death"; 40-70% corresponds to "extreme" state; above 70% - "ok" state). The assessment scores are not intended to directly quantify the fraction of healthy bees in a colony, but rather, provide a numerical scale that is straightforward to interpret and use in further decision-making. The model was integrated using Python programming language and the fuzzy logic toolkit *scikit-fuzzy* (https://pythonhosted.org/scikit-fuzzy/overview.html).
The model is run on every new measurement and depending on the output, additional models are scheduled to run, e.g., in case the output points to "extreme" state, a swarming model is then triggered (if the season is summer).

The swarming model (neural network) was integrated using the Python programming language and TensorFlow (an open source library for artificial intelligence) (https://www.tensorflow.org/). In order to run the trained model in an embedded environment

(with constrained resources), it was converted into an TensorFlow Lite model. The model expects 60 data points (corresponding to temperature readings for a one hour period) as inputs and provides a prediction of a possible swarming event. The necessary data (temperature readings) are retrieved from the on-hive database.

Additionally, to make the swarming detection more robust, a model based on weight measurement analysis was also integrated within the CDW. The detection is a simple comparison between two measurements to determine when there is a significant drop in weight (see Fig. 2.2).

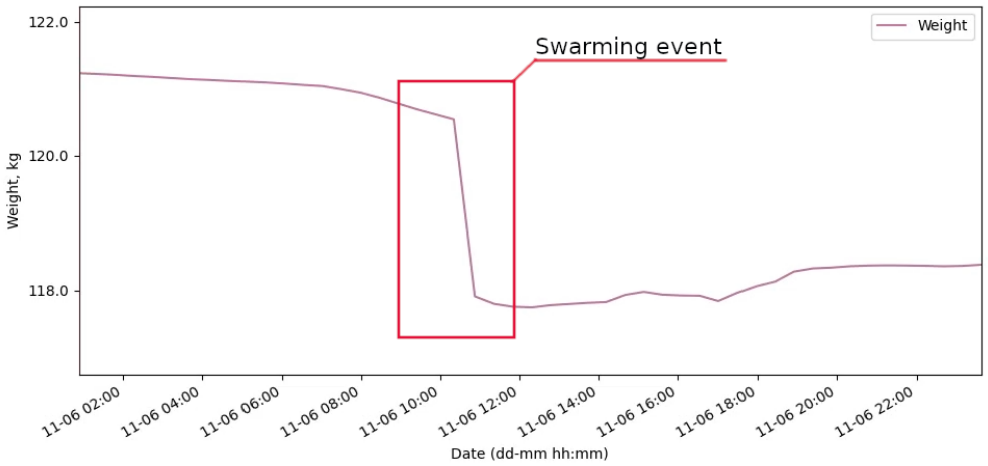

Fig. 2.2. Swarming observed in weight data

Another model using weight data analysis was developed to recognize active foraging. This model was also based on weight measurement comparison together with weight trend determination (weight is increasing, decreasing, fluctuating) by applying a linear regression model. Such recognition provides information to a beekeeper about the colony's production and if foraging is even happening.

In the field of precision beekeeping, when focusing on colony state detection, the DSS can automatically make some of the decisions (Zacepins et al. 2015) or provide suggestions/recommendations to the beekeeper. Therefore, this type of DSS can be considered as an active DSS (provides decision suggestions or solutions), based on the classification provided by Gebus & Technica (2006).
Depending on the recognized states, the DSS provides the following recommendations:

- swarming - inspection is needed (depending on the amount of bees that left the hive, the colony strength can be significantly weakened);
- death of colony - inspection is needed (human-level expertise is required to identify the possible cause (e.g., disease/parasites, robbing));
- unknown state (cannot be determined by models) - inspection is needed;
- active foraging - information about the amount of collected honey and decision to harvest;
- “Ok” - if the state is as it should be, then the recommendation is to “do nothing”.

The DSS infrastructure can be extended to accommodate further states recognised by models developed elsewhere (e.g. relating to diseases, population dynamics, or interactions with other hives and pollinators). Each additional state requires an action recommendation, which we develop in conjunction with relevant stakeholder groups.

2.2 Broodnest safety model

This model was developed for the specific practical use-case manifested during biological experiments. To investigate augmentation of the temperature regulation within the broodnest, such that the broodnest of a hive maintains a specific temperature, heaters are being operated. To avoid overheating in case a heater control block fails, the overheating detection model was developed as a safety measure. In case of malfunction, the model output is used to raise appropriate alert notifications to the user.
In essence, the model is a threshold check on broodnest temperature values, but together with other interconnected components it demonstrates data flow within the CDW and its interaction with remote services (ref T1.4) for alert notifications to the end users.

2.3 Colony collapse model stub

The probability of colony collapse increases if stressful activities are applied to the colony without proper planning. One of such activities is honey harvesting, which, when performed carelessly, leads to the colony collapsing in a relatively short time. Thus it is crucial to evaluate colony collapse probability before performing the honey harvesting.
The details about the colony collapse evaluation model are described in D5.2. For demonstration purposes a model stub was developed which maps current colony weight to collapse probability. The modular architecture of the CDW ensures loose coupling between components and the developed stub can be replaced with a real model without any changes in the rest of the system.

2.4 Accumulated weight model

This model was developed as one of the components for the D3.3 demonstrator (see Sec 3.3) and acts as an example of intermediate multi-stage data processing. The model estimates the accumulated weight of the bee colony over a specified period of time which directly correlates to the amount of harvested honey.

The model compares recent weight measurements to the historical values obtained via request-response protocol from other CDW components, such as Query Engine and, eventually, the on-hive database. Modelling results are not provided for end users directly, but instead are used by other CDW components for further processing and interpretation.

2.5 Honey harvesting procedure

The harvesting procedure is a sequence of steps performed manually by a beekeeper or automatically by a robotised beehive in order to harvest collected honey. For D3.3 demonstration purposes an illustrative automated harvesting procedure is implemented

which uses intermediary inputs from other CDW models and components to perform appropriate actuator control.

The procedure checks preconditions on the following inputs:

- Colony collapse probability is below a threshold to ensure that after harvesting the colony will survive for an extended period of time.
- Estimated amount of harvested honey is above a threshold to ensure that there is enough honey to actually harvest.
- Power level is high enough to perform the procedure from start to end.

The procedure consists of two activities: 1) evacuation of bees and 2) manipulating combs for fractional harvesting. In addition, several safety measures are implemented such as unexpected power level drop and raised colony collapse probability as depicted in the activity diagram (Fig 2.3).

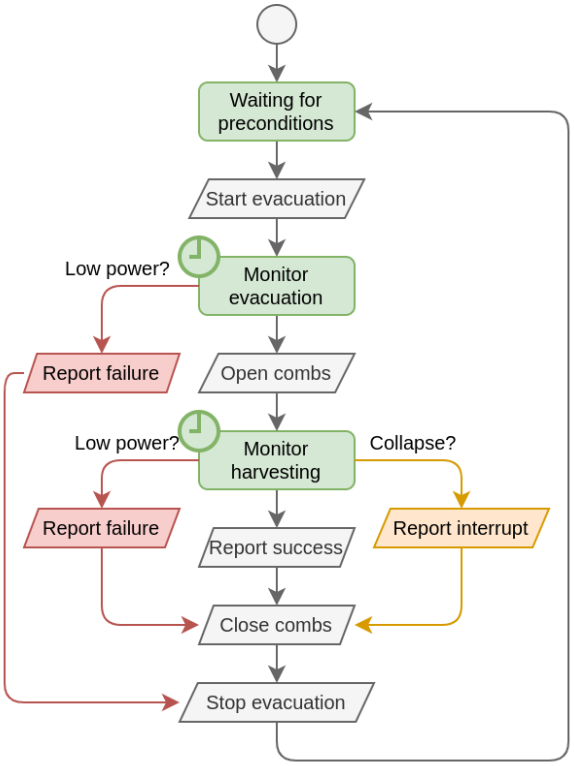

Fig. 2.3. Automated harvesting procedure.

The procedure maintains its internal state and upon reaching defined preconditions starts the harvesting process and sends corresponding commands to appropriate actuators. The results of the process (successful or not) are reported for further use, e.g. for notifying end users.

2.6. Beehaveopolis colony dynamics model

The Beehaveopolis model aims to describe population dynamics of a bee colony, including the population dynamics and foraging dynamics, and takes inputs from weather, and optionally population distribution estimates. This model extends Beehave (Becher et al., 2014) to include actuators from the HIVEOPOLIS systems, and can be used to make predictions regarding the future colony size, for example under differing scenarios of actuator utilisation. It is described further in deliverable D5.2.

We use a preparation stage to retrieve data from the on-hive database, which includes information based on brood nest size estimates derived from thermal sensor data. The model can also take weather information as input, gathered from the augmented map query service. This data channel is already tested, and will be incorporated into Beehaveopolis as part of the modelling efforts elsewhere, including the work on ecosystem hacking in WP1.
This model is fairly computationally expensive, relative to other models described above, with each execution taking 2 minutes and using up to 1GB RAM on the embedded platform.
On the other hand, the model inputs change only on a daily basis. Thus, to examine a handful of scenarios on any given day is an insignificant load. Setting resource limits with docker allows us to ensure that lower-priority tasks like these do not consume too much and cause bottlenecks for other tasks.

Chapter 3: Description of demonstrator video segments

This chapter describes three videos that demonstrate the functionality of the data warehouse and the embedded decision-support system. The first uses a very lightweight, live-streaming model that identifies abnormal conditions and illustrates remote service infrastructure. The second uses a rich colony state detection model that provides several different recommendations depending on the detected state. The third example uses a combination of data processing models to identify states, and illustrates how the core software can be used to execute automated procedures that are triggered by DSS recommendations. Each subsection below describes the demonstrator video, and includes a link to the video itself.

3.1 MS6-A demonstrator: Threshold-based broodnest safety check

Link to video: https://youtu.be/8LjRFORWmk0

The video illustrates how a basic safety check model triggers an alert when overheating is detected, via a Slack notification endpoint. The example implements:

- Integration between embedded and cloud data components
- Information exchange via MQTT protocol
- Threshold-based model detecting abnormal broodnest state
- Alert notifications via Slack messenger.

Specific components shown in the demonstration example are marked on the figure below (Fig. 3.1).

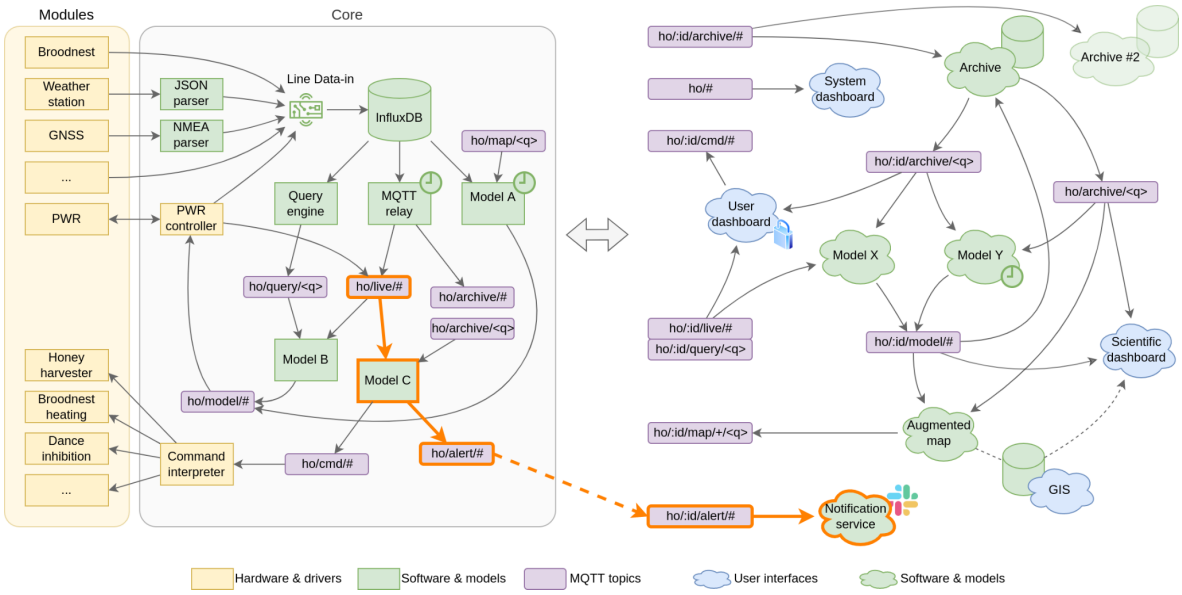

Fig. 3.1. The HO data architecture components used in the broodnest safety check, highlighted in red.

An alert delivered to the Slack channel is shown in Fig 3.2.

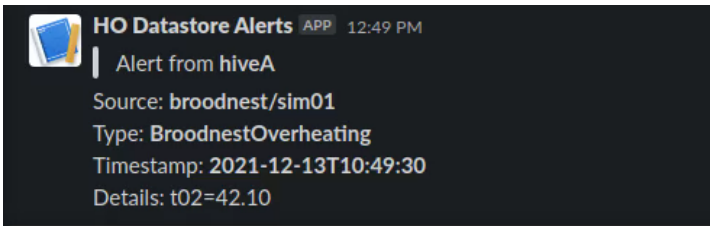

Fig 3.2. Example alert, delivered via a dedicated Slack channel

3.2 MS6-B demonstrator: Bee colony state detection

Link to video: https://youtu.be/qVUAC5iilk4

The demonstrator video presents:

- automatic recognition of different honey bee colony states: *Ok, swarming, active foraging*;
- information exchange via MQTT protocol;
- alert notifications for specific bee colony states.

The components involved within the colony state detection process are depicted in Fig. 3.3.

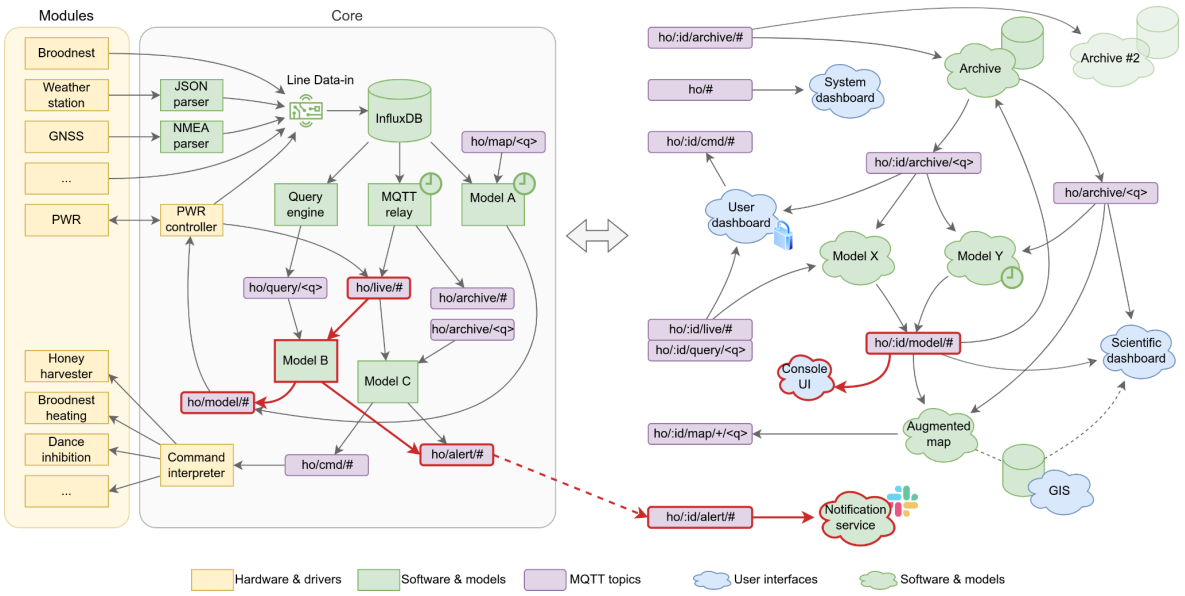

Fig. 3.3. The HO data architecture components required for bee colony state detection are emphasised in red.

To demonstrate the DSS recommendations, a simple command line interface (CLI) was developed to show the user the current status of a bee colony and recommended actions, if any (Fig. 3.4) (a notification was sent to a dedicated Slack channel as well).

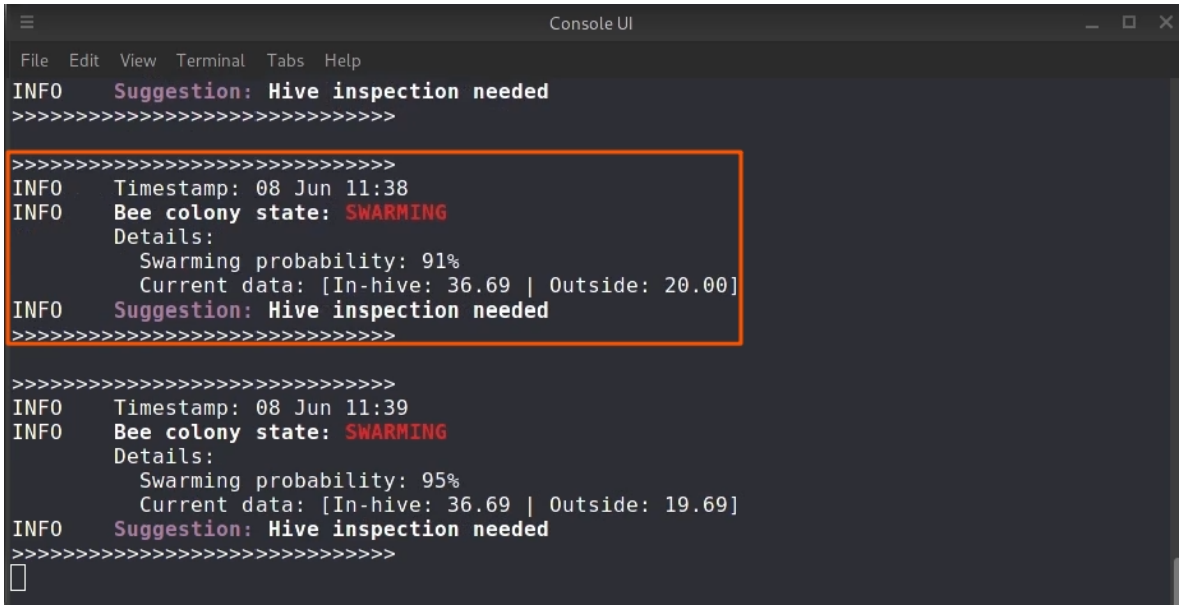

Fig. 3.4. A simple CLI to demonstrate a DSS recommended decision in case of colony swarming

3.3 D3.3 demonstrator: Honey harvesting procedure

Link to video: https://youtu.be/YJrdWQsljV0

The video illustrates the complete data analysis cycle in the core data warehouse: interconnected components implement closed-loop decision making and actuator control. Synthetic honey harvesting process is selected as a realistic use-case. Among other features it implements:

- Orchestration of multiple interconnected models and algorithms
- Data collection from hardware sensors
- Automated decision making process
- Actuator control according to the provided decisions.

Specific components of the demonstration example are depicted in Fig 3.5.

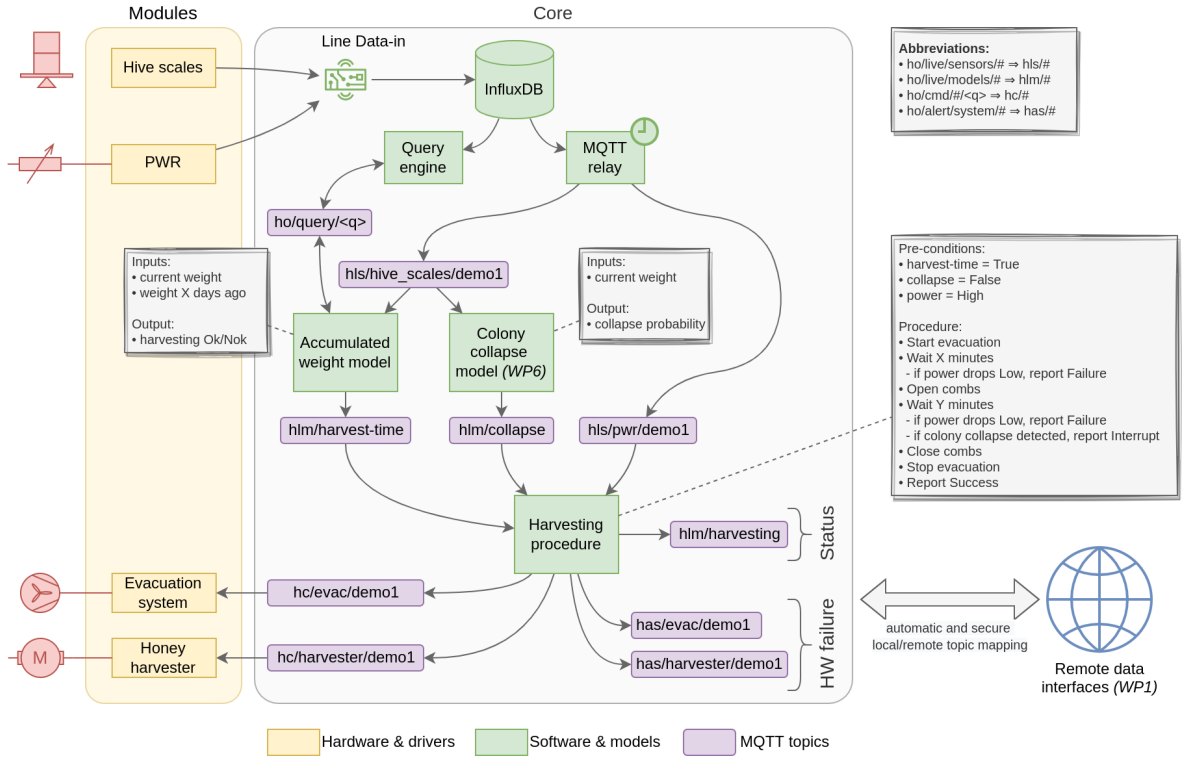

Fig 3.5. Elements of the software and hardware system used in the honey harvesting demonstration video.

The hardware setup used for the honey harvesting procedure demonstration is shown in Fig 3.6.

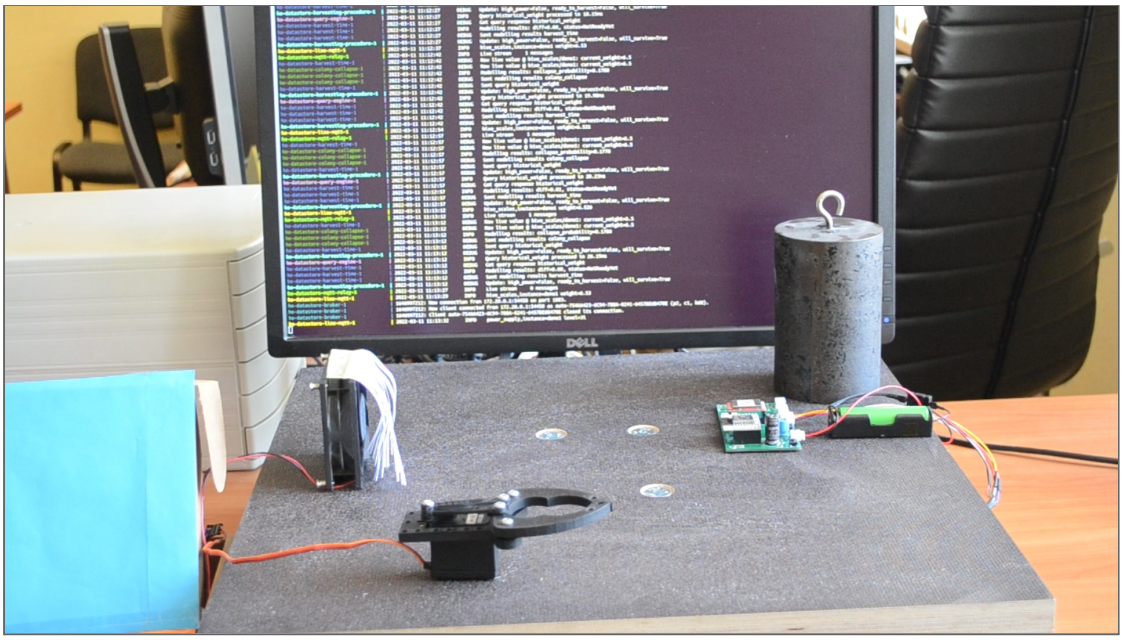

Fig 3.6. Hardware setup used in the honey harvesting demonstration video.

References

- Becher, M. A., Grimm, V., Thorbek, P., Horn, J., Kennedy, P. J., & Osborne, J. L. (2014). BEEHAVE: a systems model of honeybee colony dynamics and foraging to explore multifactorial causes of colony failure. *Journal of Applied Ecology*, 51(2), 470-482. doi:10.1111/1365-2664.12222

- Gebus, S., & Technica, C. (2006). *Knowledge-based decision support systems for production optimization and quality improvement in the electronics industry*. PhD Thesis, Faculty of Technology of the University of Oulu. 124pp.

- Kviesis, A., Komasilovs, V., Komasilova, O., & Zacepins, A. (2020). Application of fuzzy logic for honey bee colony state detection based on temperature data. *Biosystems Engineering*, 193, 90-100. doi: 10.1016/j.biosystemseng.2020.02.010

- Zacepins, A., Brusbardis, V., Meitalovs, J., & Stalidzans, E. (2015). Challenges in the development of Precision Beekeeping. *Biosystems Engineering*, 130, 60-71. doi: 10.1016/j.biosystemseng.2014.12.001