www.openswarm.eu

Call: HORIZON-CL4-2022-DATA-01

Type of action: RIA

Grant agreement: 101093046

Deliverable 4.2: Swarm Programming User Interface (UI) Wire Frame, User Guide and Software Architecture

Work Package 4: Energy aware swarm programming

Task Lead: SIG

WP Lead: UOS

This project has received funding from the European Union's Horizon Europe Framework Programme under Grant Agreement No. 101093046.

Funded by the European Union

Document information	
Author(s)	Markus Sauer, Łukasz Zalewski, José Quevedo, Saurabh Narayan Singh, Jochen Nickles, Franz Zeilinger, Genki Miyauchi, Mohamed S. Talamali, Roderich Groß
Reviewers	Sam Michiels
Submission date	31-Oct-2024
Due date	31-Oct-2024
Type	R
Dissemination level	PU

Document history

Date	Version	Author(s)	Comments
30-Jun-2024	01	Markus Sauer	Preview
31-Oct-2024	02	Markus Sauer	Deliverable

Table of Contents

Executive Summary

This document details the work conducted within the scope of Task 4.2 towards the realization of a Swarm Programming User Interface (UI). The deliverable considered inputs from state of the art technological developments including swarm-human interaction, swarm programming and behaviour modelling, as well as existing low-code solutions and its application in industry. This review was complemented with relevant OpenSwarm user stories from selected PoCs and testbeds, and an in-depth analysis of the associated characteristics and requirements. Taking all these inputs into consideration, a reference architecture for a Swarm UI has been proposed, extending the original scope from Programming to also include Swarm Engineering and Operations. The task also involved the evaluation of relevant software artifacts and the implementation of different UIs targeting specific OpenSwarm applications and testbeds. The results pave the way for the realization of the proof-of-concepts in WP6. Moreover, the proposed reference architecture and detailed functional decomposition are expected to be a major contribution beyond the original scope of the task and find applicability beyond the scope of OpenSwarm, contributing and providing significant value to the wider distributed systems and swarm communities.

Introduction

Swarm systems are inherently complex, characterized by the collective behaviour of numerous distributed interacting agents. The potential of these highly distributed collaborative systems has triggered the emergence of novel applications, each with specific requirements and expectations from the system. Consequently, significant efforts have been made to model the behaviour of swarms and to provide innovative ways for swarm programming and human-swarm interactions.

In the context of OpenSwarm, and in alignment with the Scientific Objectives SO1, SO3 and SO4, the need was identified to design and implement tools that enable operators to intuitively program swarm level behaviours that are fully adaptive, while respecting formal specifications. A fundamental part of this work, to be conducted in WP4/Task 4.2, is the development of an intuitive user interface, which is presented in the current deliverable. The resulting interface should facilitate the following processes: (i) **swarm programming**, allowing operators to provide high-level instructions to the devices (e.g., block-based visual programming) expressing the collective behaviour of the swarm (i.e., an interface for non-programming experts); (ii) **swarm management**, providing comprehensive information about the devices by interfacing with sources of monitoring information; and (iii) **control swarm execution**, allowing to upload programs into Virtual Machines, and controlling their execution lifecycle.

This document reports the work done in T4.2, from the initial review of the state of the art to the implementation of OpenSwarm UIs targeting specific applications and scenarios.

The remainder of this deliverable is structured as follows:

- Chapter 2: Provides a detailed description of the methodology that guided the work conducted in this task.
- Chapter 3: Presents the related work, identifying both academic and industrial efforts that are relevant in the context of the Swarm UI.
- Chapter 4: Motivates the Swarm UI concepts and presents relevant user stories.
- Chapter 5: Characterizes Swarm Environments and derives the main associated requirements.
- Chapter 6: Focuses on the Swarm UI design, providing key architectural requirements, and detailing the reference Swarm UI architecture.
- Chapter 7: Leaps into implementation aspects providing an evaluation of relevant software artifacts and presenting different OpenSwarm UIs targeting specific applications and scenarios.
- Chapter 8: Concludes the document.

Relation to other Tasks and Deliverables

This task is highly entangled with other tasks/work packages from the project. The more direct relations can be summarized as follows:

- T1.1: The technical requirements specified in T1.1 were considered when designing the Swarm UI reference architecture.

- T1.2: The context of the Swarm UI takes into account the architectural considerations from T1.2, ensuring not only that the UI fulfils the OpenSwarm expectations but also that relevant external components are included in the system design.
- T4.3: The Swarm Compiler is one of the key external components in the Swarm UI reference architecture which compiles the information provided by Swarm UI users at different levels into deployable units that can run in the underlying Swarm Devices.
- T4.4: The Swarm UI provides the necessary abstractions for controlling the overall lifecycle of the Virtual Machines for Swarm Control and Learning.
- WP6: The Swarm UI reference architecture is heavily influenced by user stories coming from WP6, and the resulting OpenSwarm UIs are expected to play a pivotal role in the execution of WP6.

Key Performance Indicators (KPIs)

T4.2 targets the OpenSwarm KPI ***"KPIT11 Size of the User Test Group Giving Feedback on the UI"*** where the target size of the Groups should be larger than ten. This is addressed in the context of the deliverable as follows: In the first phase of the project the need for a common ground for a Swarm UI was identified due to heterogeneous implementations needed by the different use cases of a swarm and the OpenSwarm PoCs specifically. Thus, this led to the development of a reference architecture which was jointly elaborated and peer reviewed by a group of more than ten experts and researchers. The specific implementation and case study for a swarm UI as described in Swarm Operations with Multiple Operators section has been tested with a test group of 52 users. Thus, T4.2 has already reached the KPIT11 as defined in the Description of Action of OpenSwarm. Still, we expect further evaluations with similar test group sizes during the implementations in WP6 related to the PoCs and testbed.

Technology Readiness Level (TRL)

This section summarizes the TRL developments and expectations for Swam Programming UI, concretely on its evolution within the scope of T4.2. The work in this task reflects the evolution of the general Swarm UI approach from TRL1 to TRL3 - starting with a complete analysis of the foundations and problems associated with a Swarm UI (TRL-1), which was then leveraged to establish solid technological concepts behind the Swarm UI Reference Architecture (TRL-2), and ultimately examined in various experimental solutions partially implementing relevant Swarm UI functionalities (TRL-3). Additional Swarm UI developments and higher TRLs should be achieved during the execution of the relevant PoCs from WP6.

List of Publications

- G. Miyauchi, Y. K. Lopes, and R. Groß, 'Sharing the Control of Robot Swarms Among Multiple Human Operators: A User Study', in 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Oct. 2023, pp. 8847–8853. doi: 10.1109/IROS55552.2023.10342457.
- G. Miyauchi, R. Groß. 'Human-Multi-Robot Interaction', Springer Encyclopedia of Robotics, to appear

Methodology

The work developed in this task, and detailed in the current document, followed the methodology as described in Figure 1.

An initial part of the work focused on establishing strong foundations of a Swarm User Interface, intentionally targeting the broader concepts (i.e., not limiting it to programming aspects). These foundations include an extensive state of the art analysis (e.g., swarm-human interaction, swarm programming, low-code programming) complemented with representative user stories and motivations.

This information was then leveraged for a problem analysis, characterizing the Swarm UI from different perspectives, and deriving at every step the associated requirements. From this requirement a set of eight key architectural requirements were defined, elucidating how each should be tackled by the architecture. Ultimately this analysis led to the definition of a Swarm UI reference architecture including the definition of the overall system context and the main functionalities.

Taking into consideration the expected functionalities, relevant software artifacts were evaluated. Given the broad scope of the reference architecture and the heterogeneity of applications foreseen for the OpenSwarm proof-of-concepts, the development of the Swarm UI was conceived as a set of OpenSwarm User Interfaces providing specific functionalities and targeting specific proof-of-concepts.

The overall outcomes of this task will be further developed and validated during the execution of the different proof-of-concepts from WP6.

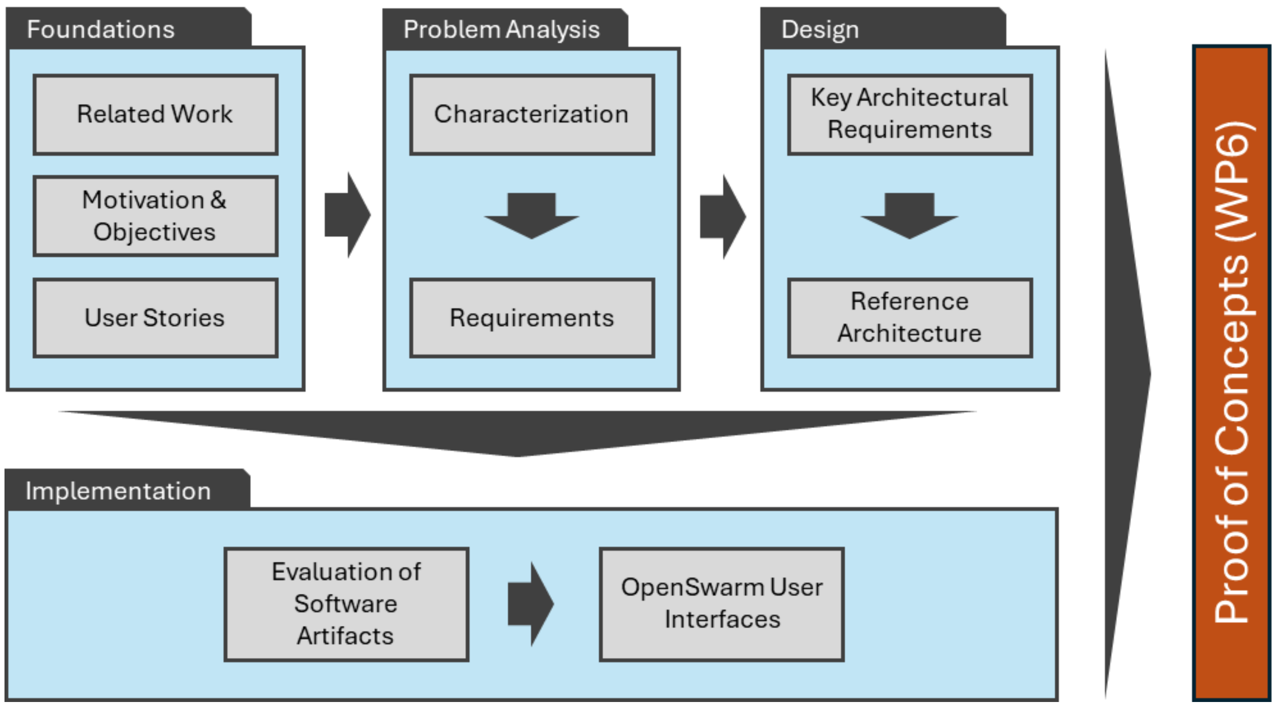


Figure 1 Methodology Overview

Related Work

This section provides an overview of works from the literature related to swarm programming interfaces. It begins by providing a review of the state of the art in human-swarm interaction, addressing relevant human factors including cognitive complexity and situational awareness as well as common control and interface design methods and interaction modalities. This is followed by a review of the state of the art in swarm programming and behavioural modelling, a topic that is further expanded as part of Deliverable D4.3 (Functional Specification, architecture and performance of the OpenSwarm Compiler). Finally, a comprehensive review of the state of the art in model-based and low-code design paradigms in industry is given.

State of the Art in Human-Swarm Interaction

To deploy robot swarms in the real world, it is important to consider how humans can supervise and interact with them when necessary. However, the distributed nature of robot swarms can pose several challenges for a human operator. This has led human-swarm interaction (HSI) to become an active research topic in recent years.

Cognitive Complexity

Cognitive complexity is an operator's "cognitive effort" to manage a group of robots performing independent tasks [1]. The concept originates from *computational complexity* in computer science, where it is used to define the time required for an algorithm to solve a problem that scales as the input size increases. A cognitive complexity of $O(1)$ represents a human operator interacting with a single robot. A complexity of $O(n)$ represents an operator interacting with $n$robots, so the time an operator spends on each robot reduces as n becomes larger. The cognitive complexity can increase super-linearly, for example, quadratically (i.e. $O(n^2)$) if the operator must also handle the interactions between individual robots. This is problematic for human operators interacting with robot swarms since the number of robots involved can range anywhere between a few dozen to over a hundred; it is not realistic for an operator to interact with every robot in the swarm without being overwhelmed. The fan-out model is a related concept which was proposed by [2] to describe and predict the number of robots a single operator can control simultaneously. The presence of a large number of robots can also provoke an operator's psychophysiological state to cause stress and anxiety [3], further increasing the likelihood of being overwhelmed by the swarm.

Instead of having to interact with individual robots, a human operator's cognitive complexity can be significantly decreased if the interaction is abstracted to the swarm-wide level. This allows a human operator to consider the swarm as a single entity, which reduces the cognitive complexity to $O(1)$. However, when the robot swarm splits into multiple subswarms, a similar situation arises where the human operator is again faced with the problem of interacting with potentially many subswarms [1].

So far, we have assumed that the robots' productivity increases when the human operator is able to spend more attention on each robot and decreases when the robots receive less attention. This concept is known as *neglect tolerance* [4]. On the other hand, robot swarms exhibit emergent behaviour, which usually requires some time before it can be observed. *Neglect benevolence* is the idea that some robot swarms may perform better when they are neglected (i.e. no interventions) by the operator for a certain period [5]. Nagavalli et al. [6] showed that upon observing a swarm's behaviour, human operators can learn to identify the best moment to send an input to the swarm. While this highlights the importance of operators to interact with robot swarms at the most effective moments, it also suggests that it is not necessary to be able to interact with every robot in order to effectively work with swarms.

Situational Awareness

Situational awareness refers to a human operator's comprehension of the current situation and projection of possible future outcomes [7], [8]. It is influenced by how information is presented to a human operator and how the operator makes use of the obtained information to make assessments of the current situation. While maintaining a good level of situational awareness is important, unlike traditional robotic systems, it is often difficult to achieve this when interacting with robot swarms due to their highly distributed nature.

When the global state of the swarm is accessible, interfaces that can show the predicted behaviour of the swarm can improve the human operator's situational awareness [5]. However, it may not always be possible to obtain such global state information when deploying robot swarms in real-world environments (e.g. a global map showing the positions of the entire swarm). Kapellmann-Zafra et al. [9] studied whether a human operator can aggregate robots when provided access to only a local camera view and proximity sensor readings from a single randomly selected robot. Under the effects of such limited situational awareness constraints, operators struggled to aggregate the robots effectively, even when they had previously observed their global dynamics. However, upon receiving sufficient training, operators became familiar with the dynamics of the robot swarm and increased their performance under the presence of the aforementioned situational awareness constraints.

Level of Autonomy (LoA) is one factor that affects a human operator's situational awareness. LoA is a 10-point scale that defines a range of autonomous behaviours that a system could exhibit during human interaction [10]. In swarm robotics, a low LoA would mean the robots often wait for inputs from the operator, resulting in an increase in the workload experienced by the operator. A high LoA would mean the robots are fully autonomous and require minimal input from the operator. While a high LoA may reduce the operator's workload, the operator gets detached from the rest of the system, which can potentially lead to the degradation of the operator's knowledge and experience related to the task. Instead of locking the swarm to a fixed LoA, [11] proposed a mixed-initiative system, which allows the swarm to flexibly change its LoA depending on the current task and human preference.

It is important for swarm designers to consider how and what information should be presented to the human operator and what LoA the swarm should exhibit depending on the specific use case they are being used in. Interface designs that control the type of information displayed to an operator are discussed in the subsequent Interface section.

Control Methods

In recent years, several methods have been proposed for dynamically controlling robot swarms. These methods aim to minimise the complexity required to control a large number of robots by departing from the traditional one-to-one teleoperation style of control [12]. Kolling et al. [1] suggested four categories for classifying existing control methods, namely behaviour selection, leader influence, environment modification, and parameter setting.

Behaviour selection. This method provides an operator with a library of swarm behaviours to choose from. The operator controls the swarm by switching between the behaviours to achieve the desired outcome. [13], [14] investigated an operator choosing from a set of discrete behaviours (e.g. Stop, Random, Rendezvous) to perform a foraging task. The behaviours had to be given to robots in the swarm at the right moment to maintain effective coverage over an area. Different sets of behaviours were examined by [15] and combining simple behaviours into plays by [16]. Behaviour selection has also been shown to control hub-based swarms [17]. Behaviour selection is one of the simplest methods to control a robot swarm. However, it requires the operator to have a thorough understanding of the set of available behaviours and their expected outcomes.

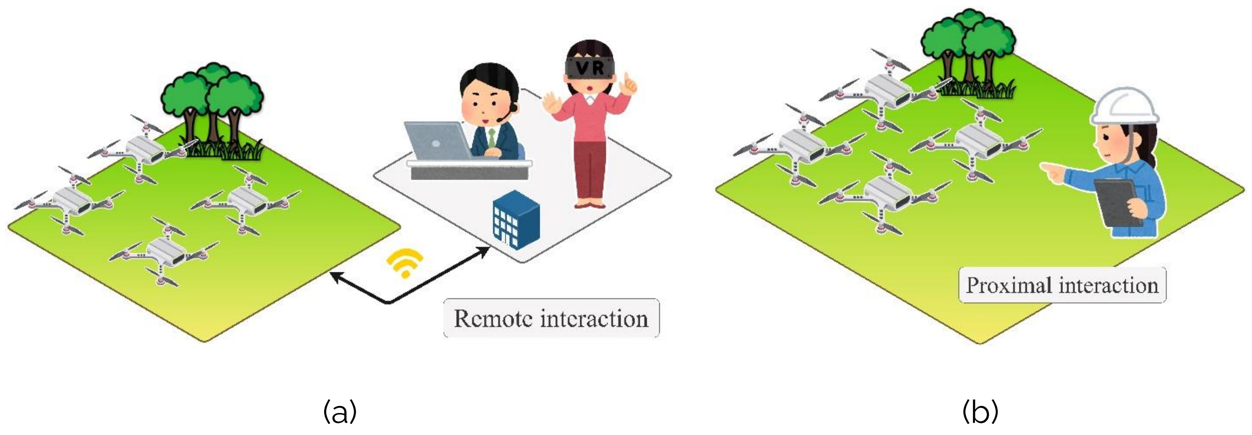


Figure 2. Illustration of the two interaction types. (a) Remote interaction - a human operator interacts with the robots from a remote location. (b) Proximal interaction - a human operator interacts with the robots while being present within the operational environment.

Leader influence. The operator directly controls one or more leader robots, which affects the behaviour of other neighbouring robots. Walker et al. [18] examined the leader-follower model based on Couzin's flocking model [19], where the followers moved synchronously with the leader. The use of multiple dynamic leaders showed even greater influence over the robot swarm [20], [21]. Instead of attracting follower robots, leaders can also act as repulsive forces, as seen in the predator approach investigated by [22]. Therein, the operator switched between attractive and repulsive forces to navigate the swarm. During proximal interaction (see Figure 2), a human could directly be perceived as a leader by the robots [23]. Leader influence has also been considered a simple method to control swarms due to its similarities with traditional teleoperation-style control. An important point to note is that the controllability over the swarm is limited by the range of influence of the leader robot.

Environment modification. The operator indirectly controls the behaviour of robots by modifying the environment. An example of this type of control can be seen using real and virtual objects as cues. Obstacles [24] and attractive/repulsive beacons [14] were placed in the environment by a human to influence any robots that came close to them. If the robots are capable of sensing virtual cues in the environment, virtual walls [25] and artificial potential fields [26] may also be modified to influence the robots as well. Environment modification is highly scalable since it does not depend on the size of the swarm, though the indirect influence on robots makes the interaction less intuitive for operators to control.

Parameter setting. This method allows an operator to modify a set of parameters that govern the behaviour of a swarm, such as thresholds or probability values that trigger a change in the robot's behaviour. For example, to encourage more exploration during a foraging task, an operator may decide to decrease the parameter value that controls the likeliness of all robots to move in the same direction [17]. Instead of using raw numeric values, a set of parameters can also be used to exhibit different types of social-like behaviours indicating the robots' *Sociability*, *Conformity*, *Commitment*, and *Disposition* [27]. Among the four classes of control, parameter setting is the least explored control method due to its less intuitive nature of having to set parameters; the operator must understand not only the effects of modifying a single parameter but also the interplay between them.

Some human-swarm interaction systems may have two or more modes of control, which lets the human operator pick their preferred type of control [24]. When multiple human operators interact with a robot swarm, interference with each other's control should be kept to a minimum. Leader influence and environment modification usually have a limited range of influence and thus, can be considered as appropriate control methods for multi-human-swarm interaction.

Interface

Due to the distributed nature of robot swarms, understanding the state of the swarm and the progress of the current task can be difficult for a human operator who is simply observing the motion of individual robots. An important factor when designing a complex system is to improve its *transparency*, which is to provide useful information in a way that is easily understood by the operator interacting with it [28]. Interface designs that effectively display information about the swarm can help improve an operator's situational awareness, which is important for gaining human trust in the swarm system [28].

Interfaces that provide a global view of the environment displaying the terrain, the position of all robots and points of interest have been the most commonly explored designs in HSI. This type of interface that provides a ``bird's-eye view" was shown to be effective for interacting with robot swarms [14]. More recently, digital twin technologies have been used to interact with swarms. Jang et al. [25] created a digital twin of the swarm and allowed an operator equipped with a virtual reality headset to view and interact with the swarm from any preferred angle. The operator was able to pick and place a robot in the digital twin, which was translated into a command to move the robot to the specified position. While interfaces that provide a global view can give a complete picture of the current situation, many assume perfect communication between the interface and every robot, which may not always be possible to achieve in challenging environments.

Instead of simply displaying every robot in the swarm, various visualisation techniques have been explored to represent swarms differently. Previous interfaces have represented a swarm as an amorphous blob [29], an arrow that represents the direction and velocity of the swarm [30], and a heatmap that represents the swarm's area coverage [31]. The progress of tasks can also be overlaid within the interface [32], [33]. Many of these visualisations are intended to highlight certain properties of the swarm or task that are found to be useful for the operator. The choice of the type of visualisation depends on the type of information available and the context in which the swarm is being used.

Interfaces that assume a local perspective, for example, from the view of a single robot, are less commonly explored. Operators engaging in proximal interaction [23], [34], [35] can also be considered to have a local perspective, as their view of the swarm is restricted to their physical position in the environment. As situational awareness was shown to be lower when operators were restricted access to a global view [9], there is a need to explore how operators with a local perspective of the environment can effectively utilise a swarm that may travel beyond their field of view.

Modalities

Since the early days of human-swarm interaction research, many works have explored the use of different modalities other than using a monitor, mouse and keyboard to control and receive feedback from robot swarms. In this section, we look at two of the most actively studied topics; gesture- and haptic-based interfaces.

Gesture is used commonly by humans to express or reinforce their intentions [36]. A range of hand signals have been used to signal the swarm to perform various behaviours, such as to split, merge, and steer the swarm [37], [38], [39], [40]. Podevijn et al. [38] used a Microsoft Kinect to detect user gestures and mapped them to specific commands for the swarm. Experimental tests demonstrated that a user was able to split a large swarm into smaller groups and guide each subswarm to different goal locations. Pointing gestures have also been used to instruct one or more robots to move from one location to another [23], [34]. In the work by Gromov et al. [34], a user was equipped with two armbands, each consisting of eight electromyography sensors, to extract the 3-D vector represented by the configuration of the arm and to recognise the current hand gesture. The overall pointing gesture was combined with voice commands, which confirmed which robots the user wanted to control. Gesture-based interfaces have shown to be very intuitive even for inexperienced users and are an effective mode of control, especially during proximal interaction.

Haptic feedback creates a sense of touch by applying force or vibration to the user. Nunnally et al. [41] was one of the first to make use of haptics in HSI. A Phantom Omni device was used by the user to broadcast a force vector to influence the behaviour of the robots. The operator in turn experienced force via the Phantom device when there were obstacles near the robots. Tsykunov et al. [42] presented SwarmTouch, which informed the change to the formation of several drones using a haptic glove that vibrated the user's fingertips. Haptic feedback can also be provided to other parts of the body, such as the work by Abdi et al. [43] where an operator felt the swarm's centre of mass through a haptic vest. Zooids [44] and Ubiswarm [45] are table-top swarms that can be considered to use haptic feedback since a user directly picks up one or more robots to interact with them. In these systems, the user physically interacts with the robots rather than experiencing a device that emulates the sensation. These works show that haptic-based interfaces are well-suited for quickly providing feedback about the swarm as they can be "felt" by the user directly, though the user must be aware of the meanings of the different haptic signals.

The two modalities discussed in this section are both intuitive to humans as they are often based on or inspired by human-human interaction. Voice is another popular modality investigated in human-robot interaction, which is often included in multimodal interfaces that combines a number of different modalities [46], [47]. More recent studies have proposed to use swarms as embodied extensions to enhance human abilities [35], [48]. Other modalities explored in HSI include face engagement [46], [49] and brain-machine interfaces (BMI) [50].

State of the Art in Swarm Programming and Behaviour Modelling

In order to program the behaviour of a single robot, it usually requires a developer to understand the capabilities and functionalities of the robot and carefully write code using a particular programming language or technology stack. Programming the behaviour of a swarm of robots adds further complexity as the developer must consider not only the robot and the surrounding environment, but also the potential interactions between an arbitrary number of other robots that share the environment. This makes it challenging for a developer to intuitively program the behaviour of the swarm and to provide any guarantee that the swarm will converge to a certain behaviour. In this section, we provide existing software tools that have been proposed to help the developer program the behaviour of a swarm. For further details on the specific modelling methods used to design the swarm's behaviour, we refer the reader to Deliverable 4.3 Swarm Complier.

Buzz [78] is a programming language for defining the behaviour of heterogeneous robot swarms. The main characteristic of Buzz is its ability to define the swarm behaviour from both the perspective of a single robot or the overall swarm. From the single robot's perspective, it can define robot-specific instructions and how to handle other neighbouring robot's data. From the swarm's perspective, it allows the selection of a subset of robots to form a robot team and define how information sharing should be handled globally across the swarm. Buzz is a domain-specific language (DSL) which is aimed at filtering out low-level details such as communication and focus the developer's attention to a higher level of abstraction. Buzz has been tested both in simulation (i.e. ARGoS) and with real robots (i.e. marXbot robot).

Some software tools provide graphical user interfaces (GUI) instead of manually writing code to enable developers to create a single robot's behaviour using graphical components (e.g. blocks and arrows). These are often referred as Visual Programming Language (VPL) and have demonstrated their ability to make programming more approachable to inexperienced users such as children [79]. However, to the best of our knowledge, there is no specific visual programming language that is targeted for the design of robot swarm behaviours. A review on recent VPLs for social robots can be found in [80].

Nadzoru [81] is a GUI for designing systems that can be modelled as a discrete event system (DES). Robots can often be considered as a DES, where their control logic is modelled as a finite state machine. Nadzoru uses supervisory control theory (SCT) to model and synthesise the described system according to its functionalities and specifications. SCT is a formal modelling approach that provides certain guarantees to the synthesised control logic, such as it being non-blocking (i.e. the control logic will never be blocked/stuck). Nadzoru is capable of automatically generating source code for the robot from the SCT models and has been demonstrated to produce the control logic for robot swarms [82,83].

In summary, programming languages and user interfaces have been proposed for the design robot swarm behaviours, but research in this area is still scarce and most robot swarms are still developed manually by a few expert developers. This limits robot swarm behaviours to be developed or fine-tuned by potential users who may not necessarily be an expert in designing low-level detains of the swarm.

State of the Art in model-based design and low-code in industry

Over the years, the low-code and model-based design paradigms, together with tools that implement them, have been getting more and popular in the industry to ensure testability and reusability of developed applications. This rise in popularity has also been driven by the promise of making programming custom business logic more accessible to people coming from all different backgrounds, not necessarily only software engineers. In this section, we will provide insights into these two paradigms and shortly provide an introduction into eight representative tools commonly used in business and industrial settings, focusing on their main characteristics.

The low-code paradigm proposes to move coding from textual to the visual domain. Through e.g. various drag-and-drop interfaces used to connect prebuilt components, users can build custom applications through simple interactions, without ever having to write code. Such interfaces allow users with no or little coding experience to build applications and thus democratize access to programming in general. Applications built using such platforms automatically generate the code required to realize the programmed business logic.

Model-based design is a concept extensively used in industries such as automotive or automation control and is closely related to the low-code paradigm. It emphasizes the use of mathematical models to represent complex system logic and behaviour, thus improving the development speed and making programming extensions to such systems easier for all different stakeholders. By using this paradigm, users can easily simulate the system behaviour in a virtual space first and then automatically generate code that can be used to verify and iteratively refine the existing models.

Both concepts are similar and as the end-goal try to abstract the traditional coding away by providing easier-to-use abstractions. We will now look at some off-the-shelf tools that are commonly used in the industry and describe their main features and functionality.

Totally Integrated Automation (TIA) portal (Figure 3**)** is a popular Siemens software for programming and configuring complex real-time capable automation systems based on Programmable Logic Controllers (PLCs) [52]. It borrows from the low-code and model-based design paradigms and enables users to use visual interfaces to program their automation systems, consisting of PLCs, HMIs, IOs and other industrial automation hardware. Through its simulation features, it enables testing of automation workflows before deploying them into production, thus allowing users to verify their models in a simulated environment. This particular tool combines a whole range of different engineering tools into a single platform, giving users a complete overview of the current state of the system. Finally, it provides a consistent library of frequently used functionalities, which are often safety certified, which allows users to visually program additional features into their deployments.

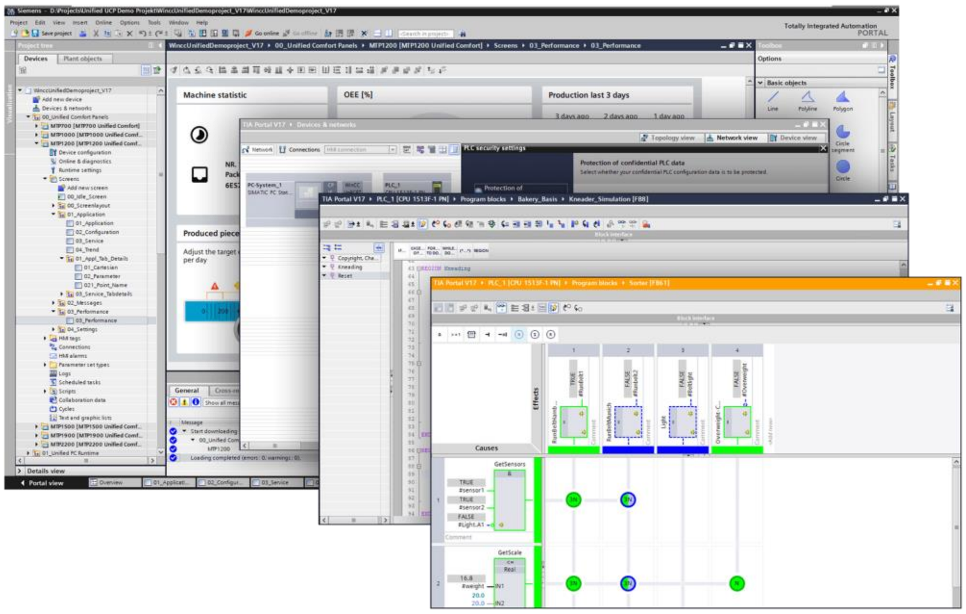


Figure 3: TIA portal GUI [51]

Simulink and Stateflow (Figure 4 **and** Figure 5**)** are two widely used tools provided by Matlab, which enable model-based design of complex systems. Simulink provides a graphical programming environment for modelling, simulating and analysing dynamical systems based on block diagrams and / or state machines [54]. The tool can then automatically generate code in C/C++ or even HDL from designed models, which enables fast deployment of new solutions. These tools also allow for Hardware in the Loop development. Stateflow on the other hand also borrows from the low-code methodology, by providing a graphical tool for designing and simulating control logic through state machines [56]. It forms an extension to the Simulink platform, by extending it with ability to model event-driven systems. Both tools follow the low-code and model-driven paradigms and are actively used in e.g. automotive, aerospace and industrial automation industries.

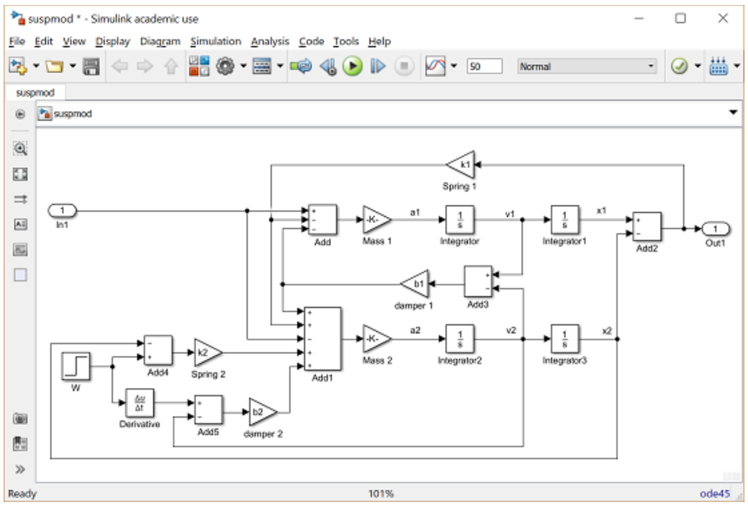


Figure 4: Simulink GUI [53]

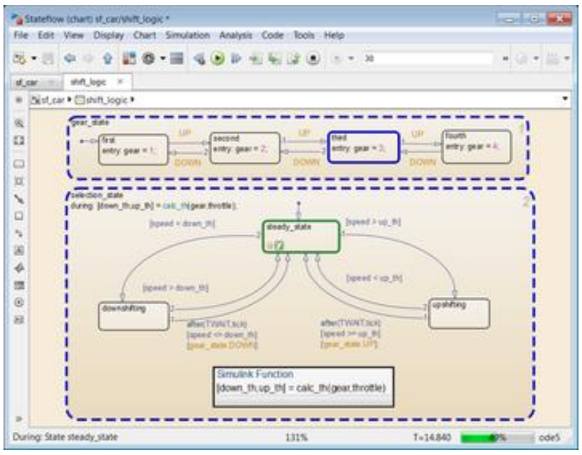


Figure 5: Stateflow GUI [55]

Mendix (Figure 6**)** is a low-code platform provided by Siemens, which targets the development of web and mobile applications [58]. It enables users with any level of experience to develop simple applications that e.g. visualize data or to provide novel means of interaction with existing systems. Through the visual editor based on drag-and-drop interactions for design, and with a suite of tools for deployment and management of deployed applications, it provides a holistic ecosystem that can be used to quickly deliver enterprise grade applications to the end-users. Mendix also provides an extensive library of pre-built templates (e.g. dashboards and widgets) that can be simply reused in new applications. Finally, thanks to the integration with the Industrial Edge Siemens IOT platform applications developed on Mendix can be easily hosted in their target shopfloor environment, directly getting access to all the local data.

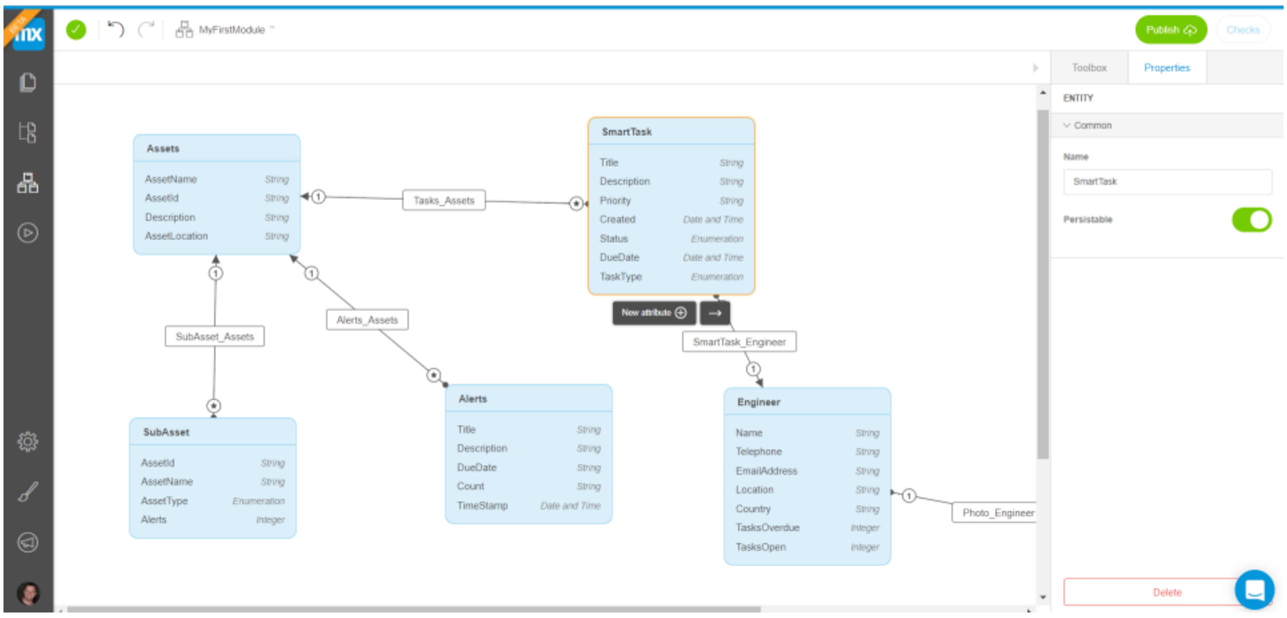


Figure 6: Mendix GUI [57]

LabView (Figure 7**)** is a graphical programming tool provided by National Instruments, which is often used for industrial data collection, instrument control and control automation [60]. It also borrows from low-code and model-based design paradigm, by utilizing visual interactions as the main interface with the user. This tool provides a library of components that can be used for data analysis, signal processing and control design, together with specialized modules for specific use cases such as real-time applications. By wiring together visual components on a screen, complex applications can be generated, that can directly start the data processing without waiting for a developer team to build them first.

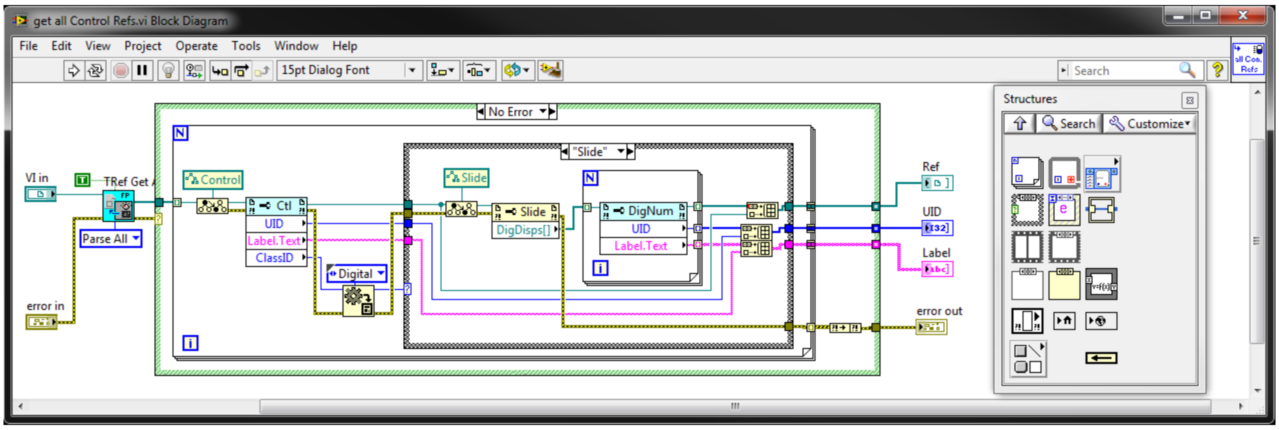


Figure 7: LabVIEW GUI [59]

For more IT-related industries, **Microsoft Power Apps (**Figure 8**)** provides an ecosystem of applications and services that can be used to build custom apps based on the low-code paradigm [62]. Through visual drag and drop interfaces, data from all other Microsoft products can be used to build custom applications, tailored to specific use cases. Users need to specify the required data model first and an application is generated based on the underlying relationships and interactions. The main use case of such tools is business process automation and custom data management applications. Thanks to the integration with the entire Office suite, this tool can be used to easily create new processes and automate existing workflows.

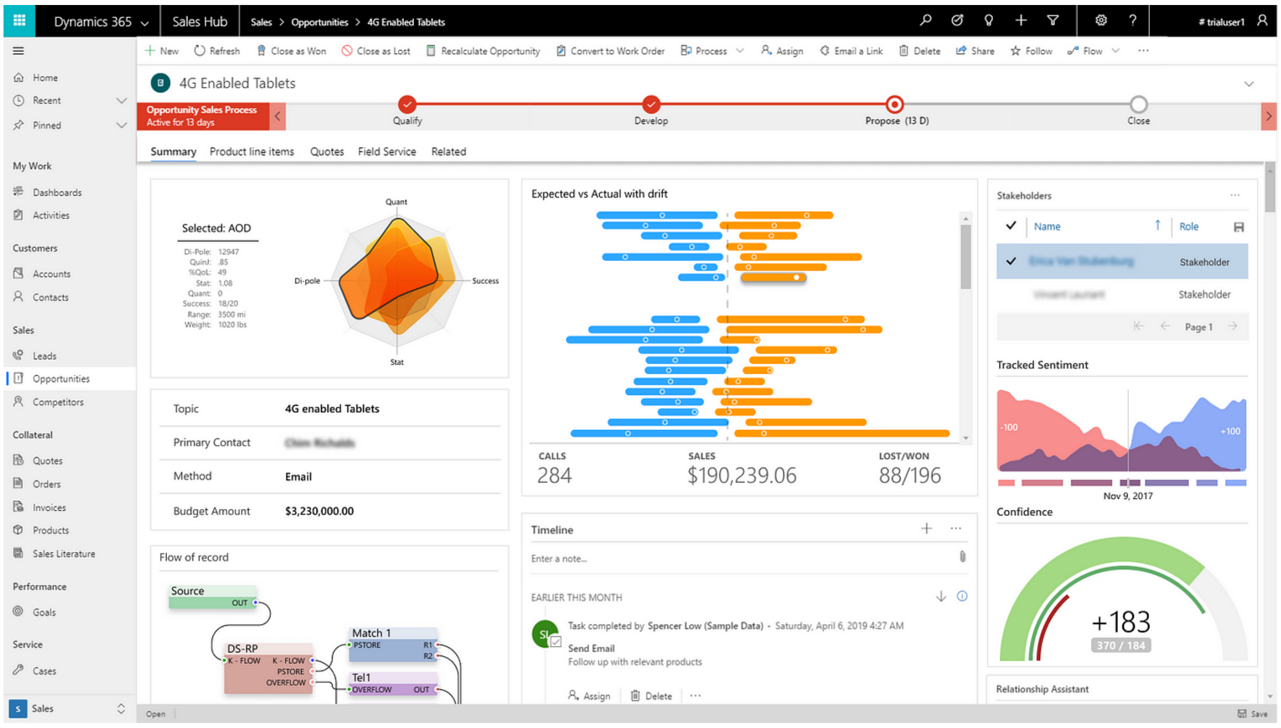


Figure 8: Microsoft Power Apps GUI [61]

SAP Build (Figure 9**)** is another low-code platform for rapid application development and process automation [64]. Because of the ubiquity of SAP enterprise software in the industry, custom applications can utilize the whole ecosystem of data to model new functionality. Moreover, generative AI can be used to generate code, data models and application logic. This tool is mostly targeted at finance, human resources and procurement departments at enterprises. It also allows to incorporate classic programming into the development process, through its pro-code paradigm. Finally, SAP build allows to easily manage the lifecycle of deployed applications using tailored dashboards and metrics.

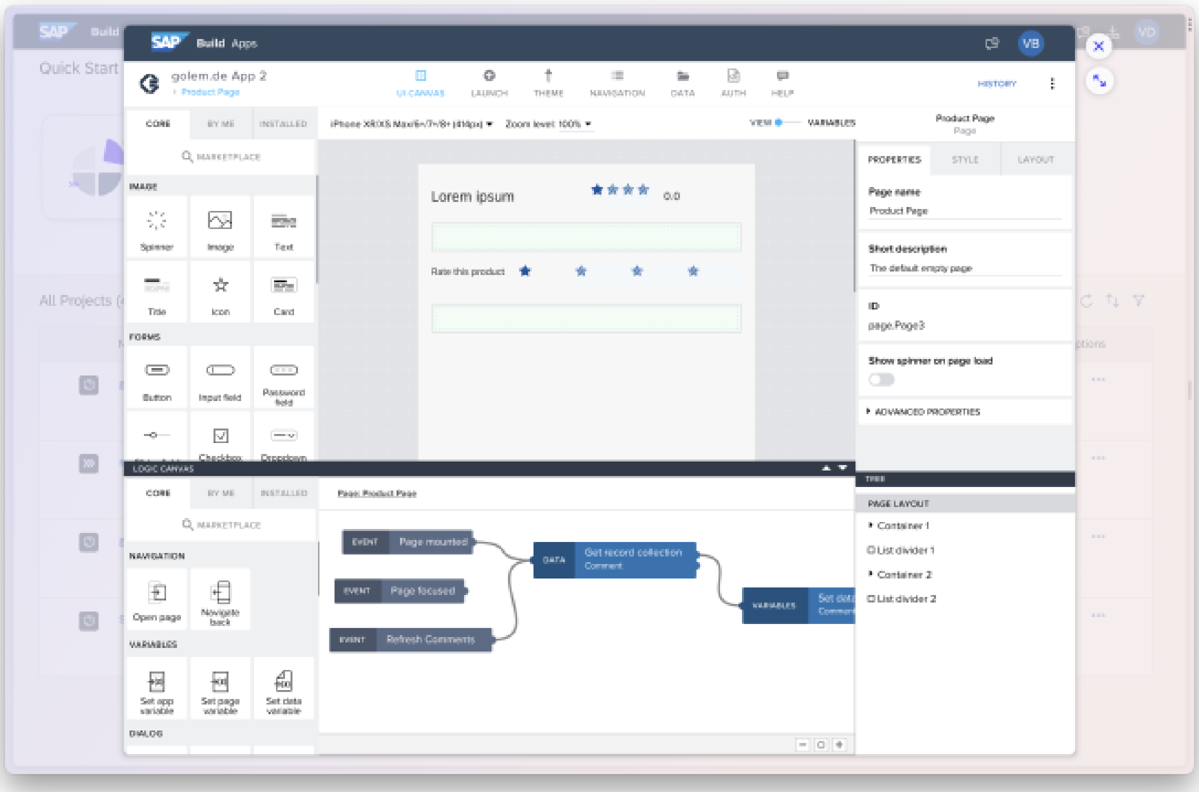


Figure 9: SAP Build GUI [63]

The domain of games engineering also offers low-code solutions for declarative specification of game logic. **Unity Visual Scripting (**Figure 10**)** is one example of such product [66]. It follows the low-code paradigm, by representing the gameplay mechanics and interactions between game objects as visual blocks that can be connected with each other in a drag-and-drop visual interface. Through this tool, non-programmers such as designers and artists can contribute to the gameplay mechanics by simply rearranging the blocks and changing their properties. Since real code is generated based on this visual logic, it can interact with any library or mechanic from the main codebase. This code can then also be extended manually by the developers to tweak its functionality even further.

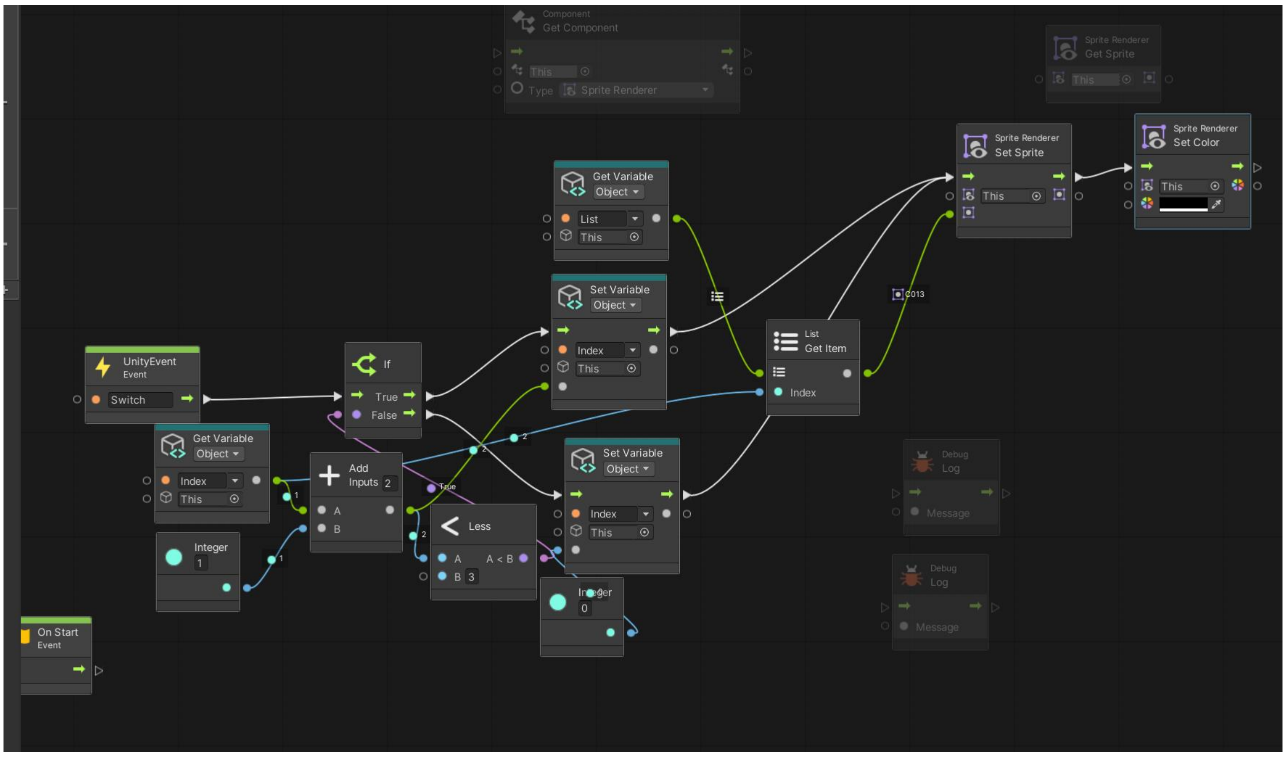
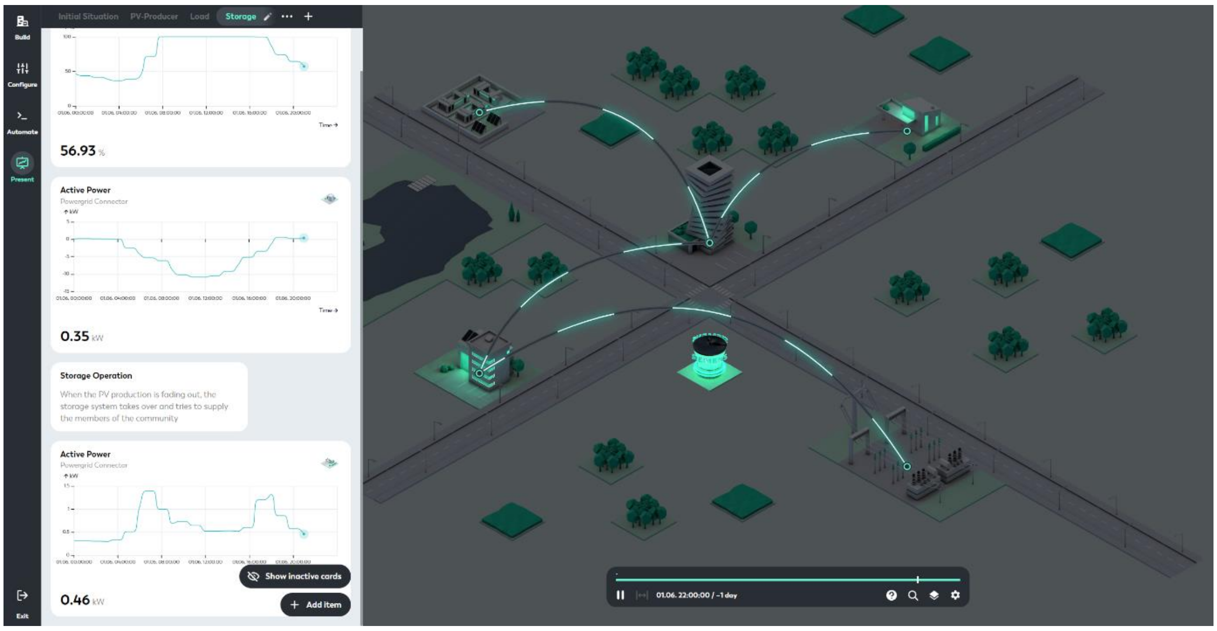


Figure 10: Unity Visual Scripting GUI [65]

BIFROST, is a completely different example. It is a narrative Smart Grid simulation tool for exploring, building, and presenting scenarios about settlements, communities and quarters. It offers simulation orchestration and creative tools to quickly explore variegated scenarios, with the ultimate goals of reducing complexity and presenting technological solutions. It fosters engagement in decision processes and enables integrative discussions across a range of expertise levels.

BIFROST1 is a web-based tool. External modules can interact with the BIFROST platform to emulate complex smart energy infrastructure-related scenarios and provides the tools to design and visualize a compelling Smart Infrastructure story, as it can be seen in Figure 11.

1 bifrost.siemens.com



Storage Operation

When the PV production is fading out, the storage system takes over and tries to supply the members of the community

Figure 11: BIFROST UI with an energy community scenario consisting of multiple different stakeholders (residential buildings, commercial buildings, battery storage), the simulation timeline at the bottom, and some simulation results on the left side.

The Figure shows the UI of BIFROST presenting an exemplary use case about an energy community concept, which compares different strategies and visualizes the resulting KPIs.

BIFROST consists of a core simulation engine to drive dynamic data generation and a 3D web UI for the construction of settlements. Its main components can be seen in Figure 12.

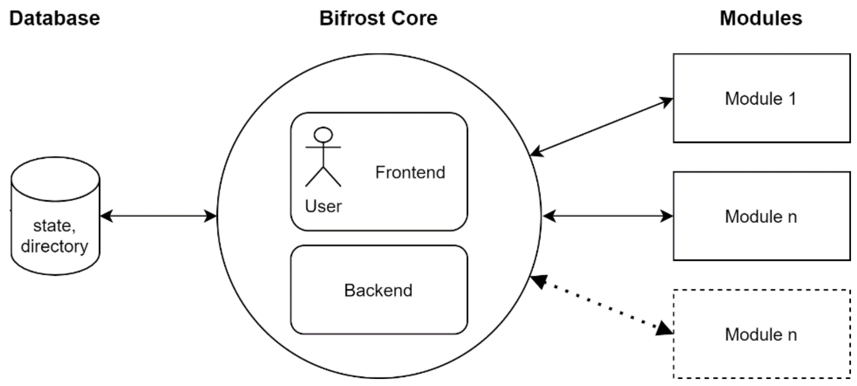
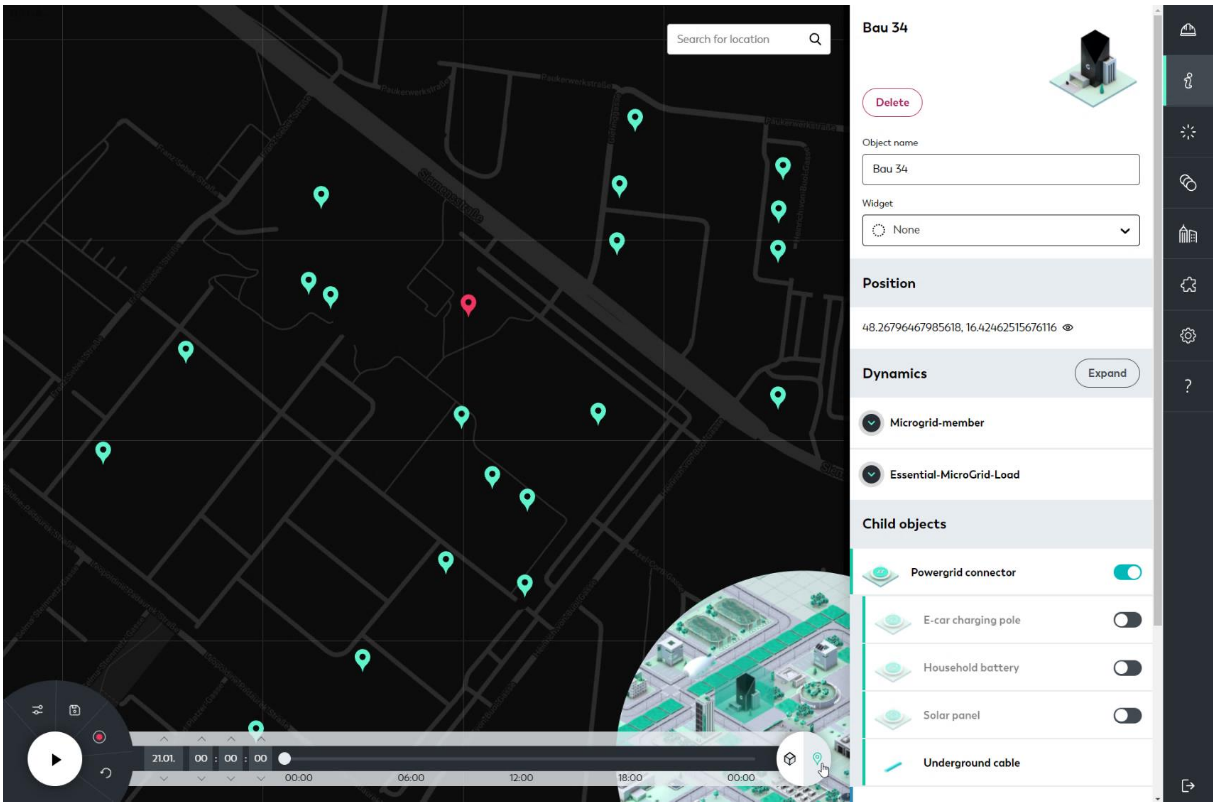


Figure 12: Main BIFROST components: The BIFROST core consists of a frontend for the human interaction and the backend for the orchestration of the internal database. BIFROST modules can interact with the core via a unified REST API interface and can provide any kind of simulation behavior.

The BIFROST backend orchestrates the internal data model, which stores all information about available structures and parameters (BIFROST *directory*) as well as the realized settlements and the corresponding simulation data (BIFROST *state*). The BIFROST core itself does not make any assumption about the content of the directory. Both structures, as well as their parameters can be adjusted to the user's needs, who can access the simulation environment through the BIFROST frontend.

The BIFROST core does not generate any simulation data but provides a unified REST API interface to any software component (BIFROST *module*). Modules can introduce any kind of behavior necessary for the current simulation run (e.g., load flow solver, weather generator, energy community controller, etc.). This flexibility allows BIFROST to tackle various use cases from CO2-efficient energy community concepts to intelligent e-car loading controllers or optimized pricing strategies.

Besides the landscape view shown in Figure 11, BIFROST also includes different layers (electrical grid, thermal grid, communication, etc.) for a clear and structured visualization as well as a geo-mode, which allows to include geospatial relations, see Figure 13:



Bau 34

Object name: Bau 34

Widget: None

Position

48.26796467985618, 16.42462515676116

Dynamics

Microgrid-member

Essential-MicroGrid-Load

Child objects

Powergrid connector

E-car charging pole

Household battery

Solar panel

Underground cable

Figure 13: BIFROST UI representation of geospatial relations in BIFROST

Summing up, low-code and model-based design tools form an important category of tools commonly used in the industry. They cover multiple different application areas and by utilizing them, the access to programming and control of complex business logic can be established, which greatly improves the flexibility of users. This also verifies that such approaches are viable for modelling large and complex systems, thus hinting that they can be useful also for swarm programming purposes.

Motivation for Swarm UI

The Swarm UI's main objective in OpenSwarm is to **facilitate swarm programming**. One of the goals of the OpenSwarm project is to let operators express high-level instructions to the devices. Hence, **"programming" does not imply writing code for individual devices, but rather expressing the desired collective behaviour of the swarm.** The operator should also be able to define sub-systems within the swarm, and express specifications for these sub-systems and the entire swarm, as well as additional control objectives. The Swarm UI should **display comprehensive information about the devices**, allowing the operator to gauge their performance.

Moreover, to assign a task to a swarm, a UI is required that allows the user to describe the task in such detail that it can be processed and executed in a self-organizing manner. The content details of the task should be able to be mapped to the functional capabilities of the swarm members. In addition to the content, the quality, time, and cost aspects under which the task is to be solved should also be able to be described.

While the core objective of the Swarm UI in Open Swarm is on the programming of Swarm Behaviour in its different extends, one of the first results of work in Task 4.2 has been that in general the scope of a Swarm UI should be considered much broader with the respect to the life cycle of a swarm system. Thus, in the requirements and reference architecture sections, the full scope of a UI for the sustainable application of Swarm along the life cycle of the swarm system has been considered. This especially includes the aspect of the operational phase of a swarm system.

Representative User Stories from the project PoCs and the Kilobot Testbed

Due to the sometimes very different use cases of the OpenSwarm system, especially in the application in the various PoCs, their requirements for a possible UI will now be discussed first. To this end, the corresponding user stories will be presented here with regard to possible effects on the UI design.

Energy

The PoC related to Energy is realized in the surrounding of a secondary transformer station in an electrical distribution system with its associated grid (Figure 14).

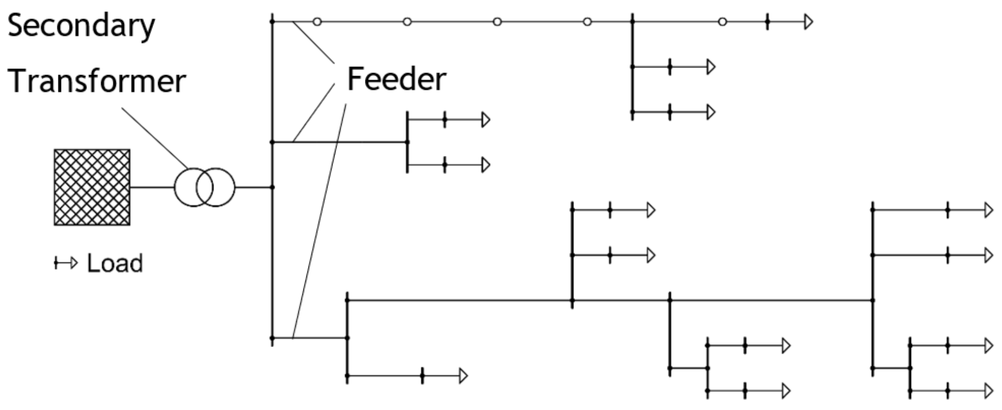


Figure 14: Example of an electrical distribution grid

Various players are active in this section of the distribution network, such as (active or passive) prosumers, sensors, central jointly operated components (such as storage systems or PV plants) and the Distribution System Operator (DSO), see additional components in Figure 15. Individual prosumers may also group themselves together to form one or more energy communities. In this case an energy community operator may be needed.

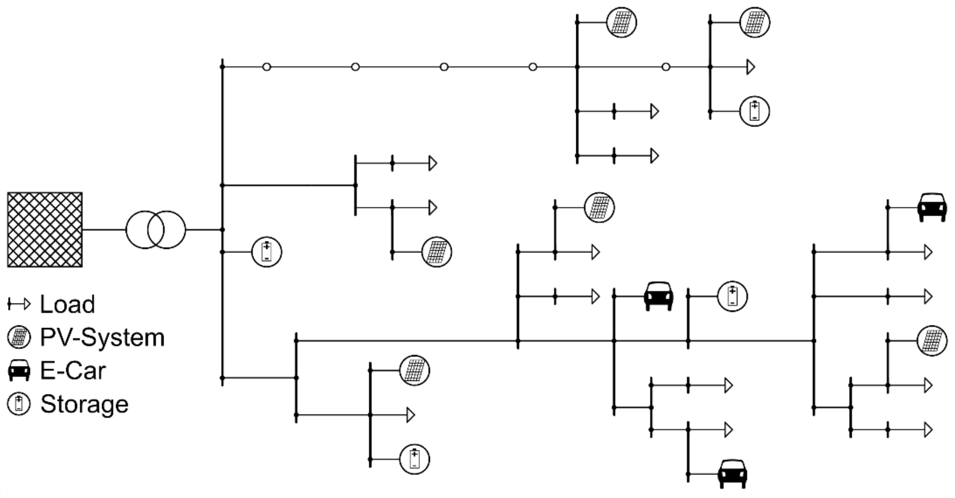


Figure 15: Adding new stakeholder and components to the system

The objective is to establish an interactive network between all the active actors in the grid section, such that they demonstrate cooperative behaviour. In addition, for the optimal use of locally generated energy, it is very important to comply with power grid restrictions such as transformers or transmission line capacity limits or voltage band limits. The idea is to realize this by applying OpenSwarm technologies i.e., via an interactive network, where the interaction between the actors can be achieved without additional configuration effort and the network (swarm) is able to react and adapt dynamically to changes in the environment. Accordingly, new actors will be integrated into the system immediately and effortlessly. The same holds for actors leaving the system, which should be possible at any time with the remaining system reconfiguring itself accordingly.

For a detailed description of the individual components and stakeholders in this system, refer to D1.1, where these are explained in more detail. Some user stories were also formulated there.

Some very specific user stories will now be given here as an example, which will also be implemented as part of the realization of the PoC.

User Story: E-Car-Charging in Energy Communities with Changing Members

In this user story, a grid area is assumed in which generators (e.g. in the form of PV systems) and charging stations for electric vehicles are installed, see Figure 16:

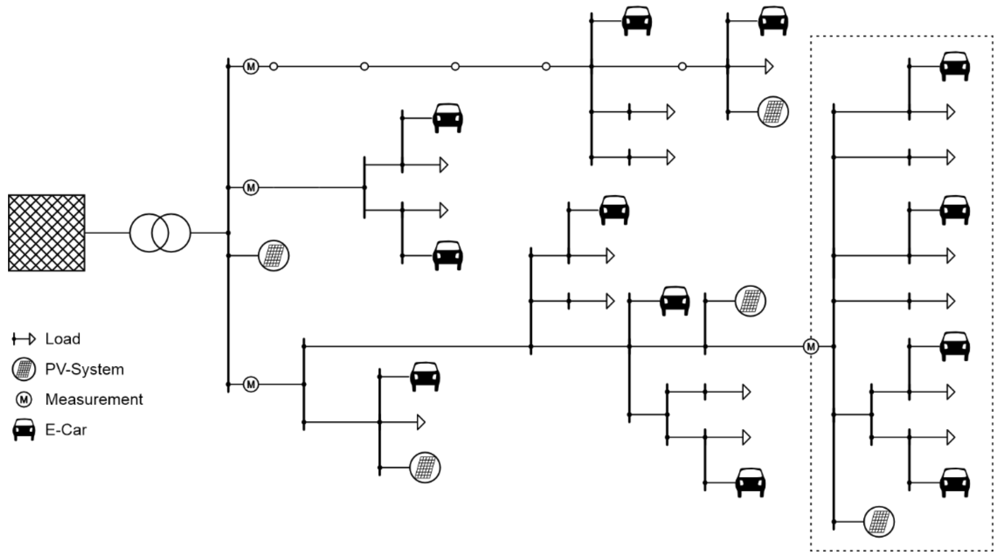


Figure 16: Example for the distribution grid under consideration

The charging stations are operated by individual owners. The energy generated by the PV systems is a low-cost source for the charging stations. So that the locally generated energy can be made available for the charging stations (in terms of billing), the owners of charging stations and PV systems join to form energy communities.

For PV system owners, this has the advantage that they can sell the electricity they generate at a potentially higher price than if they export it via the higher-level grid. They can therefore achieve higher earnings. For the owners of the charging stations, the energy from the local PV systems is nevertheless cheaper than if it had to be purchased from an electricity provider. There is also a greater chance that the energy generated will also be used in charging processes. For example, a prosumer who owns a PV system and a charging station can only ever charge their own car with the electricity they generate. However, if they also offer their generated electricity to other charging stations, other cars can also be charged with it. If the participants in an energy community also connect their systems using data technology in such a way that, for example, the charging processes are set to make optimum use of the PV power currently being generated, additional profits can be generated for the PV system owners and savings for the owners of charging stations.

We therefore assume in this user story that energy communities are established for this purpose in the distribution grid area under consideration. There are two in total: the red and blue energy community, see Figure 17:

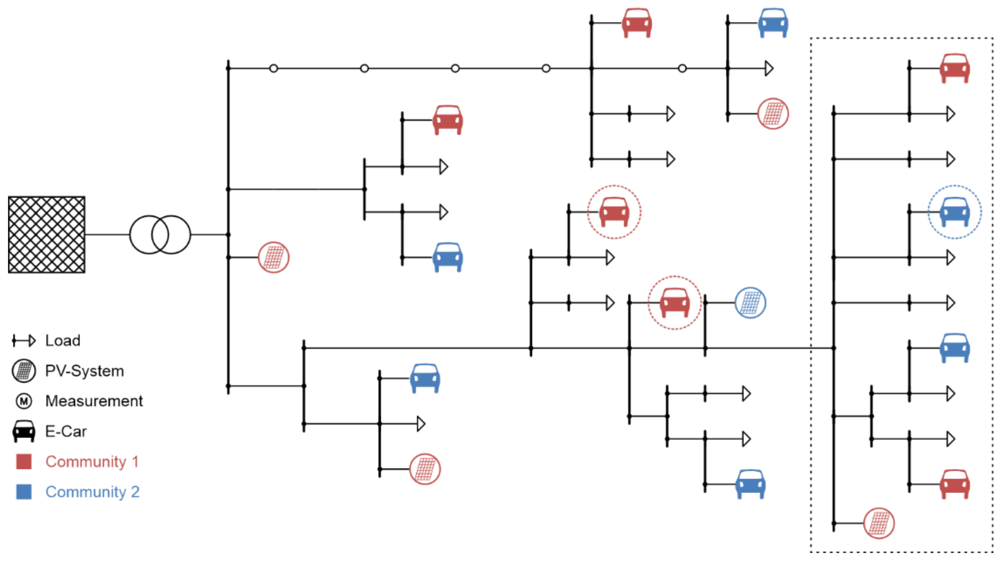


Figure 17: Assignment of PV systems and charging stations to energy communities

The blue energy community has a PV system (perhaps a community system) and six charging stations. The energy community can now control the charging stations so that when a lot of energy is generated, the charging power is increased until the charging power corresponds to the power generated. If the charging power is higher than the power generated, the charging power is reduced again.

This is the case, for example, if a vehicle is already charging and uses the entire power generated for this and then another vehicle in the community begins to charge. The energy community must now have regulations on how the second vehicle is to be charged. To stay with the example, the generated power can now be divided between the two vehicles. In other words, the charging power of the first vehicle is reduced, and the released power is made available to the second vehicle. If, for example, the first vehicle is now fully charged and completes its charging process, the second vehicle can now use the entire power generated for charging.

Other rules are also conceivable, such as “First Come, First Serve” or there are charging stations with higher priority and those with lower priority.

From the UI perspective, it should be possible in this context to define these rules for the community and communicate them to the individual participants. Communication between the participants must also be made possible: I must therefore be able to define which participants belong to which community.

If we now consider the individual components, such as the PV system and the charging stations, as elements of a swarm, the considerations of the operation of swarms presented in the previous explanations can also be found in this user story, even if the elements of the swarm are not “mobile” in the sense of spatial movement.

However, mobility exists in the form that participants can join an energy community individually and switch between them. For example, the participants circled in Figure 17 may decide to join the other energy community and leave the current one. This is shown in Figure 18:

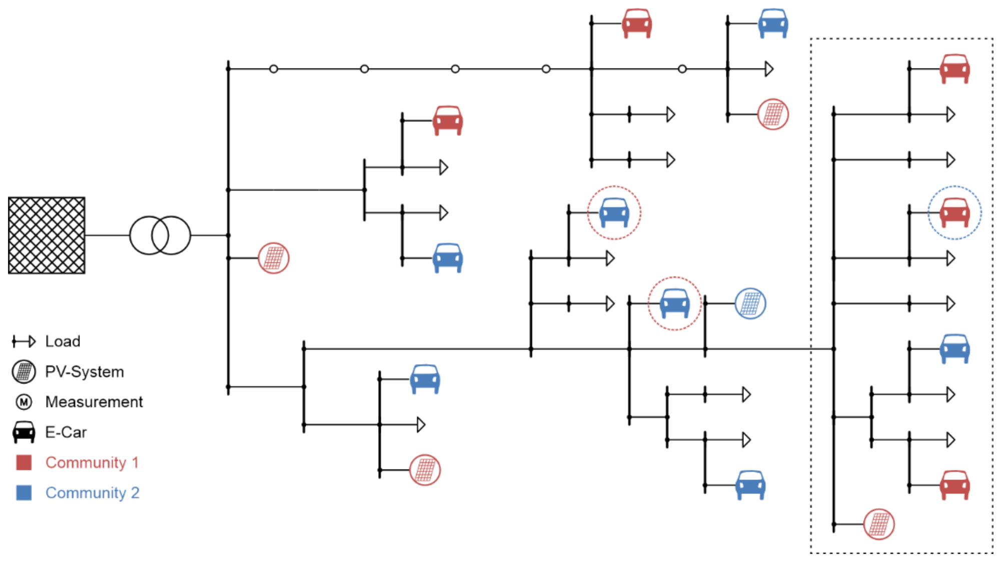


Figure 18: Transfer of participants from one energy community to another

On the one hand, this change must ensure that the new members are correctly accepted into the new community and correctly separated from the original community. From a UI perspective, this should be visible to the community operator. The current rules must also be adapted correctly, which may also affect other participants in the community.

User Story: Limit Charging Power due to Power Grid Restrictions

If we stay in the same set-up and now consider the case where, for example, energy must be drawn from the distribution grid at times when there is no PV generation. In this user story, so much power is installed for charging stations in the distribution grid under consideration that if many of them were to charge at high power, lines could become overloaded, see Figure 19:

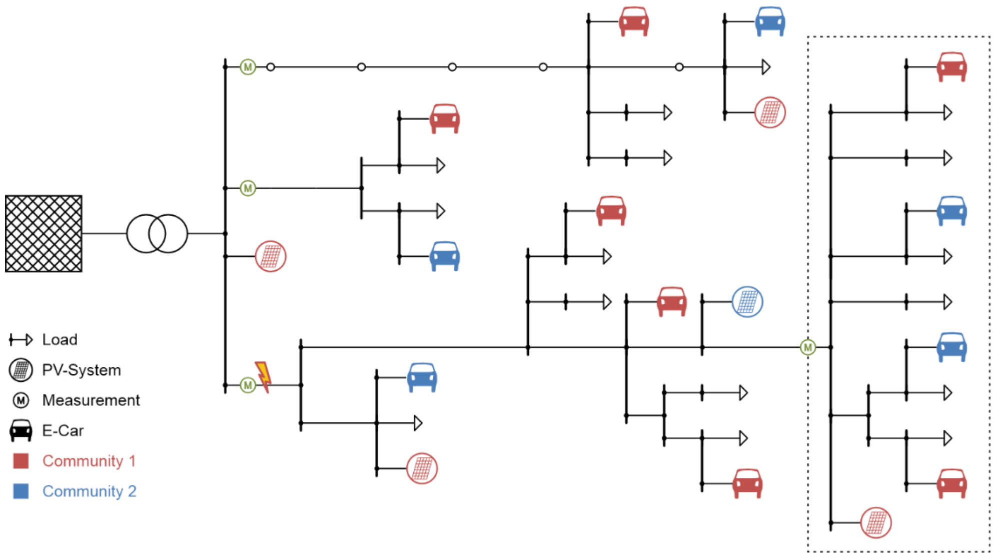
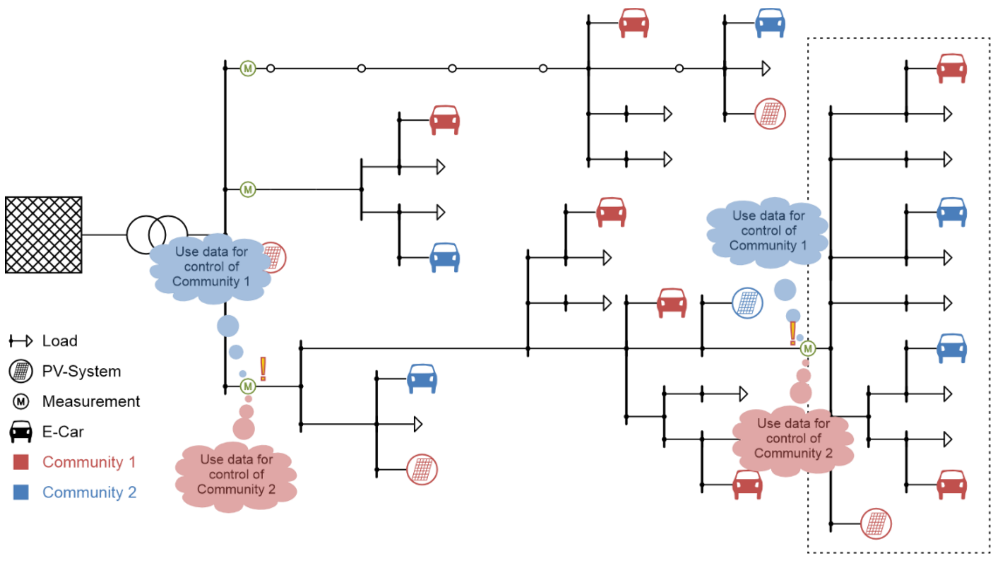


Figure 19: Power failure due to high charging power

This may lead to power outages, which is why the DSO would probably not allow such a high power to be installed.

However, the energy community can now undertake to comply with specified power values and to distribute the available power among its participants so that no overload situations can occur. To this end, however, the DSO, for example, would have to be able to communicate measurement data and limit values to the energy communities so that they can react to them, as outlined in Figure 20.



Use data for control of Community 1
Use data for control of Community 2
Use data for control of Community 1
Use data for control of Community 2
Load
PV-System
Measurement
E-Car
Community 1
Community 2

Figure 20: Use of measurement data to avoid overload situations

Here, too, there must be suitable interfaces for this data exchange. On the other hand, from a UI perspective, a way must be created to correctly assign the relevant information and rules as the operator of the community and communicate them to its own participants.

These two user stories from the Energy PoC are intended to show that the swarm under consideration can also take on very different forms than in other, more robotics-based systems. Nevertheless, the requirements described must be able to be mapped in a corresponding UI.

Environment, Health and Safety (EHS)

As part of the EHS use case, a PoC for an active, individual, and context-dependent noise warning system in an industrial environment is to be created. Factory workers should be warned, by an application running on a digital wearable (e.g., watch, smartphone, glasses), when he moves into a noise-prone context (i.e., place, time). Additionally, a general overview of the data should be visualizable by means of a noise map.

The system is therefore divided into three sub-aspects: i) the context-based collection of data (noise measurement and the associated metadata), ii) the generation and provision of the noise map, iii) the individually configurable app for the workers.

In the following, the first aspect, the collection of data in an industrial environment, is considered as a swarm programming use case since the noise measurements are to be carried out using an existing fleet of Automated Guided Vehicles (AGV) equipped with the necessary measuring technology.

This aspect of the EHS PoC places the following requirements, formulated as user stories, on the Swarm UI:

1. As **EHS manager**,
    a. **I would like** to get an overview of the current EHS state and valid regulations in the area of my responsibility (e.g., a noise map of the factory floor as data source) **in order to** assess dangerous situations.
    b. **I would like** to enforce policies (e.g., spatial resolution of square meters, temporal resolution of 10 minutes, two measurements per position, have the measurements within the next 10 days) **in order to** comply with existing regulations.
    c. **I would like** that all positions that can be accessed by people are monitored and analysed **in order to** be able to use the map holistically in all situations.
    d. **I would like** the system to perform automatic and context-based (e.g., human proximity) detection of anomalies and potential hazards **in order to** timely inform factory workers and prevent dangerous situations.
    e. **I would like** that the system works on anonymized data (e.g., human-related information) focusing only on ensuring the safety of factory workers **in order to** comply with data protection policies.
2. As **Factory owner**,
    a. **I would like** that the measurements are carried out with existing AGV (already equipped with a microphone) fleet **in order to** avoid additional effort.
    b. **I would like** that, when possible, measurements are carried out while other tasks (e.g., the transport jobs) are being processed **in order to** achieve good efficiency.
    c. **I would like** to have guarantees that the EHS policies are fulfilled, whether this is done by reserving EHS dedicated swarm nodes or doing task overlapping, **in order to** ensure factory workers' safety and comply with EHS regulations.
3. As **application developer**,

   a. **I would like** that the measurements are made with the Noise Capture (NC) application **in order to** get the measured value with the entire context (timestamp, location/zone, etc.).
   b. **I would like** a statement on the quality of the data provided by the machines and the network, **in order to** be able to interpret it correctly and derive meaningful measures.
4. As a **factory worker**,
   a. **I would like** to get context and situation related hints and warnings automatically, like a warning, when I'm entering a dangerous area in the factory **in order to** avoid dangerous situations.
   b. **I would like** to configure my personal levels of when I'm receiving warnings, **in order to** get personalized hints and warnings.
   c. **I would like** to use devices (e.g., smartphone, tablet) to interact with the OpenSwarm network, **in order to** have direct alerts of imminent danger as well as to provide feedback.
5. As an **EHS engineer**,
   a. **I would like** the network and the Autonomous Mobile Robots (AMRs)/AGVs to take information of the network into account, **in order to** adapt navigation planning of the AMRs/AGVs based on the human coverage in the proximity and the overall factory site to increase the overall safety level on-site.
   b. **I would like** the map monitoring (part of the map processor) to be able to request/trigger measurement jobs **in order to** fill in white spots (missing measurements at certain positions/time windows) in the noise map.
   a. **I would like** to have an overview (e.g., current location, last measurements, node status) of all the swarm nodes executing EHS tasks **in order to** get a sense of how the high-level policies are been implemented.
6. As a **Swarm operator**,
   a. **I would like** to easily add any kind of new systems, and network participants, **in order to** make the system flexible and scalable. This must be efficient and automized to avoid configuration failures.

b. **I would like** to have independent attachable sensor tags, which can be configured (e.g., their position) based on mobile units with known position configuration, **in order to** augment the context information on which the swarm operates

c. **I would like** that a swarm participant (e.g., an AGV) can interact with non-swarm elements to get access to all areas that people can access, **in order to** get a complete measurement coverage.

To investigate the swarm programming requirements, the Kilobot Testbed will be used as a representative of the driverless transport systems.

Kilobot Swarm Research Testbed

This section first introduces the Kilobot platform [75] and its existing UI based on the **Augmented Reality for Kilobots (ARK)** system [76]. It subsequently provides the requirements, in the form of user stories, for use with the Kilobot Swarm Research Testbed.

The Kilobot, shown in Figure 21, has become a prominent swarm robotics platform in the research community over the past decade because of its low-cost (allowing labs to afford large numbers), compact size (fitting well in typical lab spaces), robustness (reducing faults), and ease of use. However, these design advantages come at the cost of important features for swarm robotics research, such as precise environmental sensing and remote access to the robot's status.

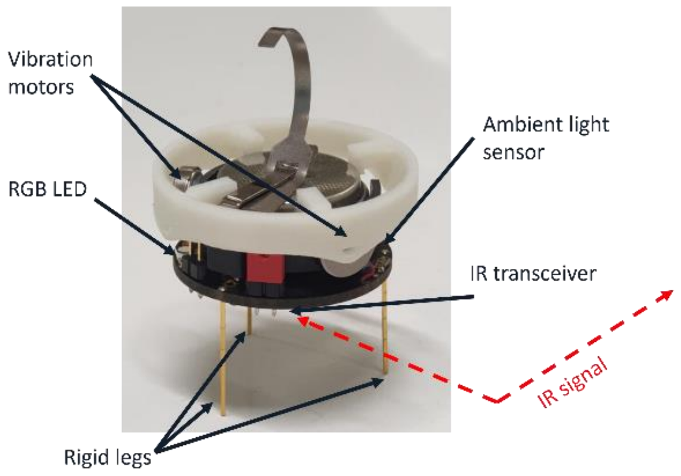


Figure 21 Kilobot platform

The Augmented Reality for Kilobots (ARK) system was developed by researchers to enhance the capabilities of the Kilobot platform. ARK allows Kilobots to sense and modify virtual environments, expanding the scope of experimental activities they can perform. As illustrated in Figure 22, the ARK system consists of software running on a GPU-enabled computer (the Base Control Station), which is connected to overhead cameras that track the robots' position, heading, and state (indicated by their LED colour). Additionally, IR emitters are used to transmit information to the robots.

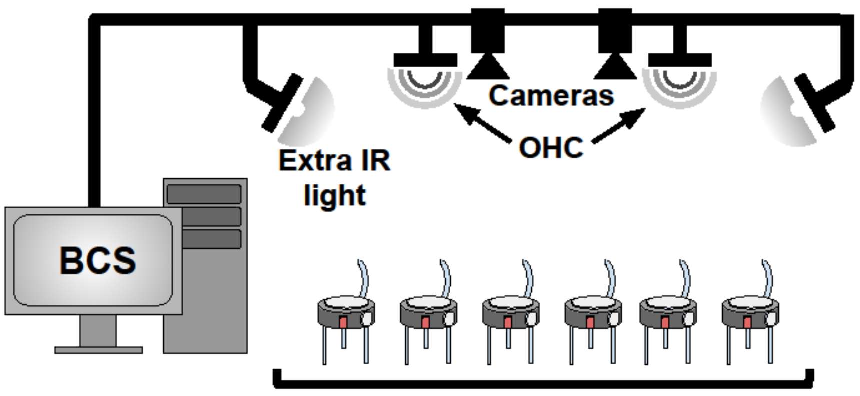
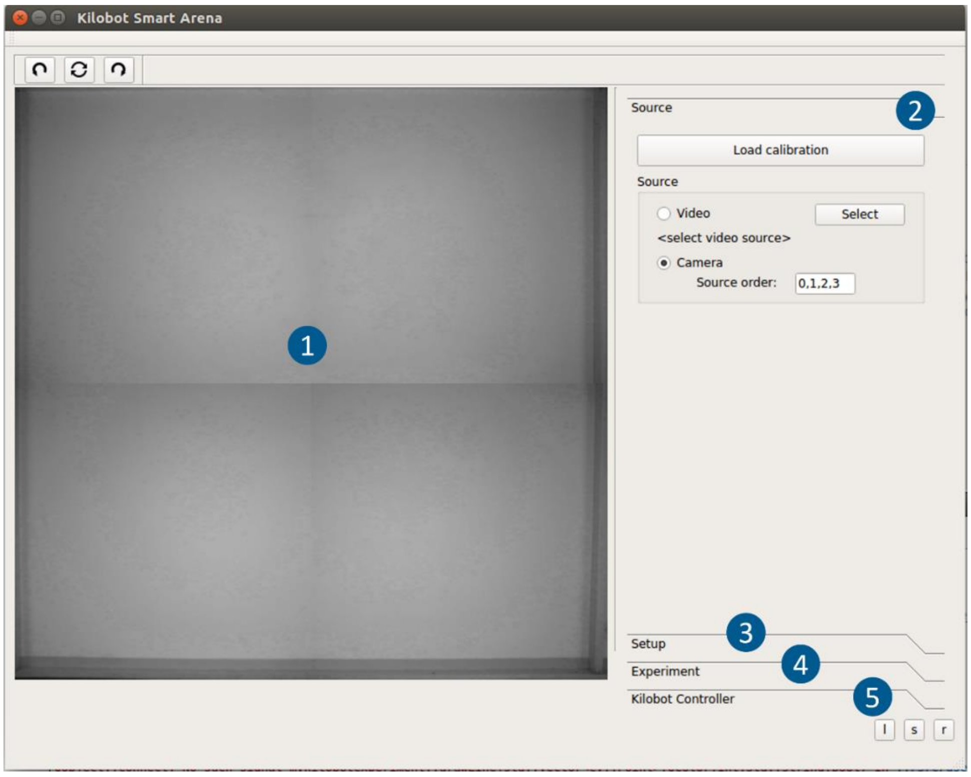


Figure 22 Architecture of the Augmented Reality for Kilobots (ARK) system [77]

ARK's User Interface, as depicted in Figure 23, features an image view on the left **(1)** and a set of functionality tabs on the right. The *source* tab **(2)** allows users to specify the image source, either from a live camera stream or pre-recorded video. The *setup* tab **(3)** enables users to set tracking parameters and perform pre-experiment routines. The *experiment* tab **(4)** provides UI elements related to experiments, allowing users to set parameters and control experiments in real-time. Finally, the *controller* tab (5) allows users to start or stop the robots and upload a controller (.hex file) to the robots.

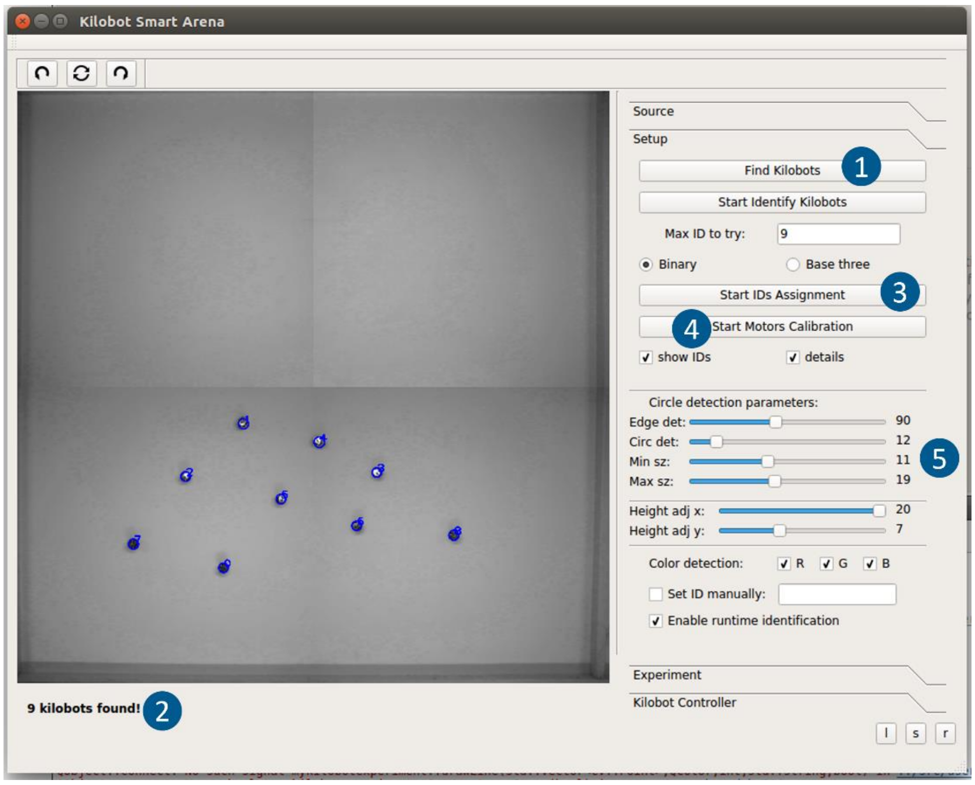
Figure 24 displays the elements of the *setup* tab. The *"Find Kilobots"* button **(1)** enables tracking of the Kilobots currently in the arena, with detected robots indicated in the image feed and their number reported at the bottom of the UI **(2)**. The *"Start IDs Assignment"* button **(3)** automatically assigns unique IDs to the available Kilobots. The *"Start Motors Calibration"* button **(4)** calibrates the robots' motors automatically, allowing them to perform basic motions such as clockwise/counterclockwise turning and forward movement. The *sliders* in **(5)** allow users to set the position and colour tracking parameters.



Source

( ) Video
<select video source>
(●) Camera
Source order: 0,1,2,3

Figure 23 Main elements of ARK'S UI



Max ID to try: 9

(●) Binary ( ) Base three

☑ show IDs ☑ details

Circle detection parameters:

Edge det: 90
Circ det: 12
Min sz: 11
Max sz: 19

Height adj x: 20
Height adj y: 7

Color detection: ☑ R ☑ G ☑ B

☐ Set ID manually:

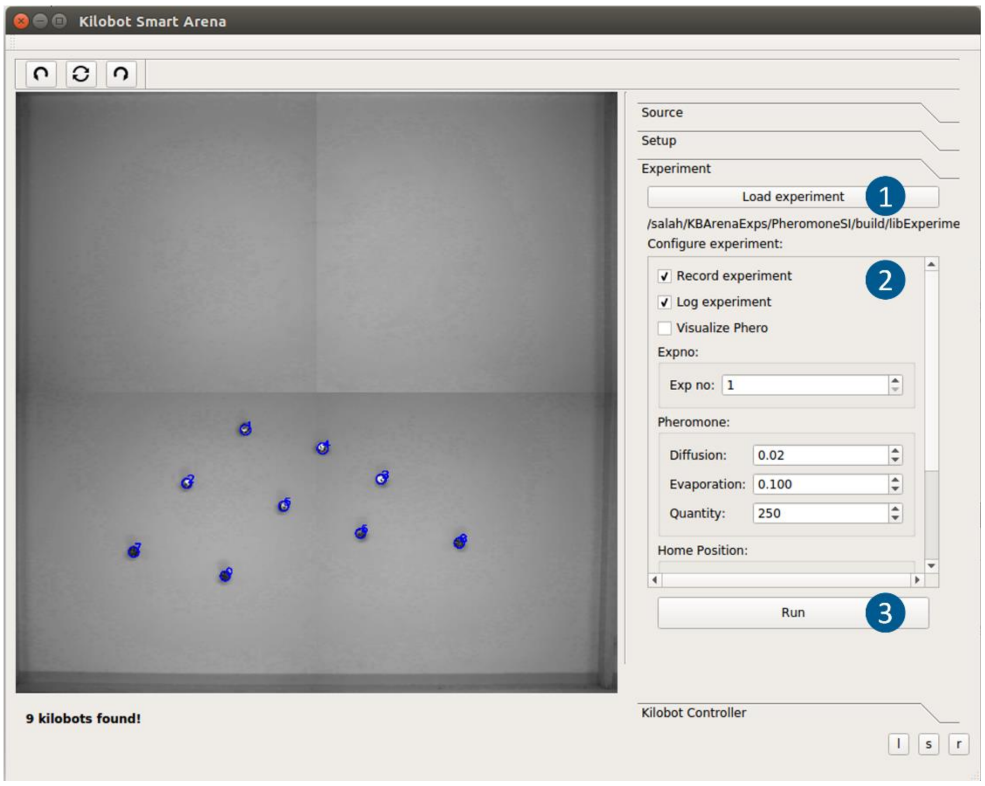
☑ Enable runtime identification

9 kilobots found!

Figure 24 Elements of the *setup* tab.

Figure 25 depicts the elements of the experiment tab. The "Load Experiment" button (1) allows the user to load a library (.o) file containing the coded experimental scenario. This file creates and dynamically updates the virtual environments and informs the robots about these environments. Additionally, the experiment file handles data logging and image saving for post-processing. It also defines the UI elements for configuring the experiment (2).

Figure 26 shows the elements of the controller tab. The “Connect OHC” button (1) allows the user to connect the IR emitter. The “Select File” button (2) lets the user choose the controller (.hex) file, while the “Upload” button (3) sends this file to the robots via the IR emitter. The “Run/Reset/Sleep” radio buttons (4) enable the user to start, stop, and put the robots into sleep mode. Additionally, the “Voltage radio” button allows users to check the robots’ charge levels, which are indicated by the colour of their LEDs.



/salah/KBArenaExps/PheromoneSI/build/libExperime

Configure experiment:

- ✓ Record experiment
- ✓ Log experiment
- ☐ Visualize Phero

Expno:

Exp no: 1

Pheromone:

Diffusion: 0.02

Evaporation: 0.100

Quantity: 250

Home Position:

9 kilobots found!

Figure 25 Elements of the *experiment* tab

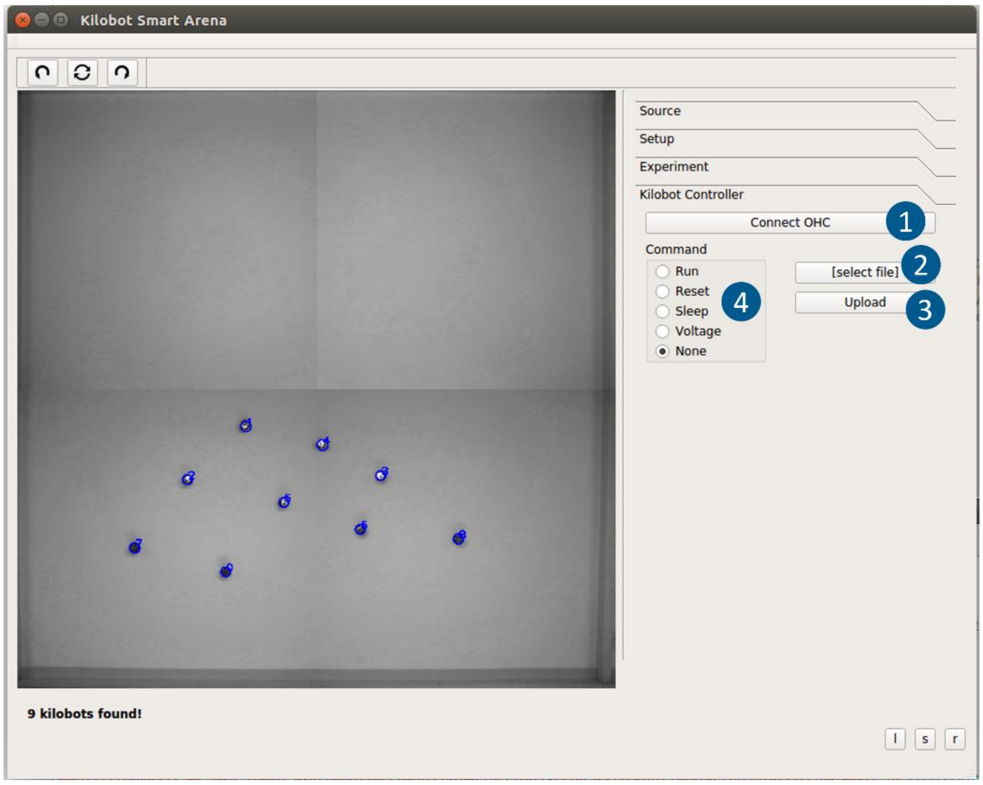


Figure 26 Elements of the *controller* tab

The current interface already addresses the following requirements, formulated as user stories:

As a **Swarm operator**,

- **I would like** to assign unique IDs to individual devices.
- **I would like** to calibrate the robots' motion, so they can consistently perform basic movements (turn left, turn right, and move forward).
- **I would like** to configure the experimental scenario, including features present in the virtual environment and parameters regarding their perception by robots.
- **I would like** to upload compiled swarm programs to the devices of the swarm.
- **I would like** to start, pause, resume, and stop the execution of the program on all devices.
- **I would like** to monitor the swarm. In particular, I would like to be provided with a bird eye's view of the world, augmented with the virtual features.

- **I would like** to track the positions of, and communicate with, individual devices during experimental trials.

As a **Swarm developer**,

- **I would like** to define experimental scenarios, including features present in the virtual environment and how the robots are perceiving them.

In addition, the **new Swarm UI** addresses the following requirements, formulated as user stories.

As a **Swarm operator**,

- **I would like** to formulate the capabilities and specifications of the devices.
- **I would like** to automatically obtain a compiled program reflecting the aforementioned capabilities and specifications.

As a **Swarm developer**,

- **I would like** to define the call-back functions for controllable and uncontrollable events that are associated with the capabilities of the robot. During this step, the function declarations are provided by the system.

Swarm UI Problem Analysis

Swarm environments are complex, collaborative, and distributed systems that involve a multitude of heterogeneous devices, applications, behaviours, and capabilities. This section provides a characterization of these environments, focusing on their users, infrastructure, applications, engineering, and operation. From this characterization we extract the associated requirements on which we further elaborate into a set of Key Architectural Requirements for the Swarm UI.

Users

Users in swarm environments can be categorized according to their specific roles and requirements as follows:

- Domain-Specific Designers: These are domain experts with limited programming capabilities. They can design, start, and stop tasks and applications. Their role is crucial in defining the tasks and applications that the swarm will execute, based on their domain-specific knowledge.
- Domain-Specific Operators: These users observe the swarm and application behaviour for anomalies and manage the swarm. They can start and stop tasks and swarm applications but do not design them. Their role is to ensure the smooth operation of the swarm and to intervene when necessary.
- Domain-Specific Developers: These users have general development skills and extend and develop templates and functionalities (e.g., for low code) to be used by domain experts. They bridge the gap between the domain-specific designers and the underlying swarm infrastructure.
- Swarm Developers: These users have a deep understanding of complex collaborative distributed systems. They develop swarm libraries and optimization mechanisms. Their role is to ensure the efficient operation of the swarm at a system level.

Derived Requirements:

- The Swarm UI should provide different development environments. For example, we can have visual ways to model the program and application, but also pure

coding; including low code development intended for domain experts, code development intended for extending the domain-specific features, and code development intended for extending the UI.
- The Swarm UI should provide a component and software catalogues (e.g., swarm behaviours, swarm libraries, application templates, device models) to support low-code development environments.
- The Swarm UI should provide means for operating, monitoring, and controlling the full lifecycle management of the swarm.

Assumptions:

- Instead of addressing each category separately it is sufficient to ensure that the UI provide the overall set of required functionalities.

Infrastructure

The infrastructure in swarm environments accounts for highly heterogeneous devices with different, software stack (e.g., RTOS, Network Protocols), hardware (e.g., CPU, memory, and GPU), and I/O (e.g., sensing and actuation) capabilities. Moreover, these devices can be highly mobile and as such resource availability and task execution capabilities would be highly entangled with the current location of the devices. Consequently, devices resources should be considered as highly volatile, adding an additional layer of complexity to the swarm environment.

Derived requirements:

- Modelling and description of devices: The Swarm UI should include the capability to model and describe the different kind of devices, in order to abstract the underlying heterogeneity and provide a valid system topology feedback to the Swarm UI user, allowing them to control where to deploy what.
- Node discovery and announcement: For usability we should support device / swarm node discovery, self-announcement, self-configuration as far as possible.
- We should provide an editor for specific configuration of a swarm node.
- Abstractions needed for swarm node visualization and control.

Assumptions:

- The Swarm UI will not consider unknown device classes.

Programming

Applications in swarm environments are highly heterogeneous, from the large collection of different and volatile infrastructure elements to different goals and requirements (both in terms of performance and resources needed). This calls for a flexible programming environment with different programming scopes and many models and ways to implement and program application and swarm behaviour. Moreover, the complexity and heterogeneity associated with Swarm Programming could leverage well established concepts of Continuous Integration Continuous Deployment (CI/CD) and infrastructure automation.

Derived requirements:

Programming requirements focus on the development, testing, and handling of abstractions in the swarm UI. These include:

- Development: The Swarm UI should enable the definition of different programming scopes that would allow to define the overall task, objectives and behaviours of an entire swarm, sub-groups of the swarm (e.g., defined by the device-type, or some defined semantic role of swarm nodes), specific swarm nodes, any combination of the above.
    - Data models are required to define and communicate the addressed scope to connected components.
    - Visualization of deployment and interfaces for scoping configuration needed.
    - Programming scope(s) need to be made transparent to the users of the swarm UI.
- Testing: The UI should provide mechanisms for testing solutions before deployment. This ensures that the developed solutions work as expected and do not cause any unexpected behaviour in the swarm.
- Handling Abstractions: This includes defining goals, restrictions, and Key Performance Indicators (KPIs), modelling infrastructure and engineering aspects, and mapping requirements and models (e.g., code decorators).
- Infrastructure Automation: Provide mechanisms for code artifact management models ranging from manual deployments on each single device and components to fully automated infrastructure.

Assumptions:

- To keep the development process agnostic of the infrastructure, the resulting software artifacts should be deployable across the swarm (i.e., in different targets). Compiling software artifacts for the right device target into adequate deployable units is external to the UI.
- There are technologies in place (e.g., data-centric overlays) that would allow for efficient communication with the swarm devices under specific programming scopes.

Engineering

Engineering in swarm environments involves complex design processes to ensure the efficient operation of the swarm. These processes need to consider the complexity of the tasks, the heterogeneity of the infrastructure, and the dynamic nature of the resources it will use.

Derived requirements:

Engineering requirements focus on simplifying the engineering interfaces (e.g., Intent-based), providing optimization mechanisms that convert high-level requirements into concrete deployment and operational decisions, providing feedback of engineering decisions, and collecting results. These include:

- Simplified Engineering Interfaces: The UI should provide simplified engineering interfaces, such as Intent-based interfaces. These interfaces allow users to specify their intentions, and the system translates these intentions into concrete actions.
- Optimization Mechanisms: The UI should have control and utilize optimization mechanisms that convert high-level requirements (e.g., energy usage) into concrete deployment and operational decisions. This allows the system to automatically optimize the deployment and operation of the swarm based on the specified requirements (self-optimization). This includes both the initial planning and the self-adaptation to changing conditions.
- Engineering Abstractions: The Swarm UI should abstract the end-user from engineering tasks such as configuring devices and communications and Resource and Task management. These engineering abstractions should be

used in conjunction with optimization mechanisms and engineering interfaces allowing engineering decisions to be tailored to the application and swarm goals and requirements.

- Multi-tenancy: The Swarm UI should be able to consider multiple applications from different tenants to be working simultaneously (e.g., EHS applications should run in parallel with traditional logistics applications for AGVs).
- Feedback of Engineering Decisions: The UI should provide feedback of engineering decisions. This allows users to understand how their requirements are translated into concrete actions by the system, allowing for further tailoring of the overall objectives, as well as debugging engineering processes.
- Collecting Results: The UI should provide the means for collecting the data resulting from the execution of the Swarm Application. This allows users to analyse the performance of the swarm and make necessary adjustments.

Assumptions:

- The UI does not need to provide very low-level management or configuration interfaces. Instead, this is something that can be done by selecting the appropriate goals, requirements, and swarm libraries.

Operation

The operation of swarm environments involves managing the overall life cycle of complex collaborative distributed systems, ensuring the efficient distribution of tasks, and handling the dynamic nature of the resources and providing transparency of the swarm behaviour to the different swarm system users, e.g. also related to situational awareness.

Derived requirements:

Operation requirements focus on the visualization, management, monitoring, and event notification in the swarm UI. These include:

- Visualization: The UI should provide visualization with different levels of abstractions (e.g., swarm, swarm partition, device). This allows users to understand the state of the swarm briefly.
- Management: The UI should provide mechanisms for managing the swarm lifecycle (e.g., deploy, start, stop resume, update tasks or applications). These

processes should be done in most cases over-the-air and provide the adequate mechanisms for detecting and handling inconsistencies.

- Monitoring: The UI should provide monitoring of the different swarm resources e.g., communications, computation, storage. This allows users to keep track of the performance of the swarm and detect any potential issues (e.g., detection of swarm misbehaviour)
- Event Notification: The UI should provide event notifications (e.g., alarms). This allows users to be notified of important events in the swarm, such as errors or performance issues.
- Error Handling: The UI should provide libraries and mechanisms for handling and potentially tracing errors at the different abstraction levels (i.e., from infrastructure to application). The long-term goal is to have a robust, self-healing swarm.

Assumptions:

- It is possible to model and assess the swarm behaviour.

General Considerations

There are a common set of general requirements that should be taken into consideration when designing any system:

- Role-based access control ensures that only authorized users can access certain features of both the swarm UI and the swarm itself (e.g., specific devices and/or operations).
- Charging and accounting mechanisms to keep track of resource usage and the associated payment for these resources.
- Integration, providing interfacing with other systems (e.g., Manufacturing Execution Systems (MES), end-user devices) and with other Swarm UIs (e.g., for joint swarm effort in emergency response scenarios involving different first responder organizations). Integration mechanisms can be further developed in alignment with federation concepts.

In addition to these general functional requirements there are also some key non-functional requirements that must be taken into consideration:

- Usability: The UI should be user-friendly, intuitive, and responsive to ensure a smooth user experience.
- Scalability: The UI should be designed to handle a large number of devices and tasks without performance degradation.
- Security: The UI should ensure the security of data and operations, including secure communication, data encryption, and protection against unauthorized access.
- Interoperability: The UI should be designed to work seamlessly with different types of devices and systems.
- Reliability: The UI should be reliable and provide consistent performance under different conditions.
- Maintainability: The UI should be easy to maintain and update, with clear documentation and support for debugging and troubleshooting.
- Extensibility: The UI should be designed to allow for easy addition of new features and functionalities.

Overall, these requirements will ensure that the Swarm UI is robust, functional, and user-friendly, and can effectively support the management and operation of the swarm system.

Swarm UI Reference Architecture

Designing the UI for designing, deploying, and managing the swarm across its lifecycle requires taking into consideration various the various requirements and user categories presented in the previous section. The base assumption is to make it easy for end-users to interact with and monitor the swarm as well as applications deployed on it. This section provides an insight into the key architectural requirements and the overall architecture and system context, as well as an overview of the main functional components and the rationale behind the design decisions.

Key Architectural Requirements

Before designing the reference architecture for the Swarm UI, we have derived a set of key architectural requirements based on the PoC user story analysis, swarm UI problem analysis. These architectural significant requirements KAR-1 to KAR-8 and a brief perspective on how to approach them are given in the following:

KAR-1: Provide a set of ***common functionalities*** *(e.g., Role-based Access Control, Accounting, Charging) orthogonal to swarm programming itself, but that support the overall utilization of the Swarm UI. These functionalities should, in general, be coupled with other system context components to enable the correct operation of the Swarm UI.*

→ For the implementation of these functionalities one can rely on existing generic open-source solutions. While the implementation of the functionalities can remain generic, there is a need for developing adequate interfaces towards the different system components (e.g., leverage resource monitoring information for accounting and charging, associate access permissions with different device types/models).

KAR-2: The UI should be user-friendly, intuitive, and responsive to ensure a smooth user experience. To that end, it should provide ***different programming environments*** *targeting the different user categories. For example,* ***designing swarm behaviours, developing***

***applications, low-code templates, libraries, and device models**. It should also support CI/CD operations and testing functionalities.*

→ The architecture should consider the different environments, providing editors tailored to each development context, as well as enabling the information exchange between the different context (e.g., developed low-code templates should be available for swarm designers). The architectural primitives for infrastructure and operation management should be made available for CI/CD related processes, the use of a Swarm Simulator could release some of the infrastructural burden while at the same time ease the process of influencing the swarm environment and context.

*KAR-3: The Swarm UI shall support **visual composition of swarm behaviour** based on **multiple modelling approaches** (e.g., Supervisory Control Theory, Workflow Modelling) on heterogeneous devices used by the Swarm Engineer and Operator. This should consider functional primitives available on the Swarm Nodes (e.g., movement), event sources and sink in their context/environment (e.g., events from business systems) and their interactions.*

→ The architecture provides core functions like canvas, nodes, edges, triggers, configurations, which then can be applied to multiple modelling approaches and the interaction and key functionalities of the UI can be shared between the different components. All visualizing components (e.g. for modelling, swarm engineering, monitoring) approaches will be preferably developed based on web-technologies, to support browser-based usage on heterogeneous end-user devices.

*KAR-4: The Swarm UI should provide simplified **engineering** primitives that abstract the overall complexity of swarm environments. These primitives can be extended through swarm libraries that implement relevant engineering algorithms. These algorithms should account for different engineering tasks, including resource management, task management, and target management. A key aspect of this process is the matching of application and behaviour definition into available infrastructure towards the achievement of a well-defined set of goals.*

→ The architecture should include a common set of engineering functionalities that shall be extendable via swarm libraries. Swarm behaviour design interfaces should provide explicit means for defining core engineering requirements (e.g., device type, required resources) and targets (e.g., energy optimization, completion time, information quality).

KAR-5: Provide interfaces for ***operation*** *of the swarm (e.g., manage application and device lifecycles, monitor swarm nodes and swarm applications). This will allow an easy extension of the Swarm UI to a fully-fledged swarm management system in an iterative manner.*

→ The architecture should provide interfaces for operating the swarm, including features for visualizing the swarm, managing devices, and handling telemetry data. Moreover, it must consider the full lifecycle scope of the swarm.

KAR-6: The Swarm UI shall support infrastructure management of heterogeneous devices, multiple ways of modelling swarm applications, and hereby multiple swarm application compilers.

→ The architecture will provide functionalities that, coupled with proper device models, will allow for device management including registry and lifecycle management, discovery as well as well-defined interfaces for orchestrating swarm applications onto the adequate devices. These functionalities will work in close interaction with the Swarm UI visual components for gathering device telemetry and managing swarm device topologies.

KAR-7: The Swarm UI architecture shall support the extension with other separately deployed UI Components to provide advanced features for swarm applications especially in the PoCs. In general, this will consider the interfacing with components of the overall system context (SBI), Interfacing with external components (NBI), interfacing with other Swarm UIs (EWI)

→ The architecture design will follow the guidelines of separation of concerns and clear split between data model, modeling, visualizing, monitoring, compiler, and deployment components. The respective components will expose interfaces as APIs. Components which are part of the UI itself will expose this as library calls, other components will use APIs via network interfaces like gRPC, MQTT, or REST. A pub/sub-based approach (like MQTT or constrained Coaty with the DDA) will be preferred as this follows the asynchronous event-based architecture approach, defined for multiple PoCs. The exchange format of the UI with other components will follow common text-based standard formats (e.g., json, yaml).

KAR-8: The Swarm UI should provide Swarm Libraries, Low-code Templates and Models Catalogues. Swarm UI development environments should be able to consume as well as publish artifacts from/to the relevant catalogues. The catalogues should provide common primitives such as version control and can be closely related to the Swarm UI CI/CD functionalities.

→ The architecture should implement these software catalogues and made available to the different development environments/editors as well as to other shared components for swarm engineering and operation tasks.

High Level Architecture View

For the overall functionalities, the components are divided into two broad groups, the "Visual editor" responsible for application as well as swarm behaviour management and the swarm configuration, and the "Functional libraries" comprising a common of support components needed for the overall operation of the UI. The diagram below provides a first insight into the Swarm UI Components, the external components it interacts with (e.g., other OpenSwarm architectural elements) and the Swarm Infrastructure.

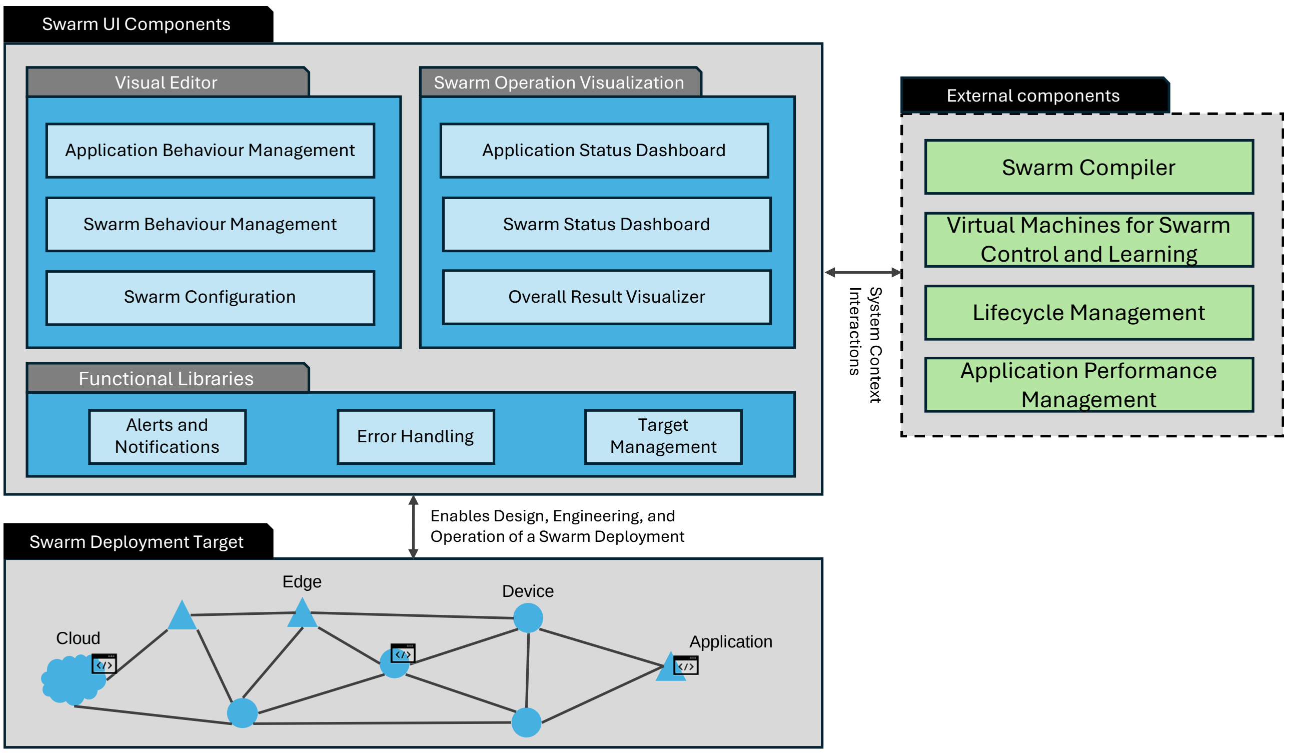

Swarm UI Components
Visual Editor
Application Behaviour Management
Swarm Behaviour Management
Swarm Configuration
Swarm Operation Visualization
Application Status Dashboard
Swarm Status Dashboard
Overall Result Visualizer
Functional Libraries
Alerts and Notifications
Error Handling
Target Management
External components
Swarm Compiler
Virtual Machines for Swarm Control and Learning
Lifecycle Management
Application Performance Management
System Context Interactions
Enables Design, Engineering, and Operation of a Swarm Deployment
Swarm Deployment Target
Edge
Device
Cloud
Application

Figure 27 High level architecture view of Swarm UI

In the following section all components part of the high-level architecture as visualized in Figure 27 are described in more detail:

1. Visual editor: A visual editor, which can be based on a low code or drag and drop in design, or based on a domain specific modelling language, enables the design, deployment and management of the overall Swarm Application/Behaviour. As mentioned in key requirements KAR 3, 4 and 5, it is important to provide capabilities that abstract complex engineering capabilities regarding the configuration and definition of the behaviour yet provide a powerful visual way to do it for the end user, The visual editor components like application and swarm behaviour management enable the same.

    a. Application Behaviour Management: Since multiple applications might be deployed on top of a swarm, the developer and/or designer will need tools and capabilities to define the behaviour of the application. These behaviours can be workflow related, dependence and relation with other applications and swarm devices, or human interaction dependent. Additionally, rules and policies as well as data collection and event flows also need to be defined for the overall application scenario. The Application Behaviour Management components cover the functionalities required to enable it.

    b. Swarm Behaviour Management: To manage the overall swarm behaviour, the end user will need tools and capabilities to design a swarm with its functional (for example, specify the devices part of the swarm, workflow that the swarm needs to support, overall size of the swarm, set of capabilities the swarm can perform) and non-functional (availability, fault tolerance, self-healing) requirements. Additionally, capabilities to first test and simulate and then deploy and monitor the real time behaviour of swarm will be needed. Swarm Behaviour management components provide the functionalities to fulfil these requirements.

    c. Swarm Configuration: The applications running on distributed swarm nodes, and the swarm and the swarm nodes itself need to fulfil certain behaviours and non-functional requirements. To fulfil the expected behaviour, user needs to specify certain configurations for the Swarm. Using Swarm Configuration creators, user can define device groups and their configurations, overall network configuration (latency, connectivity requirements, behaviour in case of loss of connectivity), environment configuration (interaction with humans, other elements, interaction with other swarm devices) and Swarm KPIs. For example, a NFR of high availability is dependent on the devices being charged and available within the swarm. A user can use swarm configurator to define the minimum battery level beyond which the device should move to charging station. Additionally, user can define the minimum number of devices a swarm should have. Defining the number of devices is part of swarm configuration whereas how the swarm should behave in such a condition is defined as part of the Swarm Behaviour Management. Essentially these components work very closely together to provide a strong Swarm Configuration and Definition capability.

2. Swarm Operations Visualisation: These components of the Swarm UI are responsible for all kind of visualizations during swarm operations like dashboards. The telemetry data received from the swarm via the Swarm UI external components is used for these operation visualizations. Users shall be able to create their own graphs and charts based on configurations. In the following the three top category of Swarm Operations Visualisations are described in more detail:

    a. Application Status Dashboard: The application status dashboards provide insights into the various application states, points within the application lifecycle, devices that are currently performing the application and the overall application status. It also visualizes various applications part of a common workflow, their event integrations and data flows.

    b. Swarm Status Dashboard: The swarm status dashboard provides insights into the swarm health, status of the swarm nodes and corresponding devices part of the swarm, their individual status, device group status, and other telemetry data as configured by the user.

    c. Overall Result Visualizer: Overall result visualizer depicts the results of the one or more deployed applications together with the information from the swarm. Users can look at whether the overall task is completed, important KPIs associated with the task and the executed workflow along the lifecycle.

3. Functional Libraries: Certain functionalities are common to and can be reused for multiple components in the visual editor and might be needed for the overall user experience. It makes sense to extract these functionalities in a set of libraries which are accessible to all the services but are maintained in one place, thus providing increased flexibility and maintainability.

    a. Alerts and notifications: The component allows users to configure certain conditions for generating alerts. When those conditions are triggered, the user gets alerts and notifications either in the UI or via an email or other services. User can configure various frequencies for the notifications, and severities for alerts. This needs to be handle both in the Swarm UI but also in the external components and specific interfaces need to be made available.

    b. Error handling: During the application behaviour management or swarm behaviour management, whenever an error is encountered, this component can be used to define the way it has to handled. Which alerts need to be triggered, which users need to be notified, what should be the fallback in case of error, what is the redundancy mechanism, how should the fault tolerance kick in are some of the topics that are covered by this component. The ultimate goal is that this component would be an enabler for self-healing swarm applications. This needs to be handle both in the Swarm UI but also in the external components and specific interfaces need to be made available.

    c. Target Management: Once the swarm and application behaviour are configured by the user in the visual editor, the task to map the application and swarm to the best suited devices and topologies is still remaining. This component is responsible for mapping the configurations and applications to the optimal swarm cluster and devices based on user defined goals. Target management might not always be accurate and uses the best match scenario for the deployment. User can also define manual intervention points and notifications in case the best match scenario is below certain defined threshold.

4. External Components: The Swarm UI components interact with external components to perform the functions requested by the user. Swarm compiler is responsible for creating deployable units for different targets based on the inputs from the swarm visual editor. Virtual Machines might be used to run this deployable units on the swarm devices. Lifecycle management functions and capabilities on devices are interfaced with the UI to provide analytics and control over the entire lifecycle of swarm and application. They interact directly with the swarm infrastructure. Application performance management information is used to have continuous monitoring information of the swarm at different levels and scopes.

Swarm UI components as well as the external components together enable running the Swarm and deployment of applications on top of it. In the following sections, let's take a deeper look at the overall System Context as well as the Functional Decomposition of the architecture.

System Context

The Swarm Programming UI interacts with a number of components to enable the functionalities designed and needed for the overall swarm applications. System context diagram provides an overview of the external components, their dependencies and interfaces with the Swarm UI and overall functioning of the system. Figure 28 shows an overview of this system context of the swarm UI.

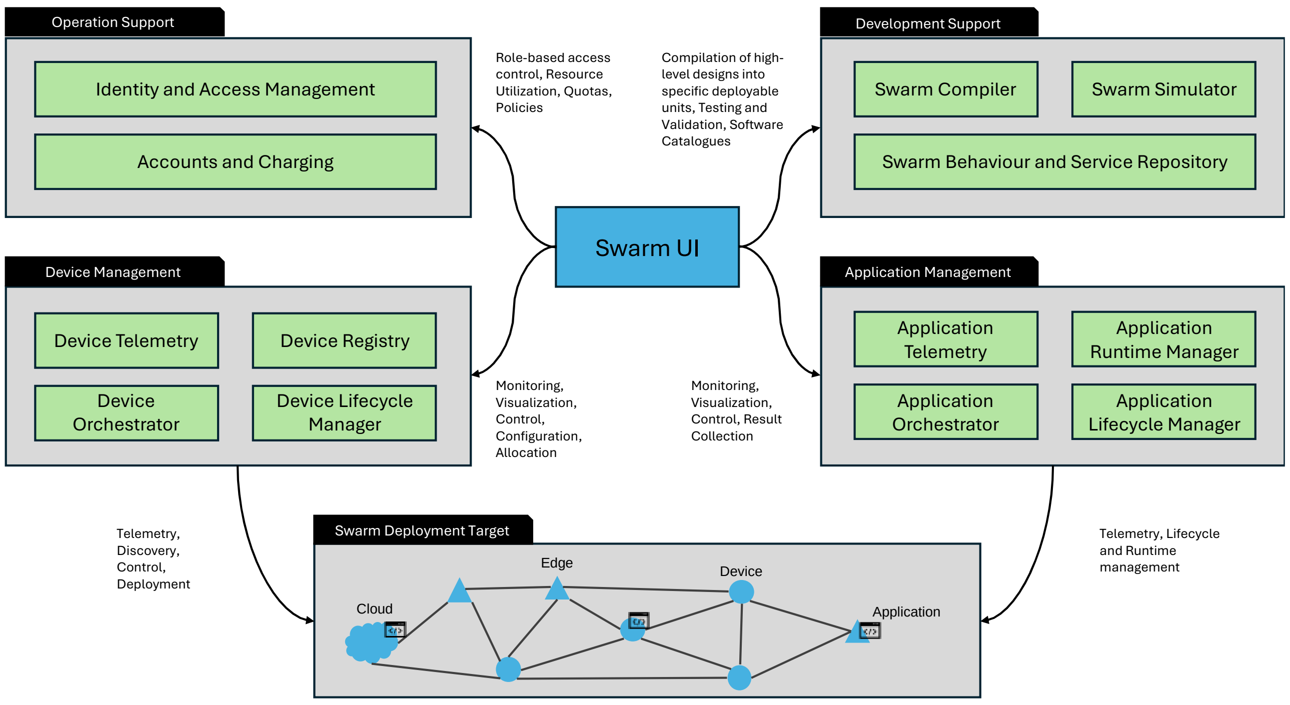

Operation Support
Identity and Access Management
Accounts and Charging
Role-based access control, Resource Utilization, Quotas, Policies
Compilation of high-level designs into specific deployable units, Testing and Validation, Software Catalogues
Development Support
Swarm Compiler
Swarm Simulator
Swarm Behaviour and Service Repository
Swarm UI
Device Management
Device Telemetry
Device Registry
Device Orchestrator
Device Lifecycle Manager
Monitoring, Visualization, Control, Configuration, Allocation
Monitoring, Visualization, Control, Result Collection
Application Management
Application Telemetry
Application Runtime Manager
Application Orchestrator
Application Lifecycle Manager
Telemetry, Discovery, Control, Deployment
Swarm Deployment Target
Edge
Device
Cloud
Application
Telemetry, Lifecycle and Runtime management

Figure 28 System context of Swarm UI

The system context contains the components interacting with the Swarm UI:

1. Device Management: Device management is a category of components responsible for the management of devices and the nodes respectively the nodes of the swarm. They concern different actions like deployment and orchestration, lifecycle management and data collection for the overall swarm.

    a. Device Telemetry Collector: Various statuses and data from devices are collected using this component. The data collected by this component feeds directly into the Alert and Notification as well as Error Handling components of Swarm UI to provide a set of information to the end user regarding the overall Swarm status.

    b. Device Registry: Device registry manages a register of devices and their configuration. Any new device added to the swarm, or removed from the swarm needs to be updated here. The Swarm UI uses this list to plan the deployment of swarm and the application.

    c. Deployment Manager and Orchestrator: Once the devices are chosen for the swarm deployment based on the previous components, Deployment Manager and Orchestrator is used for deploying the swarm configurations and the applications on the swarm. It collects the application details from the swarm compiler and deploys it on swarm devices.

d. Device Lifecycle Management: Management of devices, and their relevant data like for instance battery status, location, task status etc are part of this component. It also considers the state date of swarm devices like software versions etc. and takes care of update and software management. The component is responsible to provide insights that can be used for task assignment as well as maintenance of the device. Together with device registry, this component is used to map the requirements from the Swarm design from UI to the optimal set of swarm devices.

2. Application Management: A set of applications or tasks are assigned to the Swarm devices to perform. The Application Management components take care of that.

   a. Application Telemetry Collector: Like the device telemetry collector component, this component collects information regarding application, it's current state, completion, dependent applications and workflows and provides this information to the Swarm UI. Swarm UI combines application as well as device telemetry to enable the alerts and notifications.

   b. Application Lifecycle Manager: Various stages of the application and their status is tracked and responded to using the Application Lifecycle Manager. It is responsible for the completion of application in an optimal manner. For example, if an application is stopped or interrupted, the lifecycle manager informs the user, as well as uses fall back mechanisms prescribed in the configuration to ensure application completion. Swarm UI based application configuration provides input to this component to define the overall application behaviour.

   c. Application Orchestrator: Application Orchestrator is responsible for deploying the prepackaged application and related workflows on to the respective devices on the swarm. It works closely with the Device Management to identify optimal deployment strategy for the application.

    d. Application Runtime Manager: Since the applications can be of different classes with different runtime requirements like hardware configuration, dependencies, processor classes to name a few, it is important to manage the runtime of the application. This component together with Application Orchestrator, manages the runtime of the application observing for defined configurations like memory requirement, communication requirements etc and optimizes the application runtime. In case of challenges, or issues, it also sends information via the application telemetry regarding the same.

3. Development Support:

    a. Swarm Application Compiler: Swarm Application Compiler is one of the most important core components needed for the Swarm UI to function. This component takes as input the swarm design and configuration and compile that to create a deployable unit for swarm devices. Since there can be multiple classes of devices part of Swarm, the compiler creates multiple deployment units for different target devices. It can get the device details from the device registry part of the device management components. It can store and fetch artefacts from the Swarm Behaviour and Service Repository. For every Swarm deployment, existing components are fetched, and new deployable units are compiled via the compiler. Once the compilation is done and the artefacts are ready for deployment, they are sent to the Deployment and Orchestrator component of the Device Management to be deployed to the swarm devices.

    b. Swarm Simulator: Once the swarm and application design has started, it needs to be tested before deployment on the actual swarm. Swarm simulator provides that capability to the Swarm UI. It mimics the swarm and environment behaviour and provides insights into how the swarm would behave when the design is deployed. This service takes as input the swarm design, and application configuration from the UI, and provides a simulation result as output. The simulation result can then be visualized inside the Swarm UI. Essentially, by using this component, the user can design the swarm and in real-time get a feeling about its behaviour and make changes and fine tune the design for optimization.

c. Swarm Behaviour and Service Repository: Various deployable artefacts created by the swarm compiler as well as pre-packaged capabilities are stored in the repository. The swarm compiler as well as deployment orchestrator can fetch from the repository to deploy the swarm behaviour defined by the Swarm UI.

4. Operations Support:

   a. Identity and Access Management: To manage the role-based access control for different users within the Swarm UI, Identity and Access Management component is used. User can be assigned different roles and functionalities by the administrators. This is important because not all users should have same level of access to the swarm devices and applications.

   b. Accounts and Charging: Accounts and Charging keeps track of various usage metrics defined by the Swarm UI and related usage metrics from the Device and Application Telemetry. Based on this, the service can be used to create billing information for each account based on usage. This is an optional component which can be used if enabled in the Swarm UI.

Functional View

This section defines in detail the high-level components of the Swarm UI as part of the architecture overview. Various functional components that form the core of the Swarm UI are depicted in the diagram below. Depending on the specific realization of the architecture in an implemented solution some of the components might not be needed, others might be further extended. The following overview draws the big picture of a Swarm UI as part of the reference architecture as defined by the OpenSwarm project.

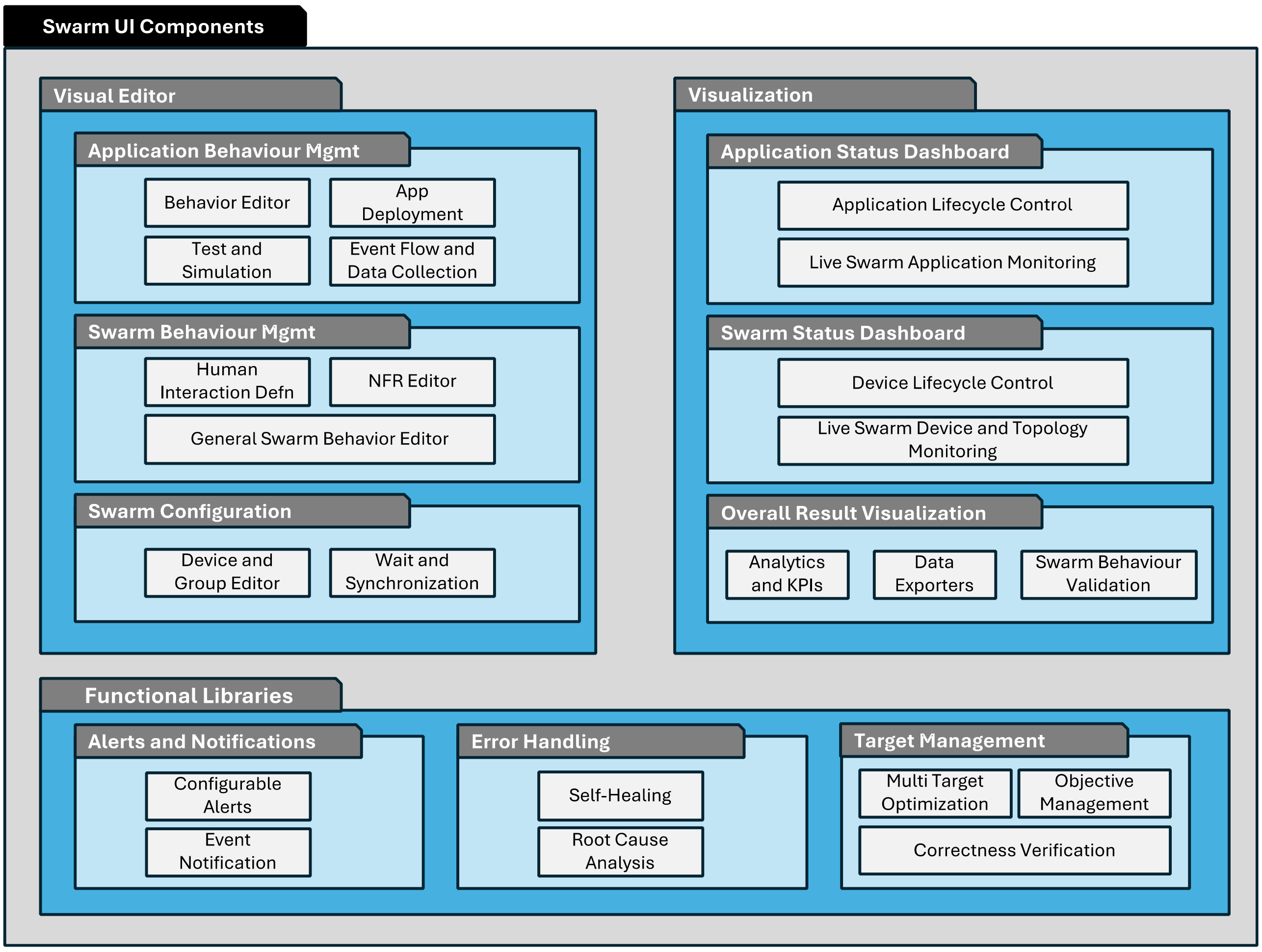


Figure 29 The function decomposition of Swarm UI

Figure 29 shows and illustration of the functional decomposition of the Swarm UI. It consists of the following subcomponents, mapped to the high-level requirements defined in previous sections, as described in the following:

1. Visual Editor:

    a. Application Behaviour Management:

        i. Behaviour editor: This component allows swarm designers to define swarm behaviours utilizing different modelling approaches (e.g., workflows, supervisory control) in a visual manner allowing the composition of swarm behaviours and leveraging a common set of primitive swarm node functionalities.

        ii. Application Deployment Control: This component takes care of deployment of the application on Swarm. User can define device classes, swarm topology and optimal configuration for the application and the deployment control takes that configuration as the basis for application deployment.

iii. Testing and Simulation: Based on the Swarm configuration, application configuration and overall constraints provided, the Swarm simulator provides a simulation-based insight on how the swarm would behave in real world. This is a visualization of swarm moving and functioning with the constraints. The user can trigger test conditions which can add to the simulation and provide further insights. The component provides capability to define acceptance and test criteria for swarm and simulate the behaviour before deployment.

iv. Event Flow and Data Collection Definition: Since various applications can be dependent on each other to complete the overall workflow, or individual applications might emit different events at different stages which might be important for the end user as well as overall application and swarm functioning optimally, this component enables those definitions. User can define for each application various stages and events emitted during those stages.

b. Swarm Behaviour Management:

    i. Human Interaction Definition: As the name suggests, this component allows defining constraints for human interaction with the swarm devices. Whether a human actor is interacting as part of the workflow or the actor in interrupting the workflow, this would affect the swarm behaviour, and the swarm should behave differently for different conditions. All these conditions can be defined inside this configurator.

    ii. General Swarm Behaviour Editor: This configuration allows creating default swarm behaviour and overall behaviour when deploying applications and interacting with objects and actors in real world. For example, when encountering a human, wait, when encountering another robot, find the most optimized crossing path to reach the destination.

iii. Non-Functional Requirements Editor: A configurator allowing users to create rules for the swarm behaviour and defining the non-functional requirements and quality of service is important for the overall Swarm definition. This component provides a visual configurator which can also be switched to a text-based configurator for better control. Additionally, at the time of configuration, a constraint validator also runs and provides insights into whether the designed configurations can be matched by the available Swarm. Once the Swarm is deployed, the constraint validator also checks for optimal deployment mapping the configuration.

c. Swarm Configuration:

i. Device and Device Group Configurator: The component allows creation of device groups, defining rules for optimization of device behaviour within a swarm, grouping rules for device classes, and their requirements. Based on these rules various devices can or can't join a swarm, their responsibilities within the swarm and the kind of tasks they perform is defined.

ii. Waiting and Synchronization: Since applications and devices will be dependent on each other in a complex scenario, it is important to be able to define the waiting conditions, state and synchronization mechanism for the application and devices. In the best-case scenario, everything will function as expected. But in the worst-case scenario or edge cases, synchronization mechanisms become extremely important to optimize the overall workflow. User can define conditions and various synchronization mechanisms for those conditions using this component.

2. Visualization:

a. Application Status Dashboard:

i. Application Lifecycle Control: This component allows defining constraints for the application running on swarm during their lifecycle. Rules like what should an application do when an interruption occurs, what are the completion criteria and so on. Additionally, a user can set up rules to observe and be notified during a specific condition in the lifecycle and make changes if the notification is issued

   ii. Live Swarm Application Monitoring: This is a dashboard with real time application status visualization. The state of the application, devices it is running on, potential KPI mapping and suggestion of improvement are all visualized on this dashboard.

b. Swarm Status Dashboard:

   i. Device Lifecycle Control: This component allows defining constraints for the devices in swarm during their lifecycle. Rules like how to switch devices in case of an issue, when to chare a device, how to replace devices inside swarm to control the minimum acceptance criteria. Additionally, a user can set up rules to observe and be notified during a specific condition in the lifecycle and make changes if the notification is issued

   ii. Live Swarm Device and Topology Monitoring: This is a dashboard with real-time data visualizing the swarm topology. It takes input from the swarm telemetry and provides insight on the swarm topology. It can also be made configurable so a user can define exact KPIs that they want to monitor. It also shows various devices and their status and metadata like location, battery status etc.

c. Overall Result Visualization:

   i. Analytics and KPI Dashboards: These configurable dashboards are based on various rules a user can set to monitor the overall functioning of the Swarm. This is different than previous two dashboards in that it can be fully configured to display queries and results created by the user.

   ii. Monitoring Data Export Controls: This component allows user to export certain data sets from the overall analytics data collected from Swarm. User can configure which data to export and how often can it be exported. The data set can be used, for example, to create periodic reports if needed.

iii. Swarm Behaviour Validation: Behaviour validation allows user to check and compare real time swarm behaviour with the expected behaviour based on the configurations from "Swarm Behaviour Management" components. Interrupts and error handling, self-healing and overall functioning of the swarm is constantly governed by the configurations, but in case of deviance from defined behaviour, this component allows quick notification and validation to the user. User can also check for cause of the deviation based on the Root Cause Analysis component from Error Handling.

3. Functional Libraries:
    a. Alerts and Notifications:
        i. Configurable Alerts: Common component that connects with telemetry data from device and swarm and runs the rules defined by the user on top of that data. Based on the rules, specific events are notified as alert to users. It allows users to configure those rules. The component can be integrated with any visualization component mentioned above.
        ii. Event Notification: Notification components define and enables notifying the users based on alerts. User can configure how frequently they want to receive notifications, the priority of notifications (low, medium and high) and the mode of delivery (dashboard, email or SMS).
    b. Error Handling:
        i. Self-Healing: The system should implement self-healing functionalities that will increase its robustness by allowing it to adapt to changing conditions and errors, thus maintaining the system operational. This function will work in close interaction with relevant Telemetry and Lifecycle management functionalities. On

the one hand Telemetry information will be used to determining potential failures (e.g., node, communication, battery) and Lifecycle management controls will be used for dynamically moving tasks across the Swarm.

   ii. Root Cause Analysis: Upon a given failure, notified by telemetry systems or alerted by the Correctness Verification functionalities, this function should be able to identify the root cause for this failure and provide detailed information to the self-healing functionalities to take the adequate countermeasures.

c. Target Management:

   i. Multi Target Optimization: Target manager is an optimization algorithm that maps tasks and applications to the optimal swarm cluster and device based on cluster defined rules by the user.

   ii. Objective Management: The overall high-level objectives provided as part of the Swarm Behaviour Definition are mapped into concrete low-level targets and fed to the Multi-Target Optimization function for actual deployment decisions. This information is also provided to the Correctness Verification function to ensure that the set objectives are achieved.

   iii. Correctness Verification: Once the swarm is configured and application behaviour is defined, the question before deployment is if the overall configuration is functionally correct and can deliver expected results. The functional correctness verification component verifies the final configuration with help of defined acceptance criteria.

Overall, these components together with the external components defined in the system context provide a basis for the Swarm UI.

Swarm UI Implementation

Now that we have analysed the motivation and perspectives behind the design of Swarm UI, derived several requirements, and defined a reference architecture, we move on to the discussion of implementation aspects. In this section, we start by looking at existing artifacts, such as visual and low-code programming tools, that implement features that could potentially be reused in a custom Swarm UI implementation. We evaluate their complexity and feature sets. Then, we move on to discussing four OpenSwarm User Interface prototypes that were created to derive further requirements and evaluate the ideas described in previous sections.

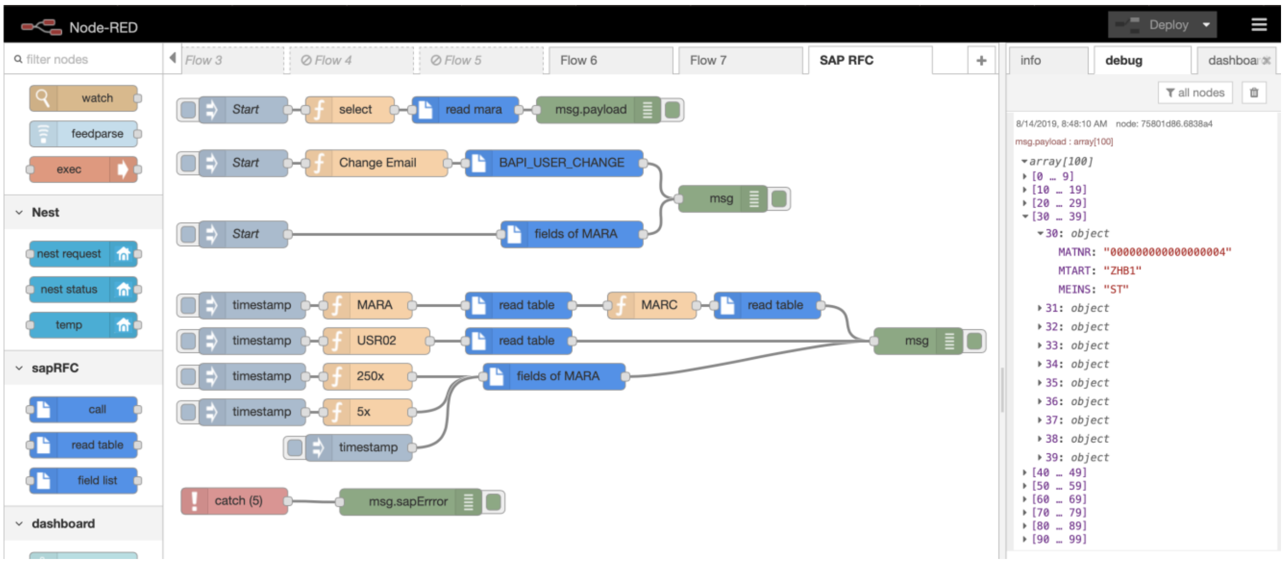
Evaluation of Artifacts for Swarm UI Implementation

To implement the Swarm UI, multiple existing tools and frameworks can be reused to speed up the development. In this section, we will discuss the relevant popular artifacts that could be used to realize building blocks and components of such a UI. We will look at the most important aspects that need to be taken into consideration when choosing such a solution.

To compile this evaluation, a number of frameworks for visual programming and low-code have been evaluated. These frameworks include the following representative examples: Node-RED [67], total.js [68], Reactflow [69] and Apache Airflow [70]. We focused on tools that are widely adopted in the industry and that are still being maintained as of this day.

To compare and evaluate these tools we looked at key properties, such as the packaging format, ease-of-setup functionality, technology stack, commercial versions, customizability and licensing.

Node-RED (Figure 30**)** is low-code flow programming software for wiring hardware, APIs and online services together. We include it here as an artifact, as it can be easily extended with new functionalities and reused to build new UI applications. Thanks to a large library of pre-built nodes and plugins, it provides an easy-to-use platform for visual programming of workflows. Due to its native integration with many network protocols (such as MQTT, WS), it allows to easily glue together multiple services into automated units of work. Node-RED is accessed through the browser from a server and its source and bundles are provided under the permissive open-source Apache 2.0 license. The tool is built in Javascript, which means that it can easily be modified through simple scripting. Finally, due to its common usage in industry, it is still being actively maintained and developed currently.



```
8/14/2019, 8:48:10 AM   node: 75801d86.6838a4
msg.payload : array[100]
▾array[100]
 ▸[0 … 9]
 ▸[10 … 19]
 ▸[20 … 29]
 ▾[30 … 39]
   ▾30: object
      MATNR: "000000000000000004"
      MTART: "ZHB1"
      MEINS: "ST"
   ▸31: object
   ▸32: object
   ▸33: object
   ▸34: object
   ▸35: object
   ▸36: object
   ▸37: object
   ▸38: object
   ▸39: object
 ▸[40 … 49]
 ▸[50 … 59]
 ▸[60 … 69]
 ▸[70 … 79]
 ▸[80 … 89]
 ▸[90 … 99]
```

Figure 30: Node-RED GUI [71]

Total.js (Figure 31**)** is another low-code flow programming platform that can be used to visually program applications from existing components. It offers similar functionality as Node-RED but extends with additional features for building UI mock-ups using an UI builder. Using custom Javascript functions, its functionality can be easily extended. It allows to easily model flows for frontend applications. It's offered under the MIT license, with a Pro paid version that runs as a Cloud SaaS service. The free version lacks certain features, such as integration with existing CMS solutions and other ubiquitous platforms, but contains the core functionality of this framework.

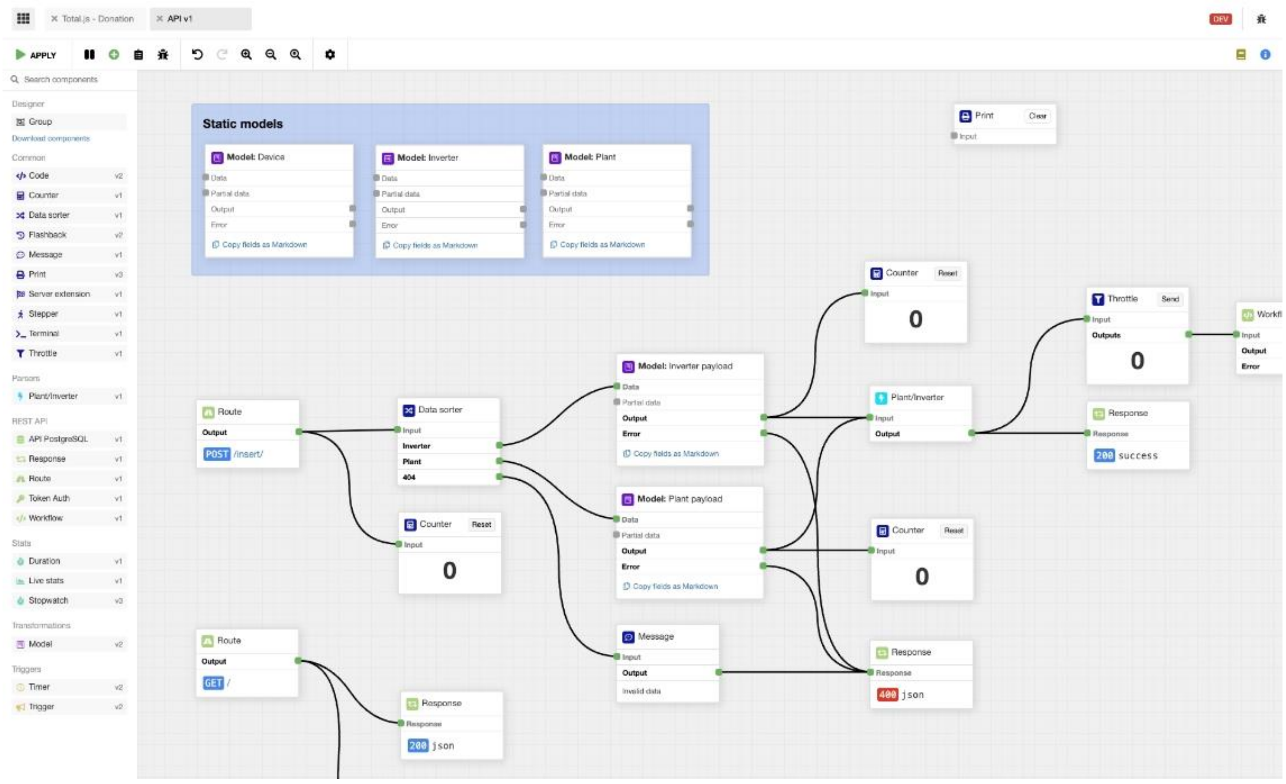


Figure 31: total.js GUI [72]

Reactflow (Figure 32**)** forms a category different than the two tools described previously, as it is simple Javascript library that allows to build applications that use an infinite canvas to place nodes and connect them using edges. It is built on the React programming framework, but is also available for other frontend solutions, such as Vue and Svelte. It does not come with pre-built nodes and functionalities, as these need to be implemented by the adopter of the technology. Thanks to its MIT license and common adoption in the industry, it is a promising library that could be used for swarm programming UI. As it also provides a commercial version called FlowFuse, it is still being actively developed and maintained.

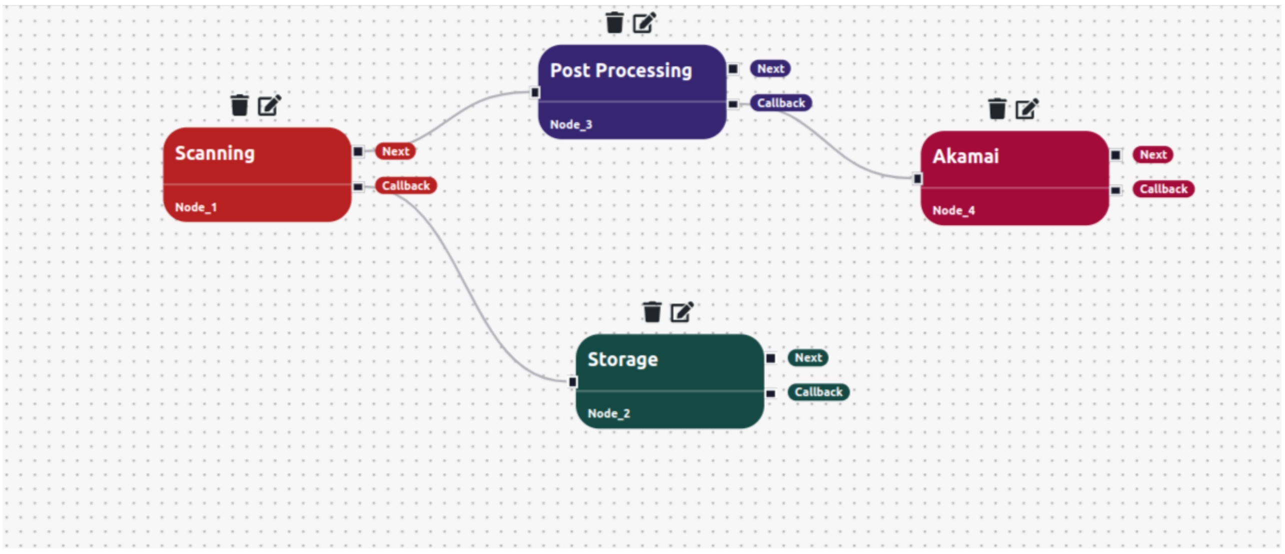


Figure 32: ReactFlow GUI [73]

The last tool, **Apache Airflow (**Figure 33**)**, is a framework for workflow orchestration, which could also be seen as a building block for new UI applications that need low-code and visual interactions functionality. It is most commonly used for data processing tasks and is being actively developed and maintained in the industry. It is packaged as a set of components, such as Database, Server, etc., that need to be deployed and connected to use the tool. It is released under the open-source Apache 2.0 license and does not offer a commercial version. Its functionality can be easily extended through various plugins that are primarily built in Python.

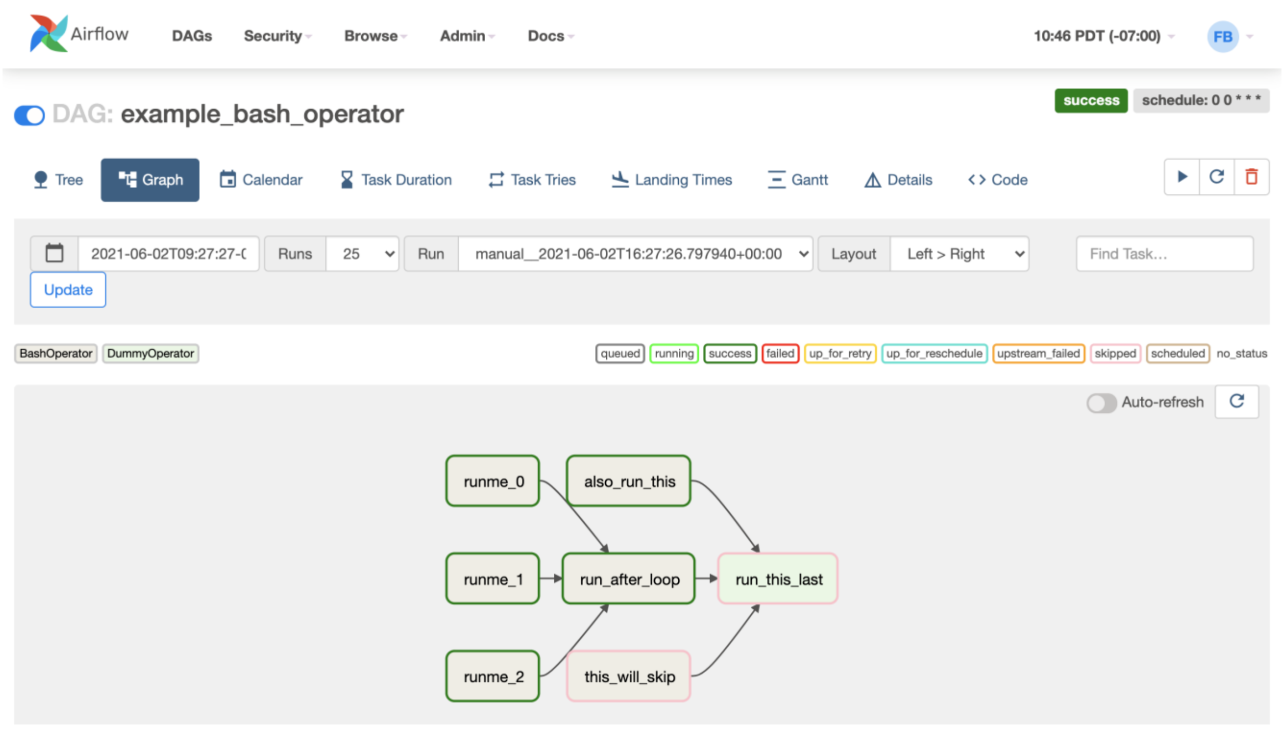


Figure 33: Apache Airflow GUI [74]

Summing up, when choosing the tool for implementing the Swarm Programming UI, aspects such as customizability, packaging, etc. should be considered. From the discussed tools, ReactFlow is the most promising for applications that require full customizability, as it only provides a basic library for basic visual drag-and-drop interactions. From complete solutions, Node-RED stands out as the most mature and widely used platform.

OpenSwarm User Interfaces

This section introduces the OpenSwarm User Interfaces developed for addressing specific architectural requirements towards the realization of the WP6 proof-of-concepts and the Kilobot Testbed. Each UI is described and put into context with respect to the key architectural requirements and lifecycle phases of a Swarm UI.

New UI for the Augmented Reality for Kilobots (ARK)

Description: This work concerns the improvement of the UI of the Augmented Reality for Kilobots (ARK) system to include the missing UI requirements described earlier. As shown in Figure 34. The new UI provides the user with 3 buttons to Design, Compile, and Upload a controller to the robots.

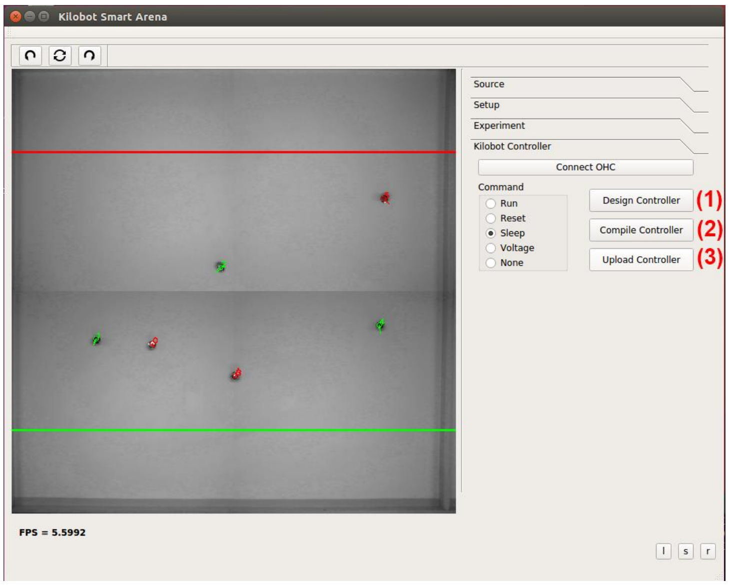
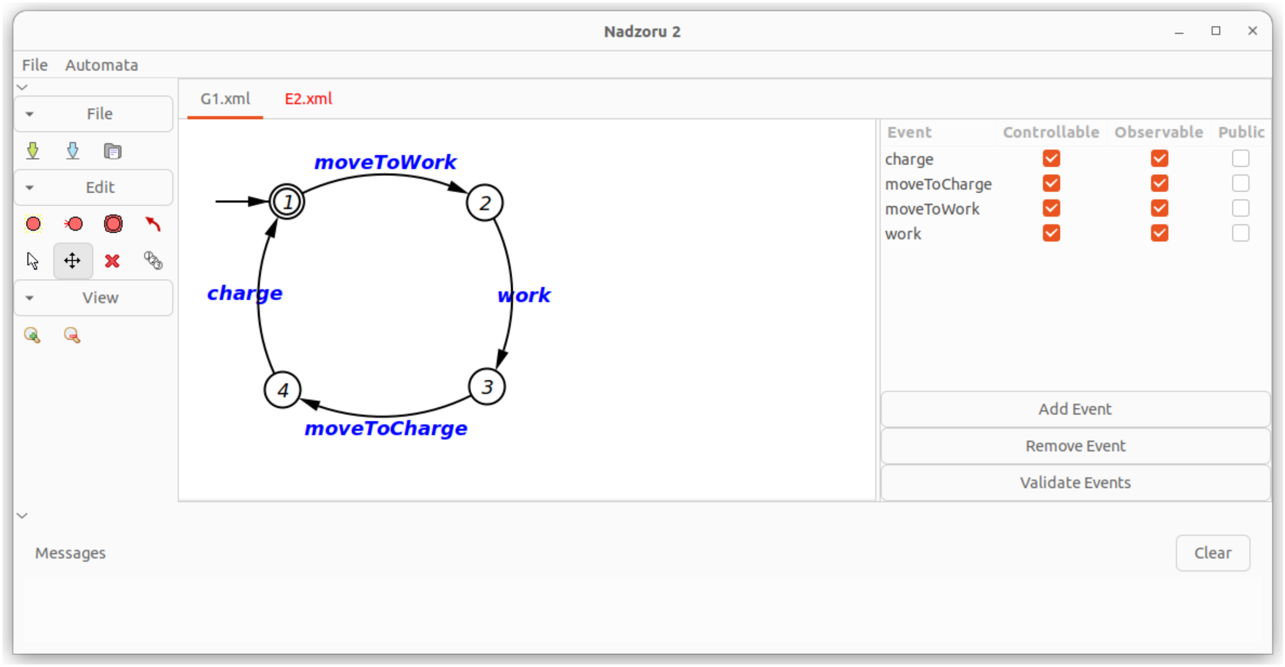


Figure 34. New ARK UI providing the user with 3 buttons: (1) The "Design Controller" button allowing the user to select a folder where the controller will be stored and opens the Nadzuro2 tool where the user can formally design a swarm controller. (2) The "Compile Controller" button that automatically compiles the designed controller. (3) The "Upload Controller" button that sends the compiled controller to the robots.



Event	Controllable	Observable	Public
charge	✓	✓	☐
moveToCharge	✓	✓	☐
moveToWork	✓	✓	☐
work	✓	✓	☐

Figure 35. Nadzoru 2 interface. The user can define the capabilities and specifications of a robot using models within the supervisory control theory framework.

The controller design is performed under the Nadzuro 2 tool developed in T4.3. Figure 35 demonstrates an interface where a swarm operator could perform most of the necessary steps to deploy a robot swarm, which includes designing the robot's control logic, creating source code that correctly implements the modelled control logic.

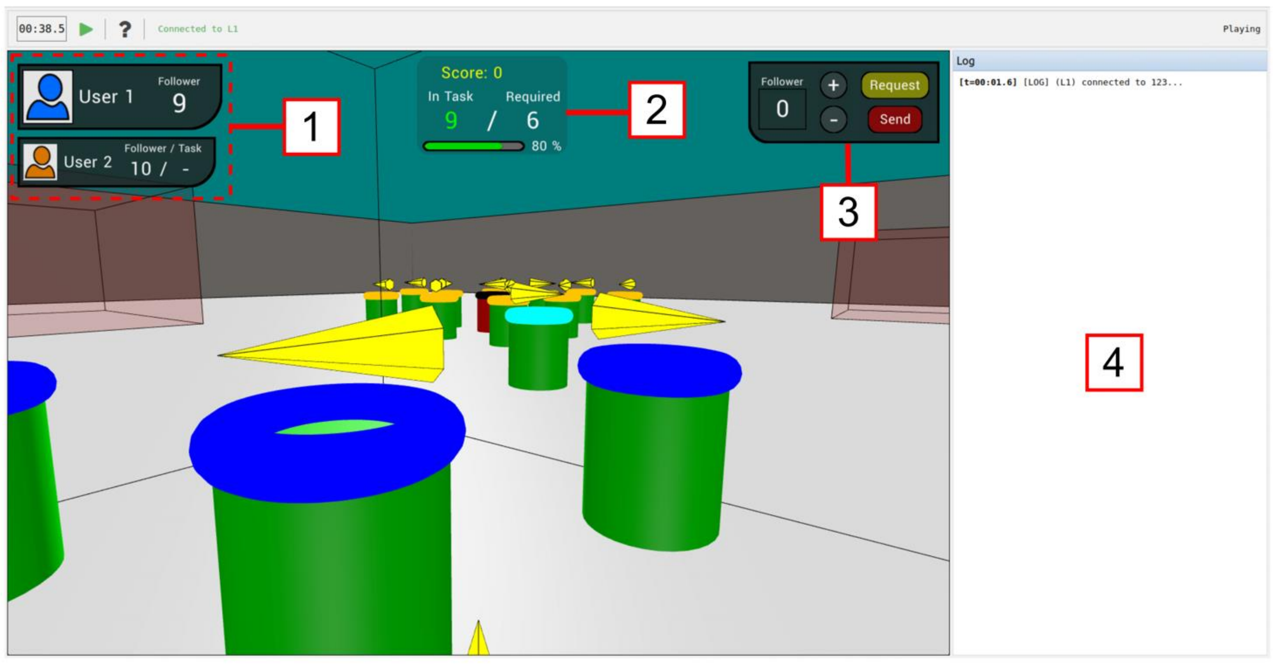
Key features:

- End-to-End development platform that can manage from modelling, building, and deployment of a swarm of Kilobots.
- Graphical user interface to easily add, remove, drag and drop states and events.
- Design models that represent the capabilities and specifications of a system.
- Synthesise models to obtain the control logic.
- Export the control logic by automatically generating partially completed source code that can directly be used on a robot (template source code must be present).

Life-cycle Phase(s) addressed: Programming, Engineering, Operation

Architectural Components addressed: KAR-3, KAR-4, KAR-7, KAR-8

Swarms Operations with Multiple Operators



```
[t=00:01.6] [LOG] (L1) connected to 123...
```

Figure 36. Interface for a user to explore a simulated environment while being followed by a swarm of robots. Two users can share robots with each other by sending them along an ad-hoc network maintained by the robots.

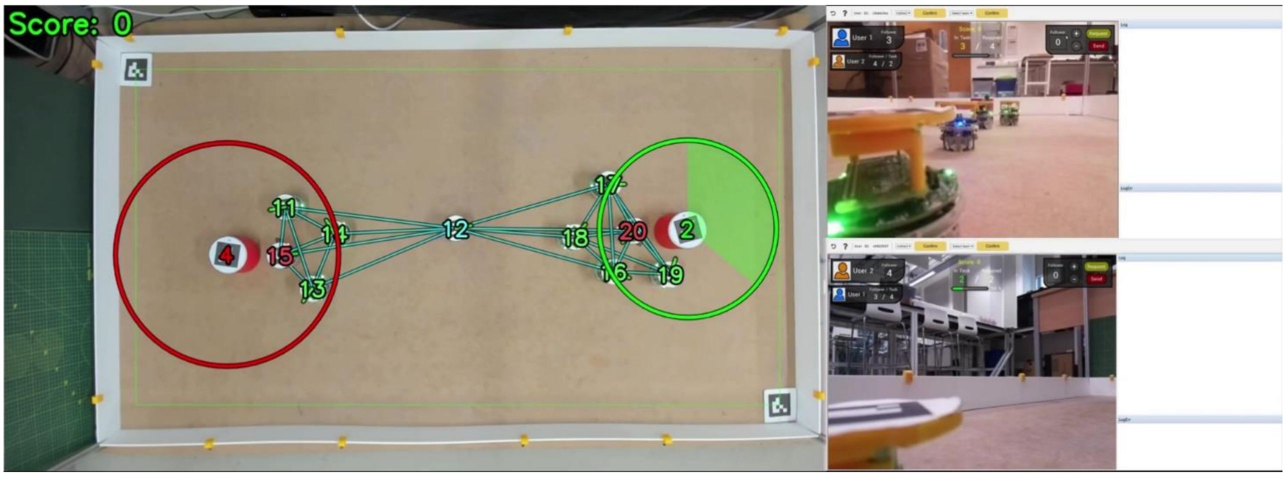


Figure 37. Demonstration using pi-puck robots in the lab. Two users each controlled a leader robot equipped with a Raspberry Pi Camera. The goal was to guide the robots inside circle. Overhead camera view of the environment, which was not visible to the users (right). The user interfaces each controlled by a user (left).

Description: This work concerns the development of a novel user interface supporting multiple operators to share control over a swarm during its operation.

Key features:

- User interface that provides a first-person perspective in the environment in which the swarm operates (see Figure 36).
- Four control panels that a user interacts with; (1) team information panel; (2) task information panel; (3) request-and-send panel; (4) log panel.
- Ability for a user to request robots from or send robots to other users.
- Inform the user the number of robots that they are in control of.
- User study conducted with 52 participants.
- Integration with real robot platforms (see Figure 37).

Life-cycle Phase(s) addressed: Operation

Architectural Components addressed: KAR-2, KAR-5

Combined Real and Digital Environment Modelling

Description: As already described in the user stories, in the case of the Energy PoC, the components are not mobile in terms of space, but rather in terms of assignment to an energy community or even the presence within the energy system.

To demonstrate how individual components can be added to the system as part of the OpenSwarm project, there are two options:

1. With the help of the BIFROST simulation software (see section "State of the Art in Human-Swarm Interaction"), energy scenarios can be presented in a simple and visually appealing way. The open data model in combination with a web-based UI and 2.5D visualization allows for a relatively simple definition of these scenarios. (Figure 38)
2. Some of the elements from the BIFROST simulation were transferred into reality in the form of 3D models and offer the possibility of running components of the OpenSwarm system within these representations. By adding elements or changing their position, the scenario can now be easily changed. This modified system is then recreated in BIFROST so that the energy system can be simulated correctly. However, the OpenSwarm components can react to their environment and process their assigned tasks as in the real system. (Figure 39)

The combination of the BIFROST simulation software with real-world implementations of components of the electrical energy system ("Real Twin" for short) is somewhat further removed from the concept of a user interface for programming of the swarm behaviour. The programming of the swarm behaviour must be done here again in a separate UI. The Real Twin is more a possibility to set up the environment, in which the swarm of PoC1 has to solve its tasks. However, since the PoC can be displayed in a testbed in this way, it is possible to check the effectiveness and applicability of the developed Swarm UI Reference Architecture here.

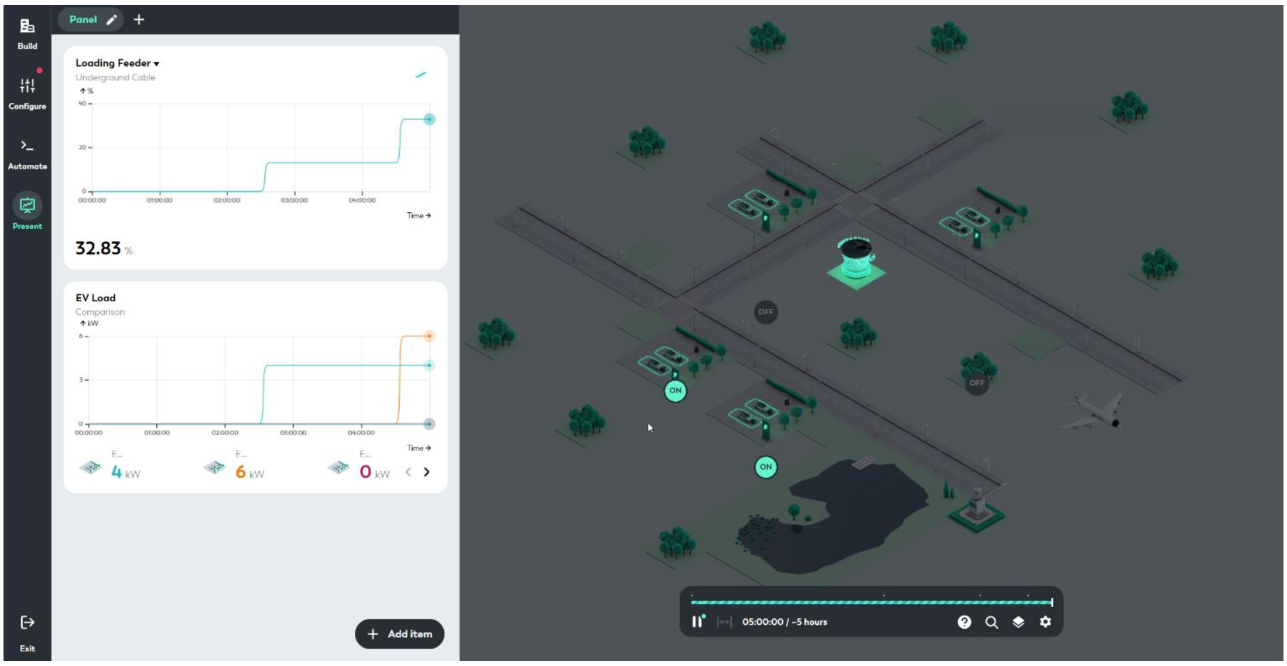


Figure 38: BIFROST UI for setting up energy scenarios

Figure 38 shows an implemented energy system scenario with several charging stations whose charging process can be started using the UI. A real system state can be represented by a simulation running in the background and used to examine swarm behaviour.

Figure 39 and Figure 40 show the part of the energy system simulation that is based in reality. The 3D-printed elements can be arranged on a base plate. Various other elements allow interaction with the energy system: vehicles can be added to charging stations, which starts charging processes in the simulation. Storage systems can be "loaded" with additional storage elements: the simulation recognizes that the storage system has been increased by the increase in weight. Houses can be equipped with PV systems and show their current energy consumption on a display.

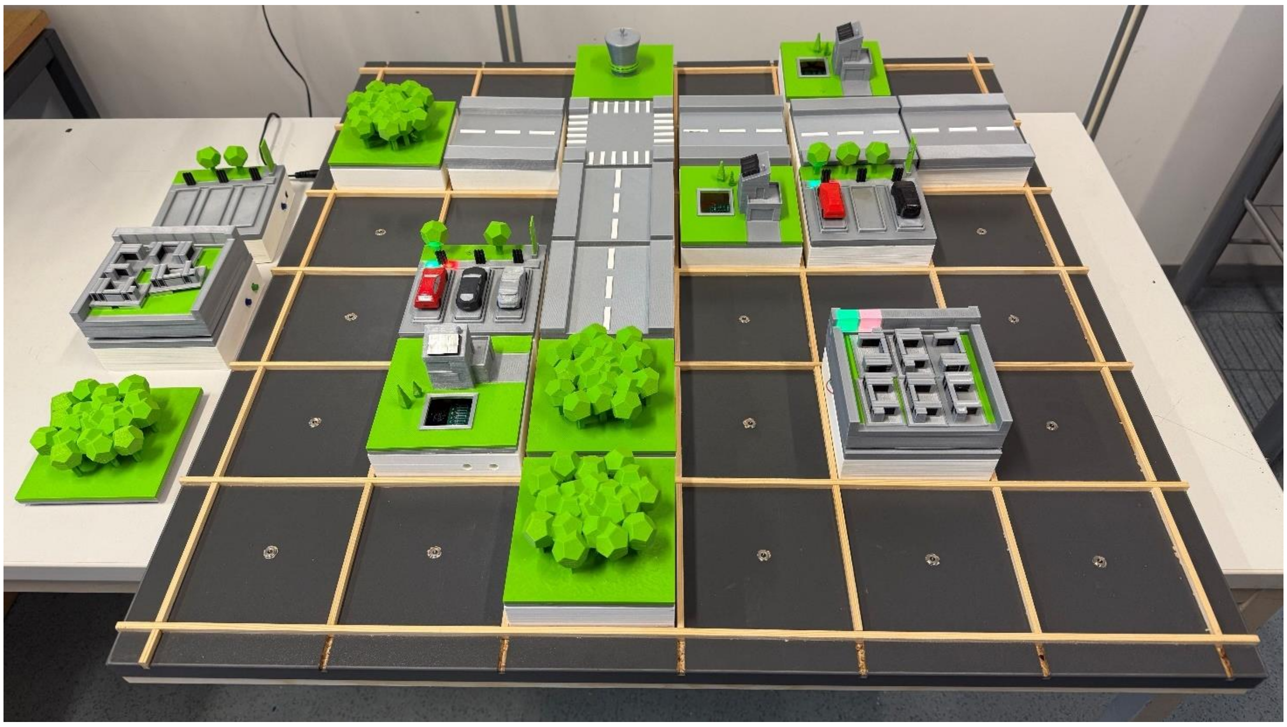


Figure 39: Real-world implementations of components of the electrical energy system ("Real Twin" for short)

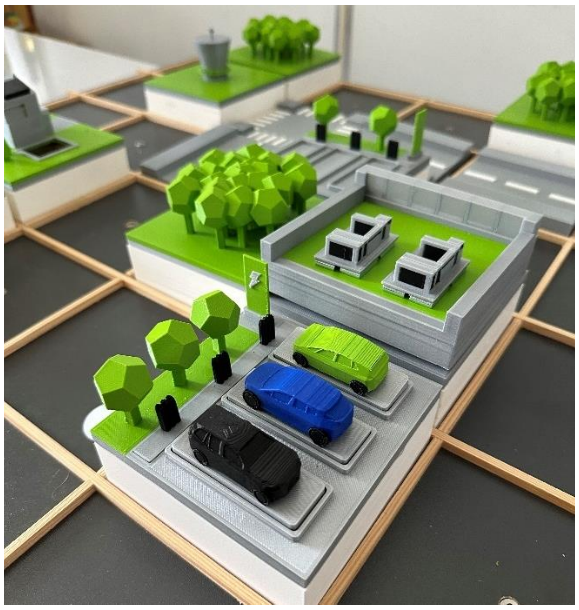


Figure 40: Some of the elements (Charging station, battery storage system)

Furthermore, hardware components and corresponding software from the developed OpenSwarm solutions can be integrated into the 3D-printed elements. The energy system simulation can provide these measurement data as if they had been integrated into a real grid.

The combination of simulation and real construction makes it possible to easily and efficiently visualize and review the user stories from PoC1. Of course, this UI is not used to program the swarm, but it is a good way to visualize and manipulate this particular form of swarm found in PoC1.

Key features:

- Combination of simulation (BIFROST) and physical "Real Twin" components.
- Scenario definition and visualization through the BIFROST web-based UI.
- Interaction with physical 3D-printed elements representing the energy system.
- Testbed for validating the Swarm UI Reference Architecture.

Life-cycle Phase(s) addressed: Operation (Testing, Demonstration)

Architectural Components addressed: Since this is not a swarm programming user interface, there is no reference to the reference architecture.

Web-Based Behaviour and Policy Modelling

Description: This work is focused on creating a novel web-based interface for swarm programming, that fulfils the proposed key architectural requirements and can be used for behaviour and policy modelling in e.g. the EHS PoC. Based on the preliminary research into existing visual programming tools (see **Evaluation of Artifacts for Swarm UI Implementation**), we have decided to build a prototype using ReactFlow and the React web-programming framework as basis. This choice was motivated by the maturity of both solutions and their extendibility and unopinionated architecture. However, the research into other solution has proven beneficial, as they have served as a source of inspiration when defining the roadmap and minimal viable product design.

On of the key objectives when developing the new swarm programming user interface has been the reusability for various modelling approaches, and the flexibility in selecting the deployment and device target scenarios. Thus, to ground the prototype in reality, we have selected EHS PoC as the exemplary scenario for requirements elicitation. Supervisory Control Theory (SCT) was selected as the modelling language for the use case, with major inspirations taken from Nadzoru developed at UoS. In the future other visual modelling tools such as Behaviour Trees can also be explored within the created prototype.

To verify modelling use cases, a prototypical UI for SCT was implemented (see Figure 41), which allows to model Control Specifications and Behaviours. Using a simple drag and drop of nodes from a sidebar onto an infinite canvas, a user can model the intended behaviour of his application. By connecting the nodes together, relationships can be formed. Node names and properties can also be modified by the user, using standard interaction patterns, such as right-click.

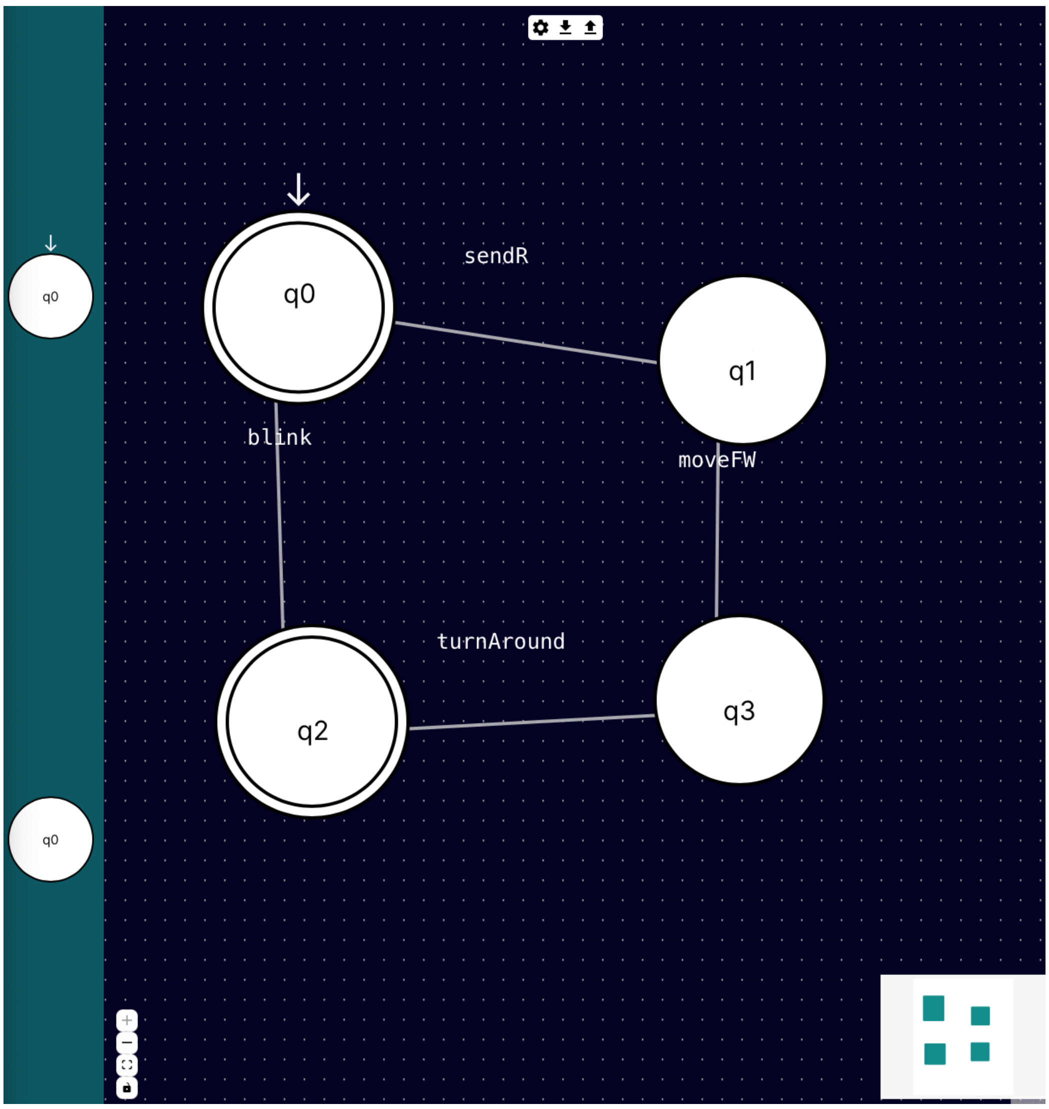


Figure 41: Screenshot of the developed prototype, with 4 nodes and 4 edges. On the top settings, download and upload buttons can be found. In the bottom left corner, the infinite canvas can be operated by zooming in and out. In the right bottom corner, a minimap of present nodes is shown.

The resulting model can then be exported in form of a JSON file and again reimported at a future point of time. The exported data also contains information required to recreate the exact look and feel of the previous session, by also persisting data such as positioning of nodes on the canvas.

Within the prototype, key components identified withing KAR-3 were implemented. Through the modular design, with clearly separated components, more elaborate user interfaces can be created in the future. An exploration was also conducted into aspects such as resizing of nodes to fulfil use cases (also mentioned in KAR-5) such as marking of certain zones with certain properties, which has proven that the selected frameworks are also capable of implementing such use case scenario. We have also successfully verified features such as: visual notification system using "toasts", which allows to display context dependent and real-time notifications as temporary blocks within the editor, allowing the system to propagate important information directly to the user.

Features found in other modelling tools, such as in Node-RED or Total.js, such as browser tabs synchronization and possibility of opening multiple tabs within the browser window were also successfully tested and verified. Moreover, accessibility aspect was also tested, by verifying that the tool can be used on mobile devices with touch screen. In the future ubiquitous features of web applications such as access control should be implemented.

We intend to open-source the created prototype together with a guide on how to extend and use it. In the future we intend to extend the prototype based on the specific requirements derived from working on the PoCs. Finally, we intend to host the application using a CI/CD pipeline to allow any project participants to test it and provide feedback.

Key features:

- Application with modular React components that can be composed together to create visual modelling tools with different graph-oriented modelling languages based on ReactFlow, together with examples on how to use them
- Prototypes already supports applications such as: modelling workflows based on graphs, applications for visually marking areas, etc.
- Features commonly found in other low-code tools, such as persistence of session and synchronization within different tabs are implemented and tested.
- Simple SCT models can be created and exported as JSON format.

Life-cycle Phase(s) addressed: Programming, Engineering, Operation

Architectural Components addressed: KAR-3, KAR-4, KAR-5, KAR-7, KAR-8

Conclusion

This deliverable provides a comprehensive overview of the development and implementation of a Swarm User Interface (UI). By exploring the state of the art in swarm programming, behaviour modelling, and model-based design, we have identified key challenges and opportunities in creating effective and intuitive interfaces for swarm systems.

By examining relevant user stories and carefully analysing the system context and problems associated with a Swarm UI, we were able to define a basic set of key architectural requirements, that we later translated into a reference architecture that addresses these key requirements. The reference architecture provides a high-level view of the system context and functional components necessary for effective swarm management and operation. The implementation of the OpenSwarm User Interfaces demonstrates the practical application of the proposed theoretical framework.

The findings from this task lay the groundwork for future research and development in swarm UI design. By continuing to refine and expand upon the work conducted in this task, we can unlock new possibilities for the realization of swarm systems in various proof-of-concepts in WP6.

Bibliography

[1] A. Kolling, P. Walker, N. Chakraborty, K. Sycara, and M. Lewis, ‘Human Interaction with Robot Swarms: A Survey’, *IEEE Trans. Hum.-Mach. Syst.*, vol. 46, no. 1, pp. 9–26, 2016, doi: 10.1109/THMS.2015.2480801.

[2] D. R. Olsen and S. B. Wood, ‘Fan-out: measuring human control of multiple robots’, in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, in CHI ’04. Vienna, Austria: Association for Computing Machinery, Apr. 2004, pp. 231–238. doi: 10.1145/985692.985722.

[3] G. Podevijn, R. O’Grady, N. Mathews, A. Gilles, C. Fantini-Hauwel, and M. Dorigo, ‘Investigating the effect of increasing robot group sizes on the human psychophysiological state in the context of human–swarm interaction’, *Swarm Intell.*, vol. 10, no. 3, pp. 193–210, 2016, doi: 10.1007/s11721-016-0124-3.

[4] D. R. Olsen and M. A. Goodrich, ‘Metrics for Evaluating Human-Robot Interactions’, *Proc. PERMIS*, vol. 2003, 2003.

[5] P. Walker, S. Nunnally, M. Lewis, A. Kolling, N. Chakraborty, and K. Sycara, ‘Neglect benevolence in human control of swarms in the presence of latency’, in *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2012, pp. 3009–3014. doi: 10.1109/ICSMC.2012.6378253.

[6] S. Nagavalli, S.-Y. Chien, M. Lewis, N. Chakraborty, and K. Sycara, ‘Bounds of Neglect Benevolence in Input Timing for Human Interaction with Robotic Swarms’, in *2015 10th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, Mar. 2015, pp. 197–204.

[7] J. L. Drury, J. Scholtz, and H. A. Yanco, ‘Awareness in human-robot interactions’, in *SMC’03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics*, IEEE, Oct. 2003, pp. 912–918. doi: 10.1109/ICSMC.2003.1243931.

[8] M. R. Endsley, ‘Toward a Theory of Situation Awareness in Dynamic Systems’, *Hum. Factors*, vol. 37, no. 1, pp. 32–64, Mar. 1995, doi: 10.1518/001872095779049543.

[9] G. Kapellmann-Zafra, N. Salomons, A. Kolling, and R. Groß, 'Human-Robot Swarm Interaction with Limited Situational Awareness', in *Swarm Intelligence*, M. Dorigo, M. Birattari, X. Li, M. López-Ibáñez, K. Ohkura, C. Pinciroli, and T. Stützle, Eds., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 125–136. doi: 10.1007/978-3-319-44427-7_11.

[10] T. B. Sheridan, *Humans and automation: System design and research issues.* in Wiley series in system engineering and management: HFES issues in human factors and ergonomics series. Santa Monica, CA, US: Human Factors and Ergonomics Society, 2002.

[11] A. Hussein and H. Abbass, 'Mixed Initiative Systems for Human-Swarm Interaction: Opportunities and Challenges', in *2018 2nd Annual Systems Modelling Conference (SMC)*, Oct. 2018, pp. 1–8. doi: 10.1109/SYSMC.2018.8509744.

[12] M. Lewis, J. Wang, and P. Scerri, 'Teamwork Coordination for Realistically Complex Multi Robot Systems', in *NATO Symposium on Human Factors of Uninhabited Military Vehicles as Force Multipliers*, 2006, pp. 1–12.

[13] A. Kolling, S. Nunnally, and M. Lewis, 'Towards human control of robot swarms', in *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, in HRI '12. Boston, Massachusetts, USA: Association for Computing Machinery, Mar. 2012, pp. 89–96. doi: 10.1145/2157689.2157704.

[14] A. Kolling, K. Sycara, S. Nunnally, and M. Lewis, 'Human-swarm interaction: an experimental study of two types of interaction with foraging swarms', *J. Hum.-Robot Interact.*, vol. 2, no. 2, pp. 103–129, Jun. 2013, doi: 10.5898/JHRI.2.2.Kolling.

[15] S. Bashyal and G. K. Venayagamoorthy, 'Human swarm interaction for radiation source search and localization', in *2008 IEEE Swarm Intelligence Symposium*, Sep. 2008, pp. 1–8. doi: 10.1109/SIS.2008.4668287.

[16] C. Miller, H. Funk, P. Wu, R. Goldman, J. Meisner, and M. Chapman, 'The Playbook™ Approach to Adaptive Automation', *Proc. Hum. Factors Ergon. Soc. Annu. Meet.*, vol. 49, no. 1, pp. 15–19, Sep. 2005, doi: 10.1177/154193120504900105.

[17] J. W. Crandall *et al.*, 'Human-Swarm Interaction as Shared Control: Achieving Flexible Fault-Tolerant Systems', in *Engineering Psychology and Cognitive Ergonomics:*

Performance, Emotion and Situation Awareness, D. Harris, Ed., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 266–284. doi: 10.1007/978-3-319-58472-0_21.

[18] P. Walker, S. A. Amraii, M. Lewis, N. Chakraborty, and K. Sycara, ‘Human Control of Leader-Based Swarms’, in *2013 IEEE International Conference on Systems, Man, and Cybernetics*, 2013, pp. 2712–2717. doi: 10.1109/SMC.2013.462.

[19] I. D. Couzin, J. Krause, R. James, G. D. Ruxton, and N. R. Franks, ‘Collective Memory and Spatial Sorting in Animal Groups’, *J. Theor. Biol.*, vol. 218, no. 1, pp. 1–11, Sep. 2002, doi: 10.1006/jtbi.2002.3065.

[20] P. Walker, S. A. Amraii, M. Lewis, N. Chakraborty, and K. Sycara, ‘Control of swarms with multiple leader agents’, in *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2014, pp. 3567–3572. doi: 10.1109/SMC.2014.6974483.

[21] P. Walker, S. A. Amraii, N. Chakraborty, M. Lewis, and K. Sycara, ‘Human control of robot swarms with dynamic leaders’, in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 1108–1113. doi: 10.1109/IROS.2014.6942696.

[22] M. A. Goodrich, B. Pendleton, S. Kerman, and P. Sujit, ‘What types of interactions do bio-inspired robot swarms and flocks afford a human?’, in *Proc. Robot.: Sci. Syst. VIII*, 2013, pp. 105–112.

[23] J. Nagi, A. Giusti, L. M. Gambardella, and G. A. Di Caro, ‘Human-swarm interaction using spatial gestures’, in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2014, pp. 3834–3841. doi: 10.1109/IROS.2014.6943101.

[24] J. Patel, Y. Xu, and C. Pinciroli, ‘Mixed-Granularity Human-Swarm Interaction’, in *2019 International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 1059–1065. doi: 10.1109/ICRA.2019.8793261.

[25] I. Jang, J. Hu, F. Arvin, J. Carrasco, and B. Lennox, ‘Omnipotent Virtual Giant for Remote Human–Swarm Interaction’, in *2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN)*, Aug. 2021, pp. 488–494. doi: 10.1109/RO-MAN50785.2021.9515542.

[26] A. Prabhakar *et al.*, 'Ergodic Specifications for Flexible Swarm Control: From User Commands to Persistent Adaptation', in *Robotics: Science and Systems XVI*, Robotics: Science and Systems Foundation, Jul. 2020. doi: 10.15607/RSS.2020.XVI.067.

[27] H. Hexmoor, B. Mclaughlan, and M. Baker, 'Swarm Control in Unmanned Aerial Vehicles', in *Proceedings of the International Conference on Artificial Intelligence*, Army Research Laboratory, 2005, pp. 911–917.

[28] J. Wilson *et al.*, 'Trustworthy Swarms', in *Proceedings of the First International Symposium on Trustworthy Autonomous Systems*, in TAS '23. New York, NY, USA: Association for Computing Machinery, Jul. 2023, pp. 1–11. doi: 10.1145/3597512.3599705.

[29] P. Walker, M. Lewis, and K. Sycara, 'The effect of display type on operator prediction of future swarm states', in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2016, pp. 2521–2526. doi: 10.1109/SMC.2016.7844619.

[30] K. A. Roundtree, M. D. Manning, and J. A. Adams, 'Analysis of Human-Swarm Visualizations', *Proc. Hum. Factors Ergon. Soc. Annu. Meet.*, vol. 62, no. 1, pp. 287–291, Sep. 2018, doi: 10.1177/1541931218621066.

[31] M. Divband Soorati, J. Clark, J. Ghofrani, D. Tarapore, and S. D. Ramchurn, 'Designing a User-Centered Interaction Interface for Human–Swarm Teaming', *Drones*, vol. 5, no. 4, Art. no. 4, Dec. 2021, doi: 10.3390/drones5040131.

[32] K. A. Roundtree, J. R. Cody, J. Leaf, H. O. Demirel, and J. A. Adams, 'Human-collective visualization transparency', *Swarm Intell.*, vol. 15, no. 3, pp. 237–286, Sep. 2021, doi: 10.1007/s11721-021-00194-6.

[33] A. L. Christensen *et al.*, 'The HERD Project: Human-Multi-Robot Interaction in Search & Rescue and in Farming', *Adjun. Proc. IEEERSJ Int. Conf. Intell. Robots Syst.*, pp. 1–4, 2022.

[34] B. Gromov, L. M. Gambardella, and G. A. Di Caro, 'Wearable multi-modal interface for human multi-robot interaction', in *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Oct. 2016, pp. 240–245. doi: 10.1109/SSRR.2016.7784305.

[35] J. D. Rockbach and M. Bennewitz, 'Robot swarms as embodied extensions of humans', *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 1261, no. 1, p. 012015, Oct. 2022, doi: 10.1088/1757-899X/1261/1/012015.

[36] C. L. Nehaniv, K. Dautenhahn, J. Kubacki, M. Haegele, C. Parlitz, and R. Alami, 'A methodological approach relating the classification of gesture to identification of human intent in the context of human-robot interaction', in *ROMAN 2005. IEEE International Workshop on Robot and Human Interactive Communication, 2005.*, Aug. 2005, pp. 371–377. doi: 10.1109/ROMAN.2005.1513807.

[37] J. Alonso-Mora, S. Haegeli Lohaus, P. Leemann, R. Siegwart, and P. Beardsley, 'Gesture based human - Multi-robot swarm interaction and its application to an interactive display', in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 5948–5953. doi: 10.1109/ICRA.2015.7140033.

[38] G. Podevijn, R. O'Grady, Y. S. G. Nashed, and M. Dorigo, 'Gesturing at subswarms: Towards direct human control of robot swarms', in *Towards autonomous robotic systems*, A. Natraj, S. Cameron, C. Melhuish, and M. Witkowski, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 390–403.

[39] L. H. Kim, D. S. Drew, V. Domova, and S. Follmer, 'User-defined Swarm Robot Control', in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, in CHI '20. Honolulu, HI, USA: Association for Computing Machinery, Apr. 2020, pp. 1–13. doi: 10.1145/3313831.3376814.

[40] V. Serpiva, E. Karmanova, A. Fedoseev, S. Perminov, and D. Tsetserukou, 'SwarmPaint: Human-Swarm Interaction for Trajectory Generation and Formation Control by DNN-based Gesture Interface', in *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, Jun. 2021, pp. 1055–1062. doi: 10.1109/ICUAS51884.2021.9476795.

[41] S. Nunnally, P. Walker, M. Lewis, N. Chakraborty, and K. Sycara, 'Using Haptic Feedback in Human Robotic Swarms Interaction', *Proc. Hum. Factors Ergon. Soc. Annu. Meet.*, vol. 57, no. 1, pp. 1047–1051, Sep. 2013, doi: 10.1177/1541931213571233.

[42] E. Tsykunov, L. Labazanova, A. Tleugazy, and D. Tsetserukou, 'SwarmTouch: Tactile Interaction of Human with Impedance Controlled Swarm of Nano-Quadrotors',

in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 4204–4209. doi: 10.1109/IROS.2018.8594424.

[43] S. S. Abdi and D. A. Paley, 'Safe Operations of an Aerial Swarm via a Cobot Human Swarm Interface', in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, May 2023, pp. 1701–1707. doi: 10.1109/ICRA48891.2023.10161343.

[44] M. Le Goc, L. H. Kim, A. Parsaei, J.-D. Fekete, P. Dragicevic, and S. Follmer, 'Zooids: Building Blocks for Swarm User Interfaces', in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, in UIST '16. New York, NY, USA: ACM, 2016, pp. 97–109. doi: 10.1145/2984511.2984547.

[45] L. H. Kim and S. Follmer, 'UbiSwarm: Ubiquitous Robotic Interfaces and Investigation of Abstract Motion as a Display', *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 1, no. 3, p. 66:1-66:20, Sep. 2017, doi: 10.1145/3130931.

[46] S. Pourmehr, V. M. Monajjemi, R. Vaughan, and G. Mori, '"You two! Take off!": Creating, modifying and commanding groups of robots using face engagement and indirect speech in voice commands', in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov. 2013, pp. 137–142. doi: 10.1109/IROS.2013.6696344.

[47] M. Chandarana, E. L. Meszaros, A. Trujillo, and B. Danette Allen, 'Natural Language Based Multimodal Interface for UAV Mission Planning', *Proc. Hum. Factors Ergon. Soc. Annu. Meet.*, vol. 61, no. 1, pp. 68–72, Sep. 2017, doi: 10.1177/1541931213601483.

[48] J. D. Hasbach and M. Bennewitz, 'The design of self-organizing human–swarm intelligence', *Adapt. Behav.*, vol. 30, no. 4, pp. 361–386, Aug. 2022, doi: 10.1177/10597123211017550.

[49] L. Zhang and R. Vaughan, 'Optimal robot selection by gaze direction in multi-human multi-robot interaction', in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2016, pp. 5077–5083. doi: 10.1109/IROS.2016.7759745.

[50] G. K. Karavas, D. T. Larsson, and P. Artemiadis, 'A hybrid BMI for control of robotic swarms: Preliminary results', in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 5065–5075. doi: 10.1109/IROS.2017.8206390.

[51] "TIA Portal GUI " Accessed: Jul. 22, 2024. [Online]. Available: https://assets.new.siemens.com/siemens/assets/api/uuid:659ff273-d199-4b0e-

9eab-da7839c53ddf/quality:high/width:2048/crop:0,321:0,156:0,67:0,754/tia-portal-zukunftssicherheit.png

[52] "Totally Integrated Automation Portal," siemens.com Global Website. Accessed: Jul. 22, 2024. [Online]. Available: https://www.siemens.com/global/en/products/automation/industry-software/automation-software/tia-portal-variant.html

[53] "Simulink GUI " Accessed: Jul. 22, 2024. [Online]. Available: https://ctms.engin.umich.edu/CTMS/Content/Suspension/Simulink/Control/figures/susp_model.png

[54] "Simulink - Simulation and Model-Based Design." Accessed: Jul. 22, 2024. [Online]. Available: https://www.mathworks.com/products/simulink.html

[55] "Stateflow GUI" Accessed: Jul. 22, 2024. [Online]. Available: https://upload.wikimedia.org/wikipedia/en/4/4e/Automatic_Transmission_Logic.png

[56] "Stateflow." Accessed: Jul. 22, 2024. [Online]. Available: https://www.mathworks.com/products/stateflow.html

[57] "Mendix GUI" Accessed: Jul. 22, 2024. [Online]. Available: https://gdm-catalog-fmapi-prod.imgix.net/ProductScreenshot/deb922d9-ab4c-46a1-a6ac-0741b50d0c69.png?auto=format&q=50

[58] "Low-code Application Development Platform," Mendix. Accessed: Jul. 22, 2024. [Online]. Available: https://www.mendix.com/

[59] "LabVIEW GUI" Accessed: Jul. 22, 2024. [Online]. Available: https://ltm-data.de/wp-content/uploads/2017/02/LabVIEW.png

[60] "Was ist NI LabVIEW? Grafische Programmierung für Prüf- und Messtechnik." Accessed: Jul. 22, 2024. [Online]. Available: https://www.ni.com/de/shop/labview.html

[61] "Microsoft Power Apps GUI" Accessed: Jul. 22, 2024. [Online]. Available: https://miro.medium.com/v2/resize:fit:1400/0*ur0PdihI8OEewSd0.png

[62] "Microsoft Power Apps – Erstellen von Apps mit KI | Microsoft." Accessed: Jul. 22, 2024. [Online]. Available: https://www.microsoft.com/de-de/power-platform/products/power-apps

[63] "SAP Build GUI " Accessed: Jul. 22, 2024. [Online]. Available: https://scr3.golem.de/screenshots/2305/SAPBuildApps/thumb620/001_vob_SAP.png

[64] "Low-Code App Development and Automation Solutions | SAP Build," SAP. Accessed: Jul. 22, 2024. [Online]. Available: https://www.sap.com/products/technology-platform/low-code.html

[65] "Unity Visual Scripting GUI" Accessed: Jul. 22, 2024. [Online]. Available: https://i.sstatic.net/9uQH7.png

[66] "Unity Visual Scripting," Unity. Accessed: Jul. 22, 2024. [Online]. Available: https://unity.com/features/unity-visual-scripting

[67] "Node-RED." Accessed: Jul. 22, 2024. [Online]. Available: https://nodered.org/

[68] "JavaScript libraries and components for web development." Accessed: Jul. 22, 2024. [Online]. Available: https://www.totaljs.com/

[69] "Node-Based UIs in React – React Flow." Accessed: Jul. 22, 2024. [Online]. Available: https://reactflow.dev/

[70] "Home," Apache Airflow. Accessed: Jul. 22, 2024. [Online]. Available: https://airflow.apache.org/

[71] "Node-RED GUI." Accessed: Jul. 22, 2024. [Online]. Available: https://user-images.githubusercontent.com/4663918/63022233-76304400-be70-11e9-8516-cab988df6b1e.png

[72] "total.js GUI." Accessed: Jul. 22, 2024. [Online]. Available: https://www.totaljs.com/download/IYxBFL1cQ61f.jpg

[73] "ReactFlow GUI." Accessed: Jul. 22, 2024. [Online]. Available: https://miro.medium.com/v2/resize:fit:1358/0*SAKT7uFAPvdfWZFO

[74] “Apache Airflow GUI.” Accessed: Jul. 22, 2024. [Online]. Available: https://airflow.apache.org/docs/apache-airflow/2.2.4/_images/graph.png

[75] M. Rubenstein, C. Ahler and R. Nagpal, "Kilobot: A low cost scalable robot system for collective behaviors," *2012 IEEE International Conference on Robotics and Automation*, Saint Paul, MN, USA, 2012, pp. 3293-3298, doi: 10.1109/ICRA.2012.6224638.

[76] A. Reina, A. J. Cope, E. Nikolaidis, J. A. R. Marshall and C. Sabo, "ARK: Augmented Reality for Kilobots," in IEEE Robotics and Automation Letters, vol. 2, no. 3, pp. 1755-1761, July 2017, doi: 10.1109/LRA.2017.2700059.

[77] C. Pinciroli, M. S. Talamali, A. Reina, J. A. Marshall, and V. Trianni, “Simulating kilobots within argos: Models and experimental validation,” Lecture Notes in Computer Science, pp. 176–187, 2018. doi:10.1007/978-3-030-00533-7_14

[78] C.Pinciroli and G. Beltrame, “Buzz: An extensible programming language for heterogeneous swarm robotics,” *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 3794-3800.

[79] J. Shin, R. Siegwart and S. Magnenat, “Visual programming language for Thymio II robot,” *In Conference on interaction design and children (idc'14),* 2014, ETH Zürich.

[80] E. Coronado, F. Mastrogiovanni, B. Indurkhya and G. Venture, “Visual programming environments for end-user development of intelligent and social robots, a systematic review,” Journal of Computer Languages, 58, p.100970. 2020.

[81] L.P. Pinheiro, Y.K. Lopes, A.B. Leal and R.S.U.R. Junior, “Nadzoru: A software tool for supervisory control of discrete event systems,” IFAC-PapersOnLine, 48(7), pp.182-187. 2015.

[82] Y. K. Lopes, S. M. Trenkwalder, A. B. Leal, T. J. Dodd, and R. Groß, ‘Supervisory control theory applied to swarm robotics’, Swarm Intell., vol. 10, no. 1, Art. no. 1, Mar. 2016, doi: 10.1007/s11721-016-0119-0.

[83] G. Miyauchi, Y. K. Lopes, and R. Groß, ‘Multi-Operator Control of Connectivity-Preserving Robot Swarms Using Supervisory Control Theory’, in 2022 International Conference on Robotics and Automation (ICRA), May 2022, pp. 6889–6895. doi: 10.1109/ICRA46639.2022.9812242.

Glossary

ARK: Augmented Reality system

AGV: Automated Guided Vehicle

API: Application Programming Interface

CI/CD: Continuous Integration/Continuous Delivery (or Deployment)

CPU: Central Processing Unit

DDA: Data Distribution Agent

EHS: Environment, Health, and Safety

FaaS: Function as a Service

GPU: Graphics Processing Unit

gRPC: gRPC Remote Procedure Call

HSI Human-Swarm Interaction

I/O: Input/Output

KAR: Key Architectural Requirements

KPI: Key Performance Indicator

LCM: Life Cycle Management

LoA Level of Autonomy

MES: Manufacturing Execution System

MQTT: Message Queuing Telemetry Transport

PoC: Proof of Concept

REST: Representational State Transfer

RCA: Root Cause Analysis

RTOS: Real-Time Operating System

SMS: Short Message Service

SCT: Supervisory Control Theory

UI: User Interface

List of Figures