Open-Digital-Industrial and Networking pilot lines using modular components for scalable production

Grant Agreement No	:	**101017141**
Project Acronym	:	**ODIN**
Project Start Date	:	**1st January, 2021**
Consortium	:	**UNIVERSITY OF PATRAS – LABORATORY FOR MANUFACTURING SYSTEMS AND AUTOMATION** **FUNDACION TECNALIA RESEARCH & INNOVATION** **KUNGSLIGA TEKNISKA HOEGSKOLAN** **TAMPEREEN KORKEAKOULUSAATIO SR** **COMAU SPA** **PILZ INDUSTRIEELEKTRONIK S. L.** **ROBOCEPTION GMBH** **VISUAL COMPONENTS OY** **INTRASOFT INTERNATIONAL SA** **GRUPO S21SEC GESTIÓN, S.A.** **FUNDACION AIC AUTOMOTIVE INTELLIGENCE CENTER FUNDAZIOA** **DGH ROBOTICA, AUTOMATIZACION Y MANTENIMIENTO INDUSTRIAL SA** **PSA AUTOMOBILES S.A.** **AEROTECNIC COMPOSITES SL. U.** **WHIRLPOOL EMEA SPA** **WHIRLPOOL MANAGEMENT EMEA SRL**

Title	:	ODIN Networked Component final version
Reference	:	D4.3
Availability	:	Public
Date	:	30/12/2023
Author/s	:	INTRA, S21SEC
Circulation	:	EU, Consortium

Summary:

The purpose of this document is to present the design and final prototype of:
a) OpenFlow communication and integration architecture and
b) Active DT Protection Framework and DT Intelligent Threat Analysis Toolkit

ODIN project has received funding from the European Union's Horizon Europe research and innovation programme under Grant Agreement No 101017141.

Co-funded by the European Union

Table of Contents

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

This document describes the ODIN Networked Component final version, which is the result of tasks T4.1 “Reference integration and communication architecture for reconfigurable production” and T4.2 “Cybersecurity and data processing in autonomous production environments”. The “ODIN Networked Component” is comprised of the following two software modules.

a) The OpenFlow module, which is responsible to integrate, orchestrate, manage and coordinate production resources to execute manufacturing schedules, and

b) The Cybersecurity module, which is responsible to provide detection and response capabilities on the deployed Networked Component.

The final prototype of the OpenFlow module offers integration, orchestration and management/configuration functionalities. The initial prototype of OpenFlow module includes the following functionalities:

- Orchestration of Modules and Resources,
- Emulated execution of a production schedule,
- Simulated execution of a production plan in a 3D virtual environment,
- Recovery strategies and system level reaction on different types of events, such as shopfloor events, execution failure events, safety violation events, security events.
- Control, Monitor Task and Action Execution Flow,
- Monitoring of Network Software Modules Status,
- Controlling of OpenFlow Execution Flow,
- Request execution task replanning,
- Validation of Open Schedules,
- OpenFlow Knowledge Repository,
- Information Exchange with ERP systems,
- UI offering control, monitoring, and views of OpenFlow functionalities to end user.

The final prototype for ODIN Cybersecurity solution includes both the process and methodology for ODIN threat modelling and the cybersecurity toolkit for incident detection and response.

The threat modelling for ODIN Cybersecurity follows threat modelling paradigms such as the MaGMa [18] and MITRE ATT&CK [19].

The Cybersecurity module includes both incident detection and incident Response functionalities.

The final prototype of the ODIN Networked Component offers essential and core integration, orchestration and cybersecurity features in the ODIN system. The ODIN Networked Component has already been used verified in the context of the ODIN Pilot cases development and preliminary ODIN Pilot cases and will be used and will be used to integrate, orchestrate and secure the developments in all the ODIN Pilot Cases.

LEGAL DISCLAIMER

The ODIN project is co-funded by the European Union’s Horizon 2020 research and innovation programme under the Grant Agreement No 101017141. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

1. INTRODUCTION

The transition from mass production to mass customization has indicated the need of deploying flexible manufacturing systems that are able to handle multiple product variants [1].

In an increasingly fast-paced and ever-changing global market, advanced collaborative and flexible manufacturing is a strong competitive advantage. The ODIN project aims to bring a flexible, modular, and scalable solution for the manufacturing industry. The ODIN project proposes a solution that includes the following four components:

- The ODIN Digital Component
- The ODIN Open Component
- The ODIN Industrial Component
- The ODIN Networked Component

This deliverable, namely D4.3 "ODIN Networked Component final version", describes the concept, features design and implementation of the final version ODIN Networked Component. The design and development of the Networked Component followed an agile approach and was performed in tandem with the design and development of the other ODIN modules and in parallel with the ODIN Pilot Cases developments.

The ODIN Networked Component is comprised of the following modules.

- The OpenFlow module whose final prototype is presented in section 2 and
- the Cybersecurity module whose final prototype is presented in section 3.

The final prototype of the OpenFlow module provides many functionalities related to the integration, orchestration, monitoring and control of the resources and software modules in the shopfloor. In addition, it provides connection to external, systems such as ERP. It has been successfully used in the integrated developments for the pilot cases. In particular, it is used in White Goods, Automotive and Aeronautic Pilot cases.

The final prototype follows the ODIN Reference architecture described in D1.4. An earlier version OpenFlow module has been described together with details about the module design in D4.1 "ODIN Networked Component initial prototype". OpenFlow architecture offers centralized coordination, orchestration, and an integration platform for the software system [2].

The final prototype for ODIN Cybersecurity solution provides cybersecurity services for the ODIN Pilot Cases. The ODIN Cybersecurity solution features both a targeted threat modelling and an effective cybersecurity toolkit for incident detection and response based on paradigms such as the MaGMa and MITRE ATT&CK.

2. OPENFLOW

2.1. Introduction

This section presents the final prototype OpenFlow module which is one of the two modules of the ODIN Networked component. The final prototype of the OpenFlow module is the newest and final version of the OpenFlow module, whose initial prototype has been presented in the ODIN deliverable D4.1 "ODIN Networked Component initial prototype". The development of the final prototype OpenFlow module follows the OpenFlow Architecture and specifications of the ODIN Architecture specifications of D1.4.

OpenFlow architecture offers centralized coordination, orchestration, and an integration platform for the software system [2]. OpenFlow interoperates with and manages other ODIN modules. The latest version of the OpenFlow functionalities are presented in section 2.3. These features have been presented in D1.4, were implemented through the development phase of WP4 and were also targeted by the initial OpenFlow prototype. Table 1 summarizes key OpenFlow features and provides an overview of the different feature status in the OpenFlow initial prototype and the final prototype of the OpenFlow module.

Table 1: OpenFlow features status

OpenFlow Features	Initial OpenFlow Prototype	Final OpenFlow Prototype
Orchestrate Modules and Resources	Initial prototype	Completed.
Emulation	Only Actions	Completed. Includes Actions, services and events.
Simulation	Initial prototype supporting validation scenarios	Completed and validated with ODIN Digital Twin.
React on Shopfloor Events	Initial prototype supporting validation scenarios	Completed and validated in Whitegoods and Automotive Pilots
React on Safety Events	Initial prototype supporting validation scenarios	Completed and validated in Whitegoods and Automotive Pilots
React on Security Events	Initial prototype supporting validation scenarios	Completed and validated in Whitegoods and Automotive Pilots
Control & Monitor Task and Action Execution Flow	Initial prototype supporting validation scenarios	Final prototype that can support industrial scale scenarios
Monitor Networked Software Modules Status	Initial prototype supporting validation scenarios	Final prototype that can support industrial scale scenarios, visualization using interactive diagrams with motion
Control OpenFlow Execution Flow	Initial prototype supporting validation scenarios	Final prototype that can support full scale scenarios
Request Replanning	Only in emulation	Can work in emulation, simulation and execution.
Validate Open Schedules	Partial, targeting initial prototype	Final prototype that can support industrial scale scenarios
OpenFlow Knowledge Repository	Supports Initial Prototype Functionalities	Completed, supports all functionalities

OpenFlow Features	Initial OpenFlow Prototype	Final OpenFlow Prototype
Information Exchange with ERP systems	Initial Prototype Connection Established	Included and validated, supports production plan orders and mobile resources' location tracking. ERP emulation.
User Interface	Supports Initial Prototype Functionalities	Completed, supports all functionalities

2.2. Final Prototype

The final prototype of the OpenFlow module, aims to streamline the Human Robot collaborative software system integration. Being modular, responsive, scalable, flexible and extensible allows the system to be quickly deployed in different manufacturing environments. The OpenFlow module can work with any size of manufacturing system and can also be extended with appropriate new functionalities when needed. Task planning, perception, actuation, AR application and robot control modules integration with OpenFlow makes the OpenFlow architecture able to coordinate the execution of all required tasks.

The modularity, scalability and flexibility aspects are important as the transition from mass production to mass customization requires the design and operation of systems able to handle the increasing product variety [1].

2.3. Functionalities Overview

The OpenFlow orchestrator main functionality is the execution of an OpenFlow production schedule, which defines its interactions with other modules.

The OpenFlow production execution schedule models the tasks and actions that need to be executed by the production resources to implement a production order. The production schedule also models the system level reactions and recovery options in anticipation of disruptive events such as safety, security or different type of failures, such as equipment or network failure.

The production schedule model is comprised of a directed graph of actions, together with meta-information that group the actions into tasks and define how to dispatch the necessary commands for each action.

Each action is a node in the graph and connects to other actions with directed vertices. Every vertex has a source action, a destination action and a vertex unique name. Vertices among the same source and the same destination actions are uniquely named.

Actions in the OpenFlow architecture model atomic, indivisible processes or operations. These actions can represent human tasks, robot or machinery equipment operations, or computing processes. Each action is controlled by a specific action actor.

OpenFlow supports different technologies, including the Robot Operating System (ROS), that is a set of software libraries and tools used to build robot applications.

All active actions are executed in parallel with no assumptions regarding the relative order of the action executions. The actor model is a concurrent computation model, that introduces actors as a primitive of concurrent computation. In a nutshell actors exchange information only by immutable messages.

Tasks are defined as groups of actions; each task has a single first action and a single last action. When requested, OpenFlow starts a schedule by automatically activating all the entry-point actions and stops a schedule when the first stop (terminating) action has been reached. Once an action reaches a terminal state the action actor sends start signals to the appropriate actions that they could start. Selecting suitable next actions as well as the preconditions that need to be fulfilled before an action can start after receiving a start signal depends on the configurations in the OpenFlow manufacturing schedule

and the monitored execution status of the action. The OpenFlow orchestrator selects the proper actions to start depending on the execution outcome of an action as well as the overall execution context.

2.3.1. Login

The OpenFlow final prototype supports multiple accounts, providing better granularity on the management of different accounts than the initial OpenFlow prototype. The functionality is transparent to the user, who only needs to enter the login credentials in the Login Page.

Figure 1: OpenFlow UI: Login Page

Login Page offers a user-friendly, simple login functionality to OpenFlow UI. OpenFlow provides role management. Each user has access only to the information of the company related company. After the login, the user is redirected to the landing page.

2.3.2. Schedules View

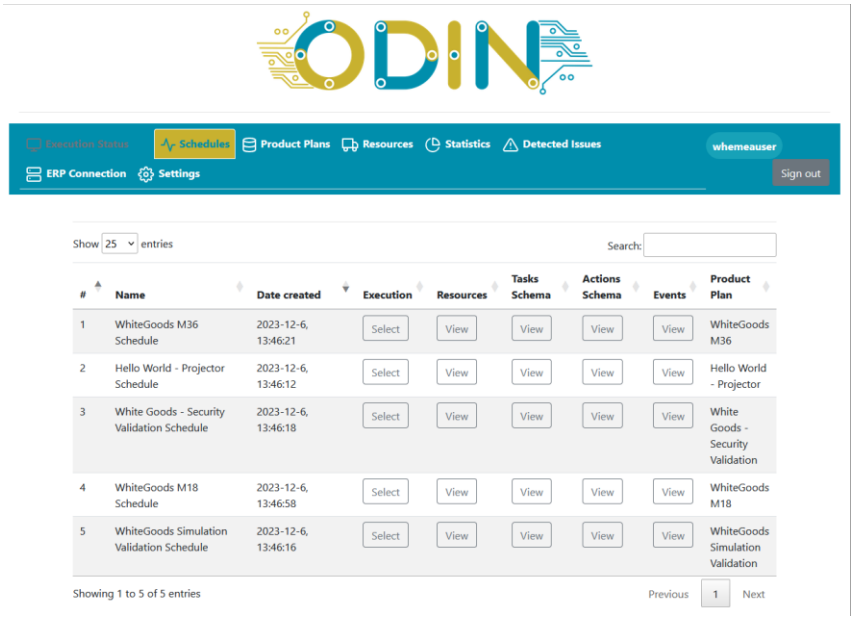

Figure 2: OpenFlow UI: Schedules Overview

The Schedules Overview screen of the OpenFlow final prototype software module displays useful information to users, providing a high level of overview as well as detailed information. The Schedules Overview screen offers access to visual representations of the required Resources of each Product Plan and information about these Resources, Schedule's Tasks and Actions as well as the core functionality to select a Schedule for execution by clicking the related "Select" button in the Execution column as shown in Figure 2.
OpenFlow allows the user to see different information for each schedule through a set of different action buttons. In particular, the OpenFlow provides the following action buttons for each schedule.

- Schedule Execution.
  - When this button is pressed, the OpenFlow sends the selected schedule in the Execution Status View, that is presented in section 2.3.3.
- View Resources.
  - This button shows information about the resources that are used in the particular schedule.
- View Tasks Level Schedule Diagram.
  - This button shows a task level diagram for the particular schedule using the task level schedule visualization functionality of OpenFlow. An example task level schedule visualization for the White goods pilot case can be seen in Figure 3.

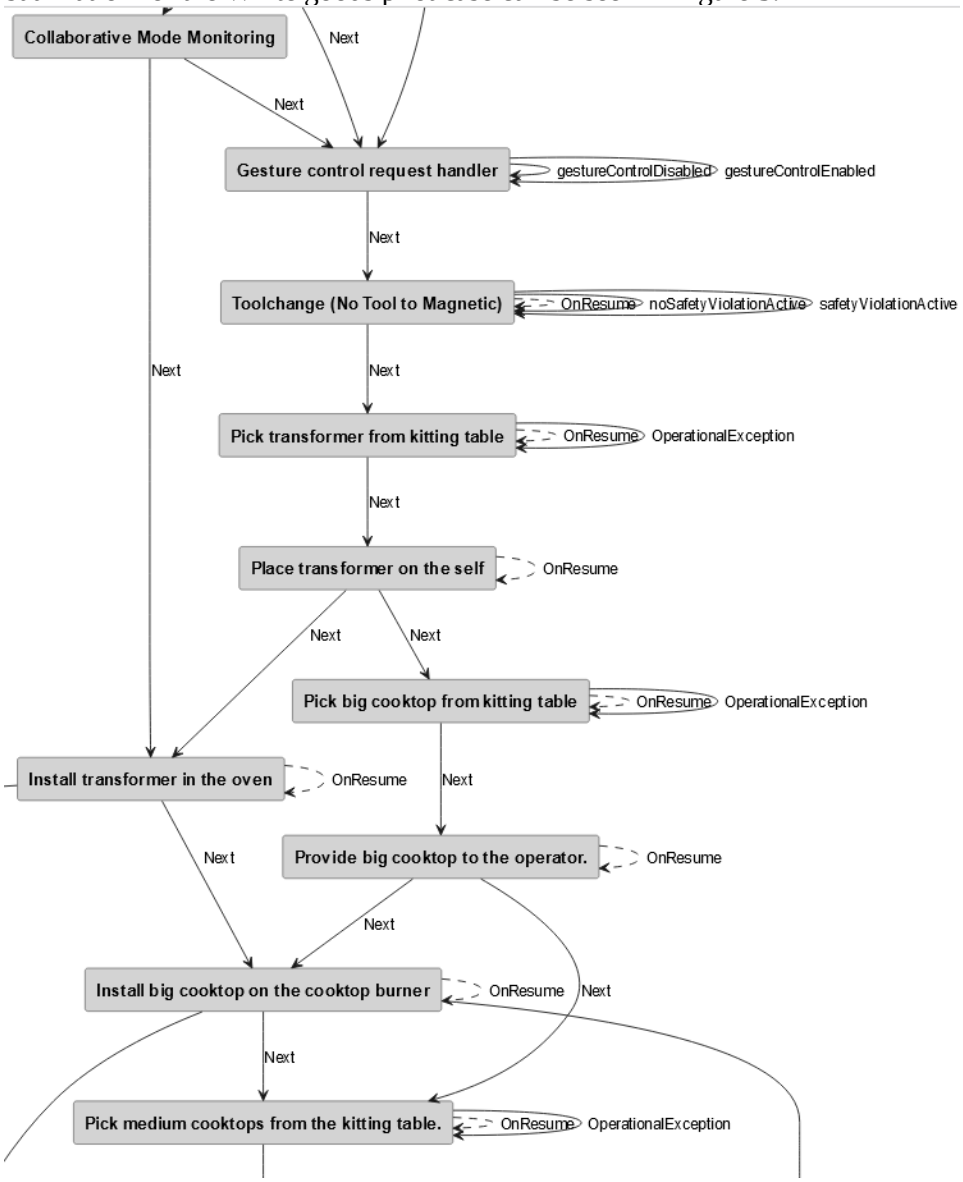

Figure 3: Task Level Schedule Diagram - White Goods Pilot Case

- View Actions Level Schedule Diagram

  This button shows an actions level diagram for the particular schedule using the action level schedule visualization functionality of OpenFlow. An example action level schedule visualization for the White goods pilot case can be seen in Figure 4.

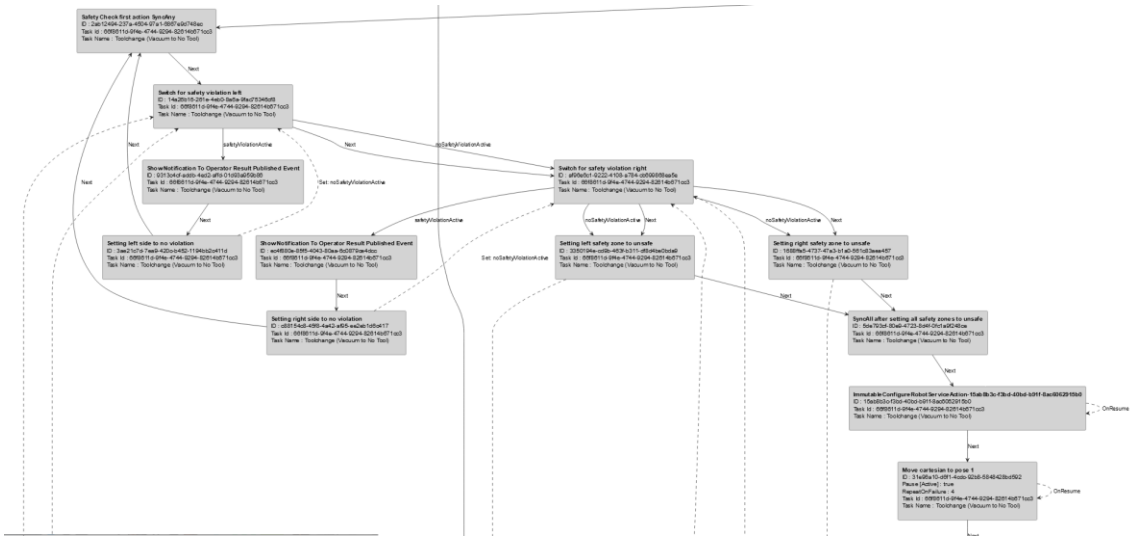

Figure 4: Actions Level Schedule Diagram – White Goods Pilot Case

- View Events

  The OpenFlow Schedules view can show all supported events for each schedule. During the execution of a production schedule the OpenFlow can monitor and log the triggering of these events in dedicated event logs. The OpenFlow can show the events for a specific schedule after clicking the Events button on each schedule. Information for each event such as Name, Event Type and the predefined schedule execution actions that handle the event tracking in real-time can also be shown. A screenshot of the events view is available below.

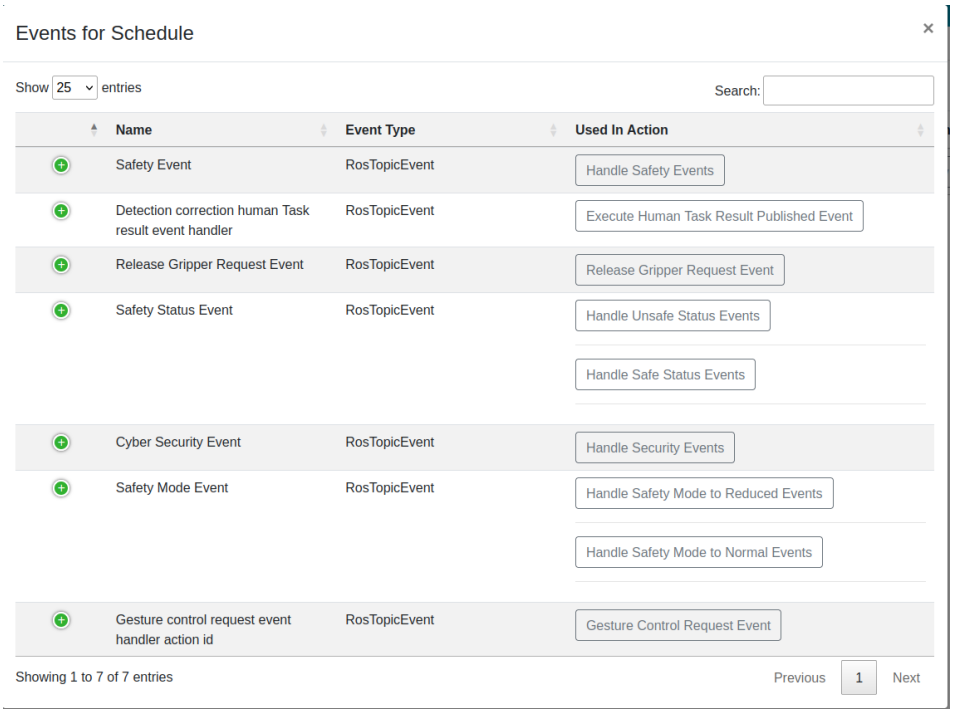

Figure 5: Schedule – Events Overview

The OpenFlow can provide more detailed information for each event depending on the Event Type. For instance, the user can expand events implemented as ROS Topics to see their definition. More specifically, the OpenFlow can show to the user the exact ROS definition file, the event ID, the message type, the Graph Name used in the current ROS network and the ROS MD5 as shown in Figure 6. This figure shows information about the safety related events. The information presented in this view is useful to support integration testing and onboarding, as it allows quick comparisons between interface definitions. When one of the displayed events occurs, it is being logged in Execution Status view which will be described in section 2.3.3.

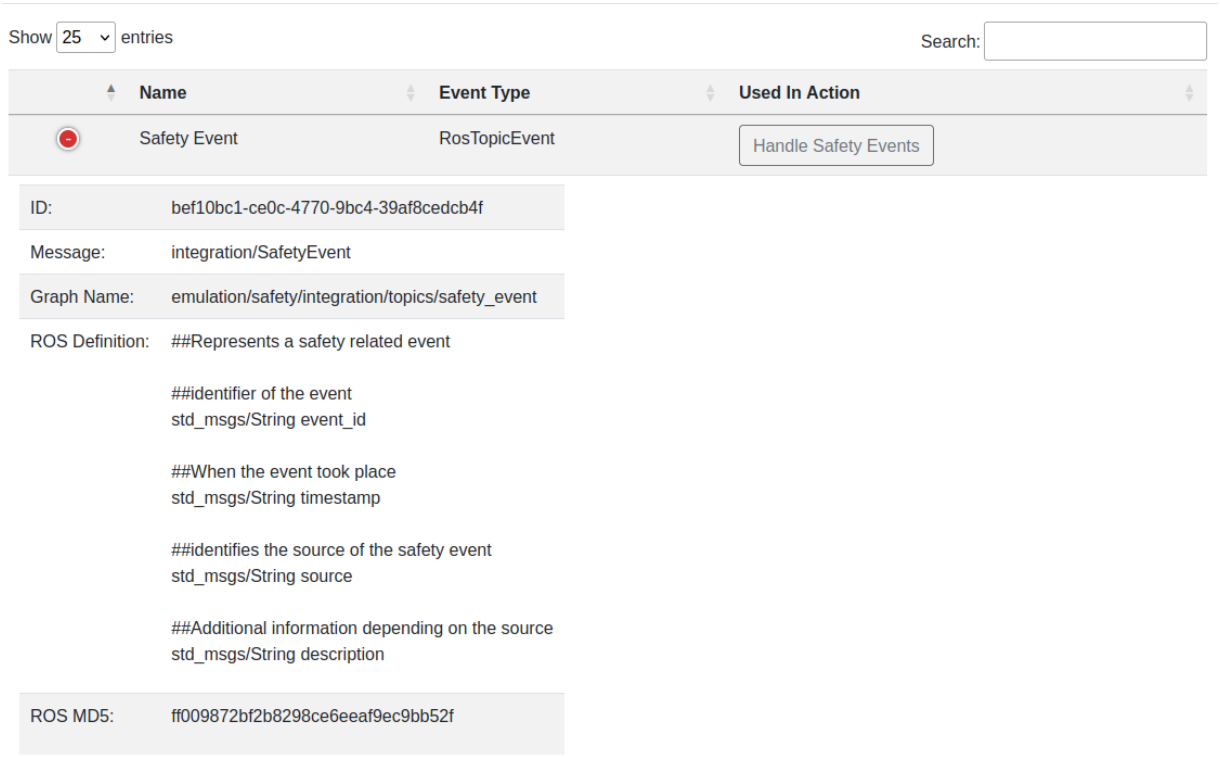

Figure 6: Schedule – Detailed Events view

2.3.3. Execution Status View

The OpenFlow UI Execution Status screen provides information about a running schedule and offers control options to the user, effectively allowing the user to control the production execution. The Execution Status View becomes visible, when schedule is selected. The user can select any schedule visible in the schedules view, (section 2.3.2) for execution and this will open the schedule in the Execution Status View. The OpenFlow schedule Execution View is shown in Figure 7.

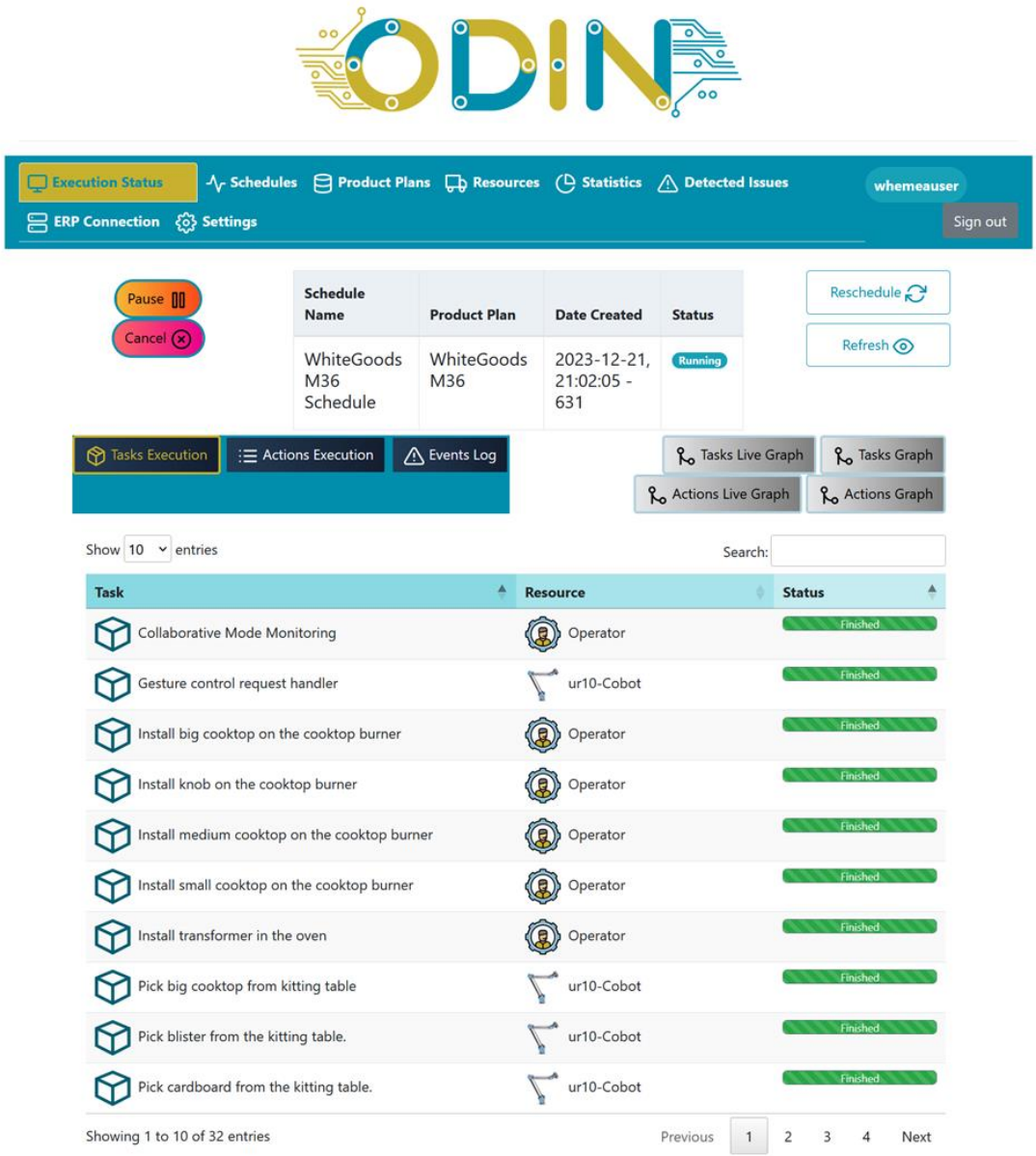

Figure 7: Open Flow UI: Execution Status View - Whitegoods Pilot Case

The OpenFlow UI Execution Status View enables the control of a Schedule and offers visualization of the execution in Task level or in the Action level which is more detailed.

- **Production Schedule Execution Control**

The final prototype of the OpenFlow software module enables the control of the execution at a system level. In particular the OpenFlow final prototype can start, pause, cancel and resume the execution.

The Start button starts the execution of the selected Schedule and enables the Pause-Resume & Cancel buttons and thus the related functionality.

While the Schedule is running, the OpenFlow allows the user to pause the current execution and resume it later on demand.

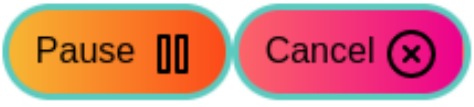

Figure 8: Options while Schedule is running

After clicking on the Pause button, the Pause button is replaced by a Resume button when the Schedule has been paused as shown in Figure 8. The Cancel Button cancels the Tasks & Actions that are active. The UI Controls are then updated to only allow the user to Start the Schedule from the first Task again.

Figure 9: Options while Schedule is paused

The final prototype of the OpenFlow module makes a clear distinction between the cancelling and pausing a running schedule. In particular, during the pausing functionality, the OpenFlow will make sure that the system will pause at the next stable, resumable opportunity, this may require some more time to complete, as the system will seek the next optimal point to pause. However, after pausing a schedule the OpenFlow final prototype can resume the operations seamlessly. On the other hand, the cancel button will invoke the cancelation functionality. The cancelation will try to stop the schedule as soon as possible, without considering the possibility to resume . In both cases, that is when the Schedule is paused or stopped, the OpenFlow final prototype allows the user to reschedule the remaining Tasks as a new Schedule ready for execution through the Reschedule button in Figure 7.

- **Schedule Status Visualization functionality**

The OpenFlow user interface provides real time monitoring on the status of Tasks & Actions as well as visualizing the Schedule in two different graph version, an animated interactive graph and a static graph. This information is shown in the Schedule Status View.

The status is shown in real-time in both the Task and Action level in the table (Figure 7).

To dispatch real-time status information for Tasks and Actions, OpenFlow utilizes a specific WebSocket server that offers bidirectional connection and supports multiple simultaneous connections so that external or internal software modules can receive the same update on each status dispatched. Clients receiving status updates from OpenFlow using this WebSocket connection include the OpenFlow UI – Execution Status view and Interactive live graphs which will be presented below. Figure 10, presents the information stream described above between OpenFlow and UI features.

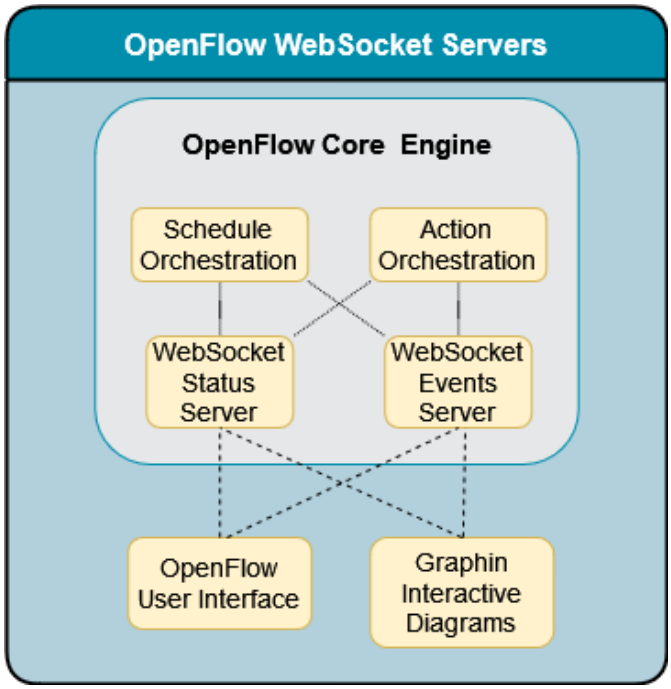

Figure 10: OpenFlow WebSocket information stream

- **Task Execution Status**

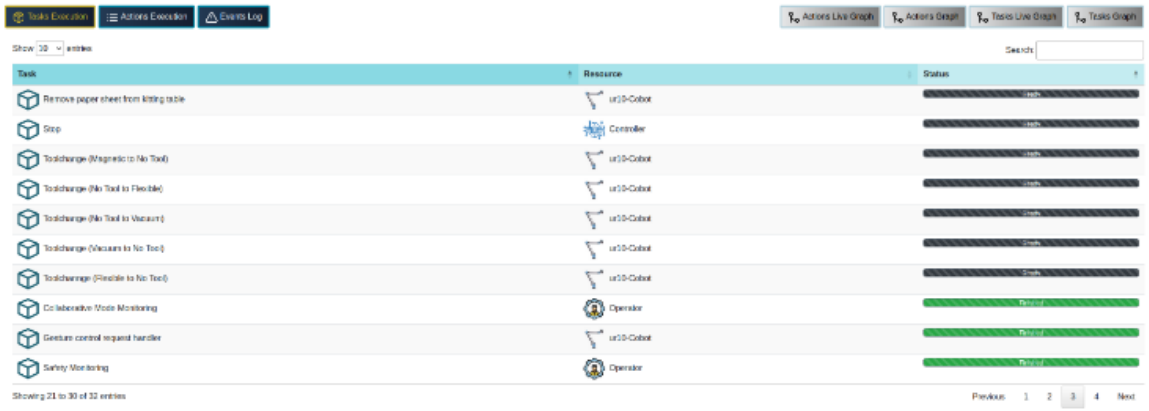

Figure 11: Tasks Execution Status – White Goods Pilot Case

While the Schedule is running, the status of the Tasks is constantly updated to show the actual execution status. The Tasks consist of many actions and offer a higher level of abstraction and observation. The screenshot in Figure 11 shows tasks in different status at the same screen.

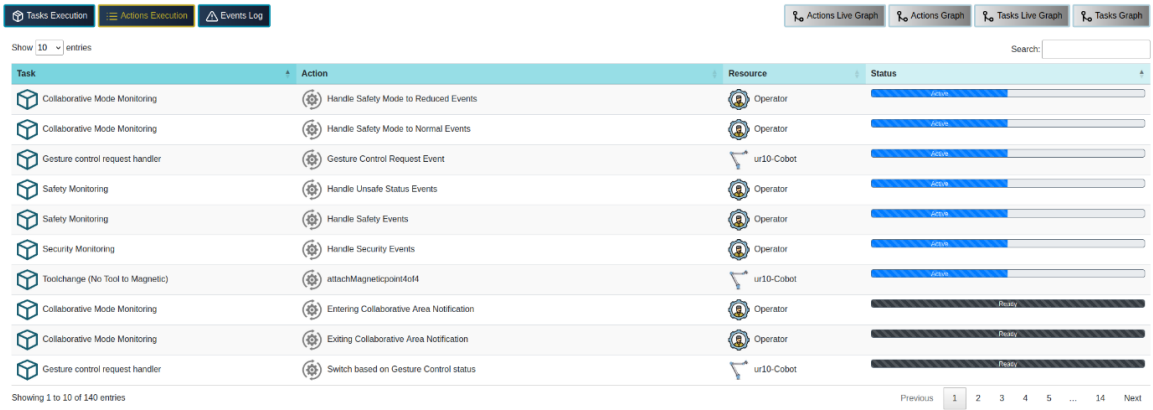

Figure 12: Actions Execution Status – White Goods Pilot Case

- **Action Execution Status**

Similar to the Task Execution Status, the Actions Execution tab panel displays actions names, their resources and Task and their status as shown in Figure 12. The Actions are atomic execution steps that are part of a higher-level Task abstraction.

- **Action & Task Information**

By clicking on each Action, the user can access a specific view presenting information data for the action clicked. Such data include the ID, the network resource utilized by this action, the task that it belongs too, if it can be repeated as well as the ID of next actions in the current selected . Additionally, ***Open Live Graph*** button opens the Actions Interactive Graph and zooms into the position of this action in the graph. Action Live Graph is presented below in this section.

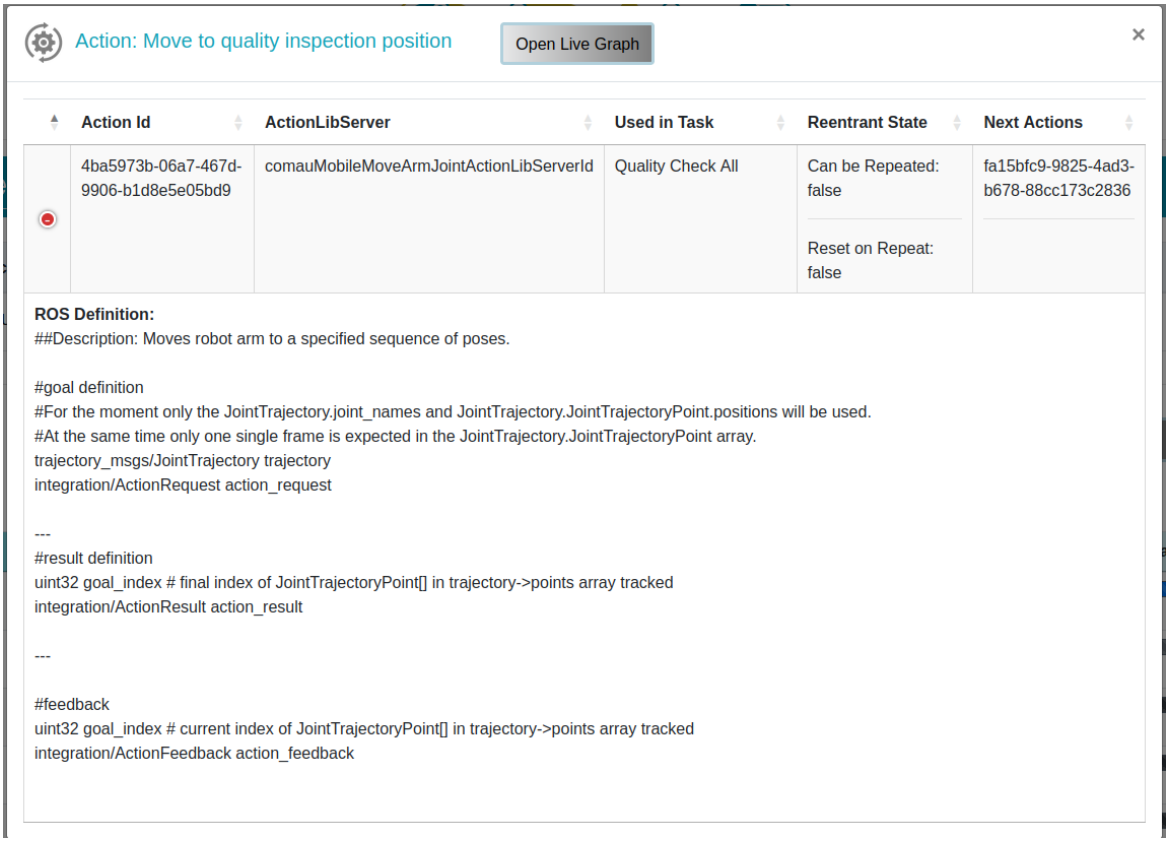

Figure 13: Action Information

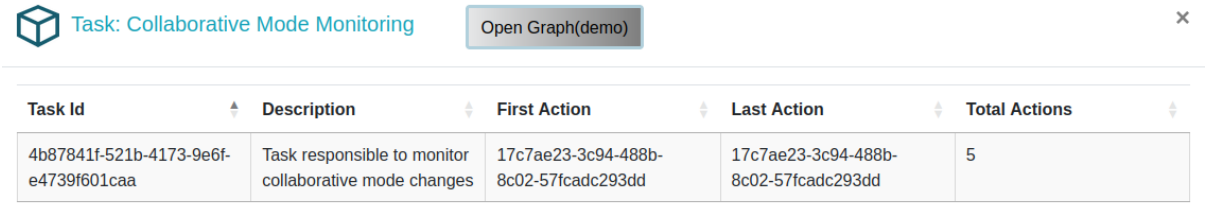

Figure 14: Task information

Similarly, clicking a Task opens a view to review Task information such as its ID, description, first and last actions as well as the total number of actions included in the Task as shown in Figure 14.

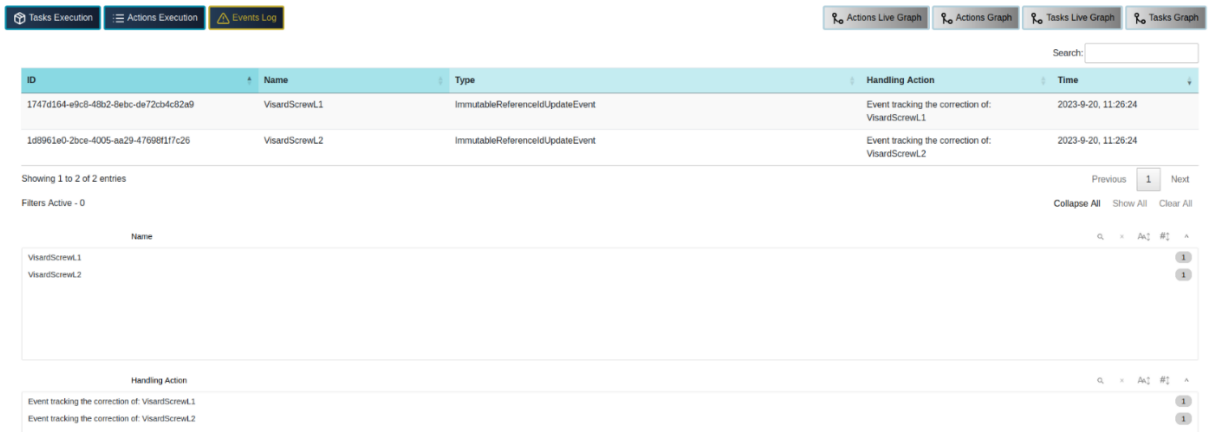

Figure 15: Event Logs

- **Event Logs**

The Event Logs View displays captured events during the execution of a specific schedule. The events are dispatched from the OpenFlow to the User Interface through a specific WebSocket for each running schedule, making it possible for external interfaces to subscribe and get updates from captured events. OpenFlow displays information about all logged events such as their name, type, logged time, and related invoked action. Additionally, filter tables offering selection to display only events of specific name or per handling actions in case of scenarios and production procedures in which many events are being logged. Figure 15 above presents the aforementioned functionalities through the Events Log button in the Execution Status view.

- **Actions & Tasks diagrams**

The OpenFlow UI can graphically depict the Execution Schedule in either Action or Task level granularity. Actions Graph and Tasks Graph option in Execution Status View offers visualization of the sequence of required actions and tasks respectively in order to execute a complete Schedule. A new diagram is generated for each case and a new page will load displaying the respective graph. Figure 16 shows the task diagram of the preliminary White Goods pilot case. Due to the size of the diagram a zoomed in part has been added.

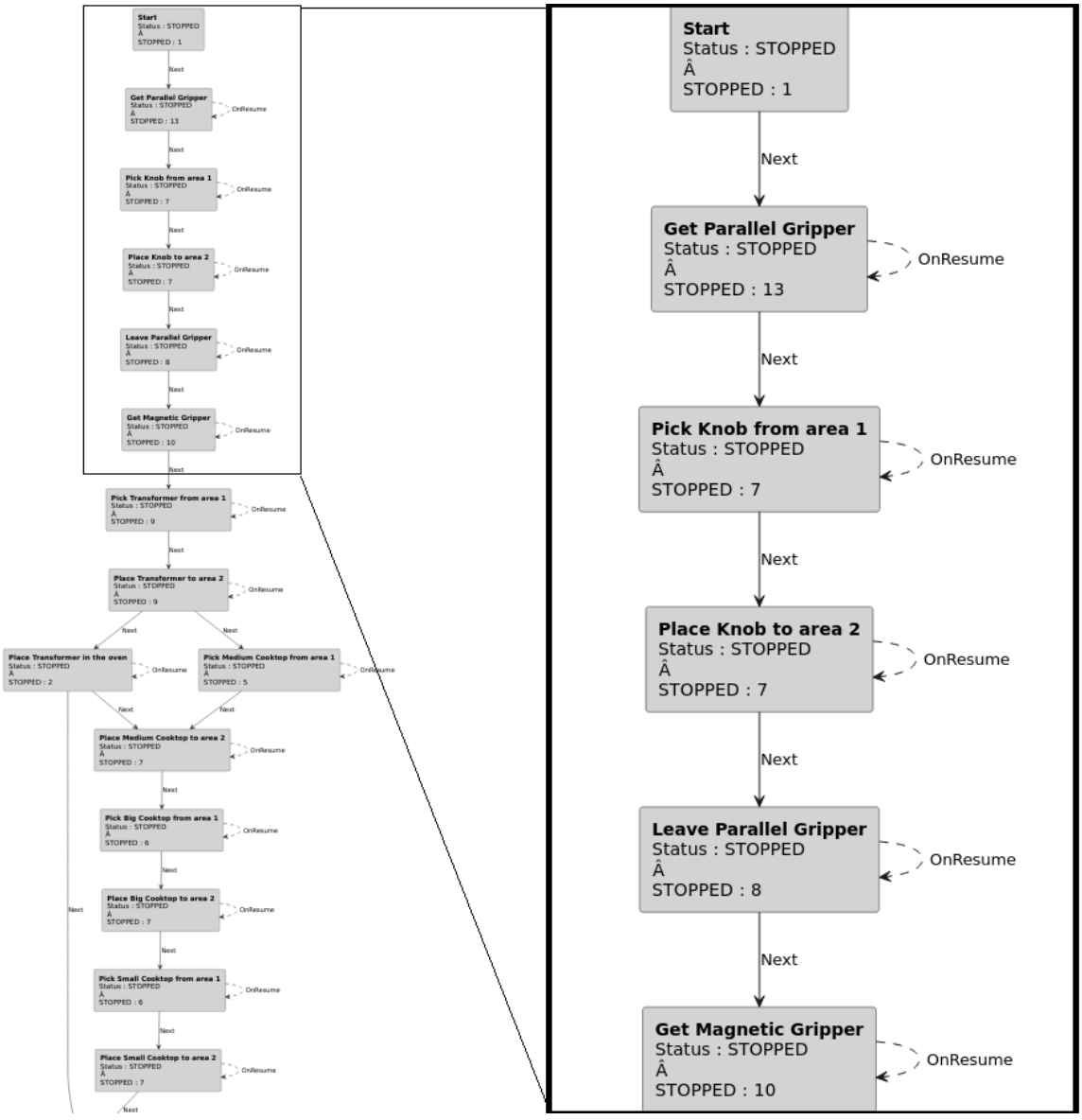

Figure 16: Tasks Level Schedule Visualization diagram – Whitegoods Pilot Case

- **Actions & Tasks Interactive Graphs**

Execution Status View offers access to interactive Graphs for the Tasks and Actions through ***Tasks Live Graph*** and ***Actions Live Graph*** buttons. Graphs are loaded to a new browser window, offering an independent standalone view into the structure and the order of the scheduled Tasks and their Actions. Graphs use the same WebSocket utilized be OpenFlow to dispatch the status of the running schedule and inform the User Interface about the current, completed and pending Tasks and Actions accordingly. Tasks Live Graph also displays the completion percent of each Task taking into account the comprised actions as shown in Figure 17. The Actions Live Graph displays the completion percent of each running Action as well as the Task list that is being updated when a Task is completed on the sidebar as shown in Figure 18.

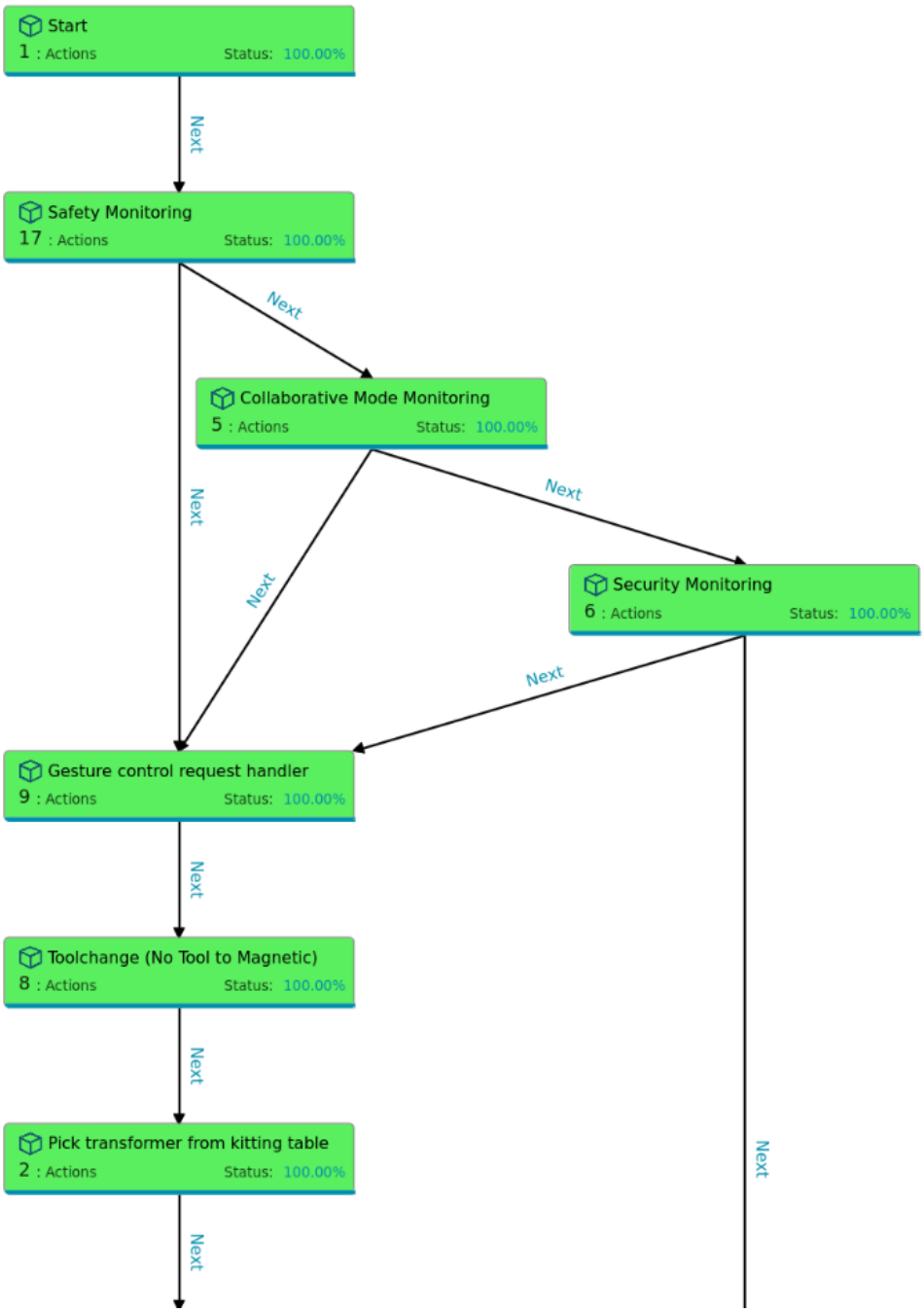

Figure 17: Tasks Live Graph

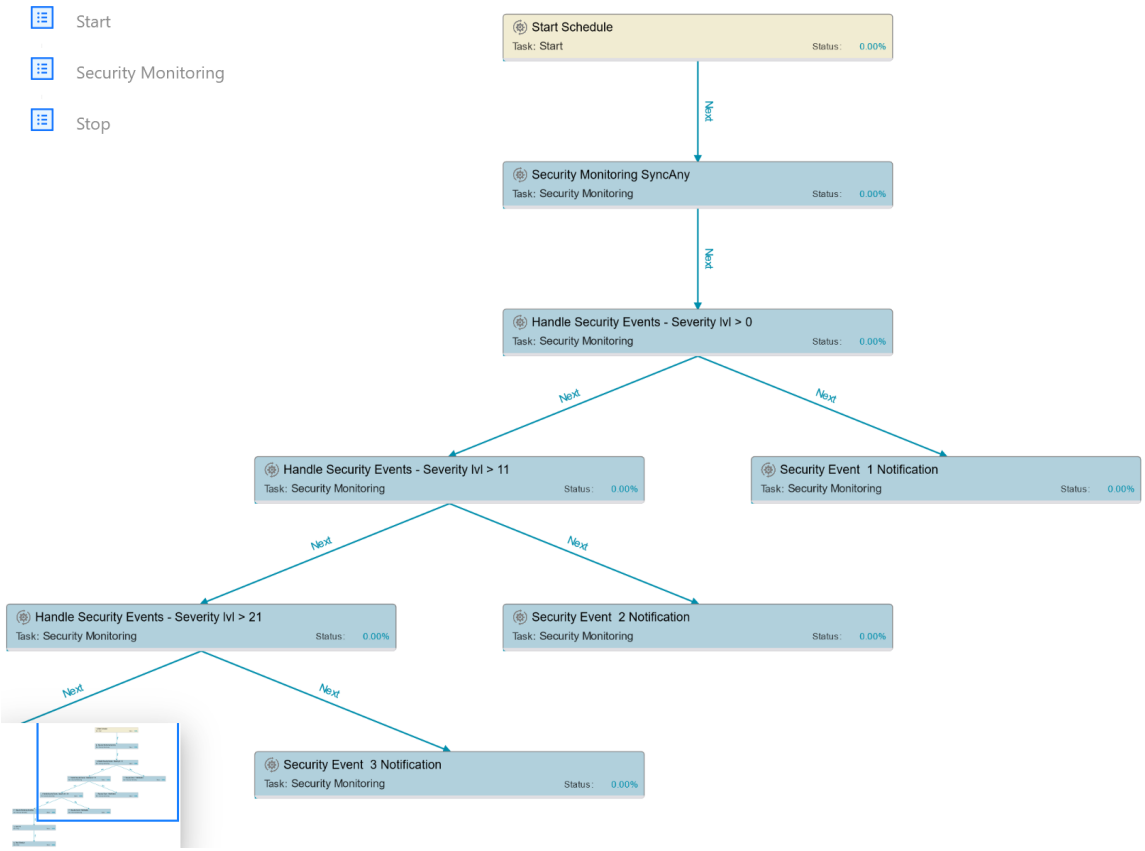

Figure 18: Actions Live Graph

Additionally, by hovering the mouse on each Task or Action displayed on the Graph, the User can see information about each Task or Action, such as ID, First Action and Last Action (Tasks Live Graph) as shown in Figure 19 and Figure 20 respectively.

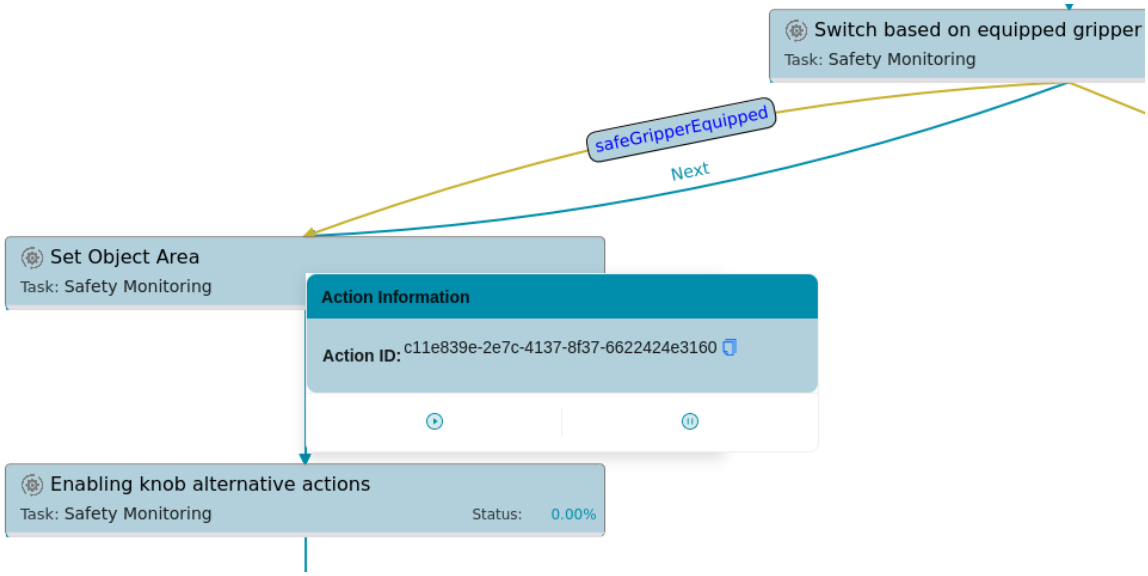

Figure 19: Actions Live Graph (action information)

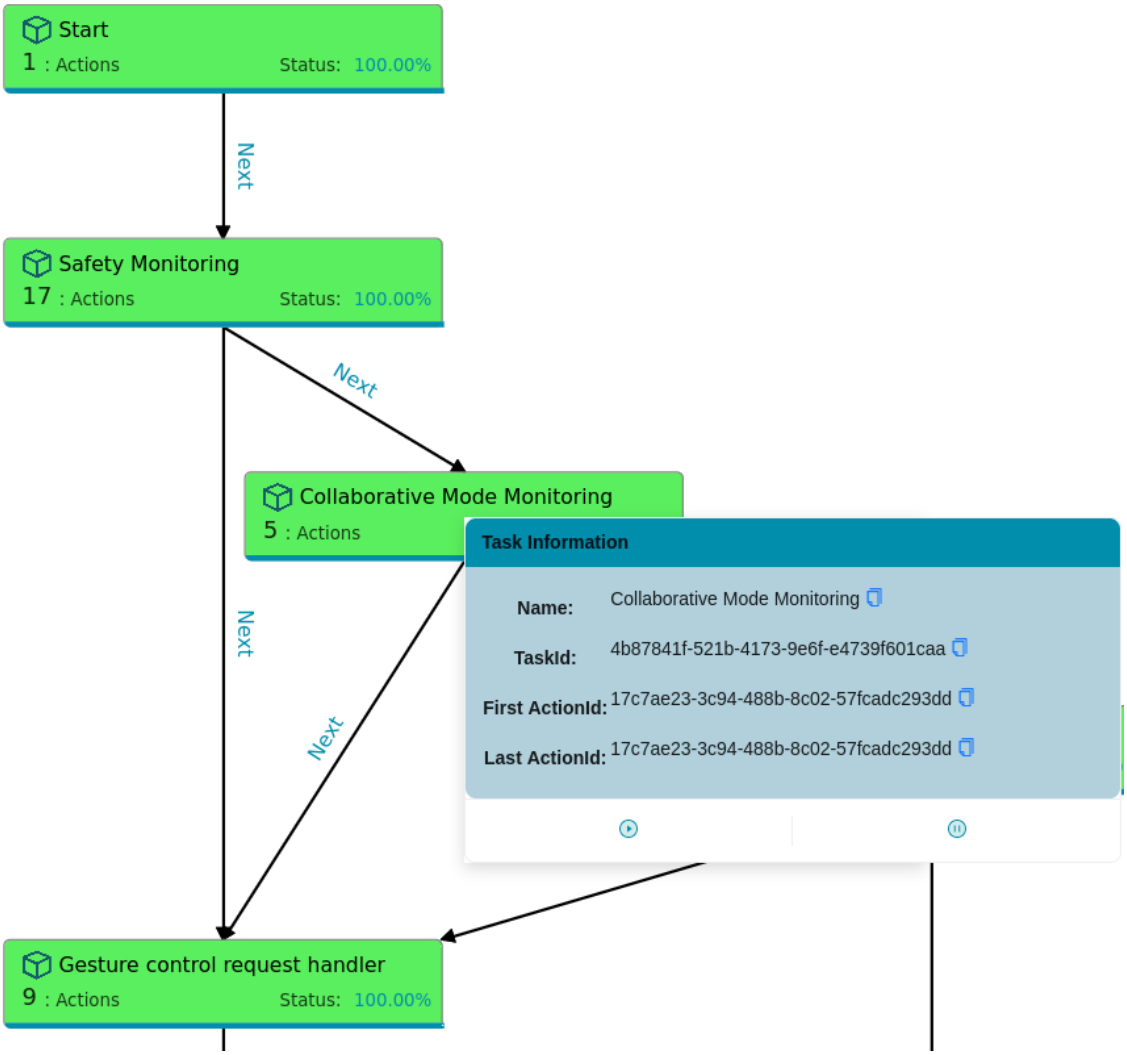

Figure 20: Tasks Live Graph (task information)

2.3.4. Product Plans Overview

The OpenFlow final prototype models the production information of products in a generic Product Plan description model. This model contains information that allows scheduling, rescheduling and seamless communication with the AI Task Planner. The OpenFlow provides the required functionality that allows users to view and inspect key information from the Product Plans as well as to request from the AI Task Planner a new schedule. The OpenFlow Product Plan screen displays only the Product Plans that the logged in User has the rights to view. Figure 21 is a screenshot of the Product Plan view, that shows the available product plans for the Automotive pilot case. Each Product Plan can be used to generate a Schedule through the Plan New Schedule option.

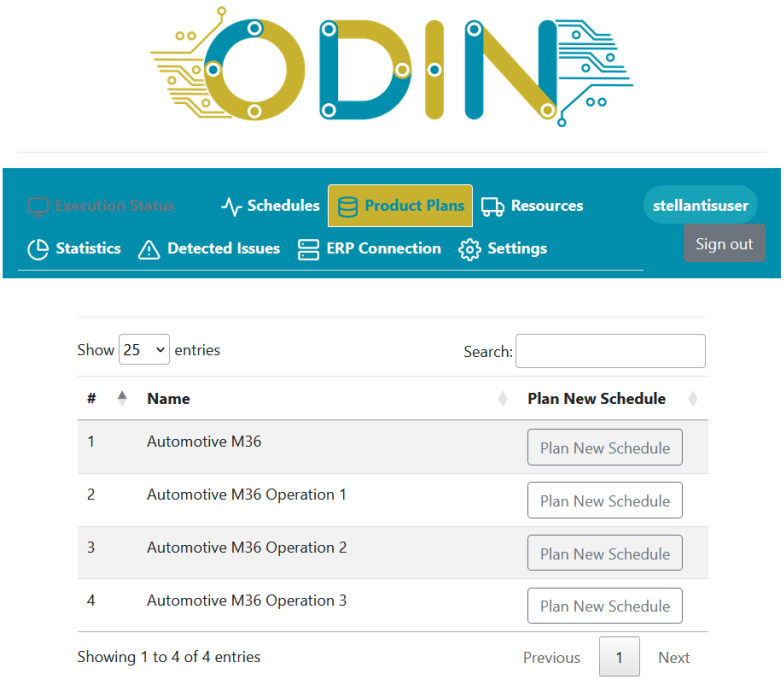

Figure 21: Open Flow UI: Product Plans View (Automotive Pilot Case)

The "Plan New Schedule" button of the Product Plan's OpenFlow user interface, shown in Figure 21 will send all required information to the AI Task Planner and receive the planning information, subsequently the OpenFlow will convert this information to an OpenFlow Production Schedule that can be executed and monitored through the OpenFlow final Prototype Module through the Execution Status View that is presented in section 2.3.3.

2.3.5. Resources View

The OpenFlow resources view displays all available resources to the company the User belongs to with a specific icon for easier recognition, which can be assigned as resources in Tasks and Actions of Schedules. Figure 22 shows the Resources of White Goods pilot case.

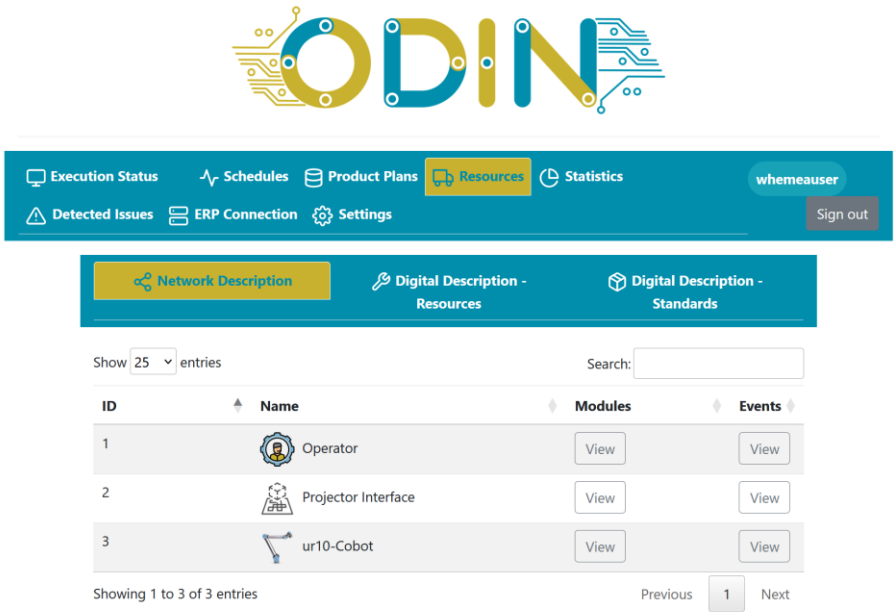

Figure 22: Open Flow UI: Resources View

Additionally, Modules column on each resource displays the available Network Resources that can be utilized from this resource in a Schedule. Figure 23 shows the available ActionLib Servers of ur10-Cobot for White Goods use case. OpenFlow UI also supports each network resource to have a specific assigned icon, though in this case for simplicity all ActionLib Servers have the same icon.

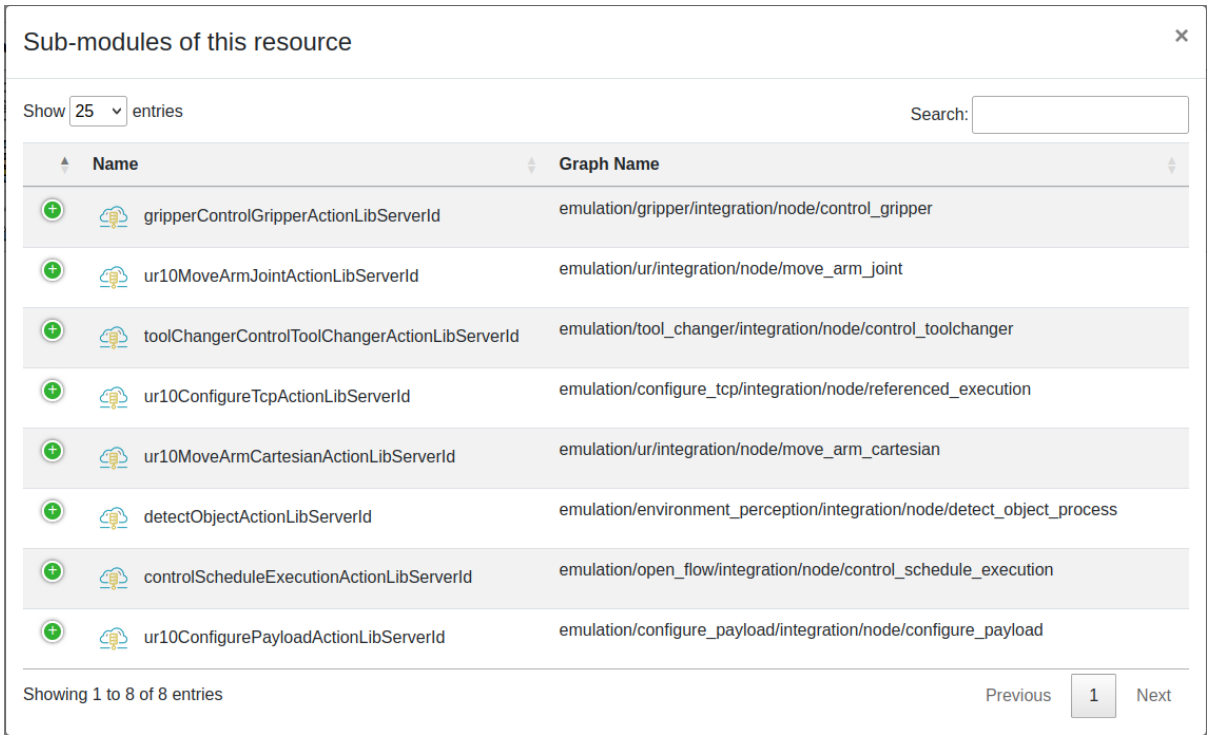

Figure 23: Open Flow UI: Resource's modules

Furthermore, in modules information window the User has the option to expand a specific network resource and gain access to additional information such as the ROS Definition and ROS MD5 hash for the ROS files being utilized for each Action, Service or Message as shown in Figure 24. This feature can be used from project partners through development procedures to verify the commonality of shared ROS definition files in both OpenFlow and external network modules.

Figure 24: Network Resource Information

2.3.6. Statistics View

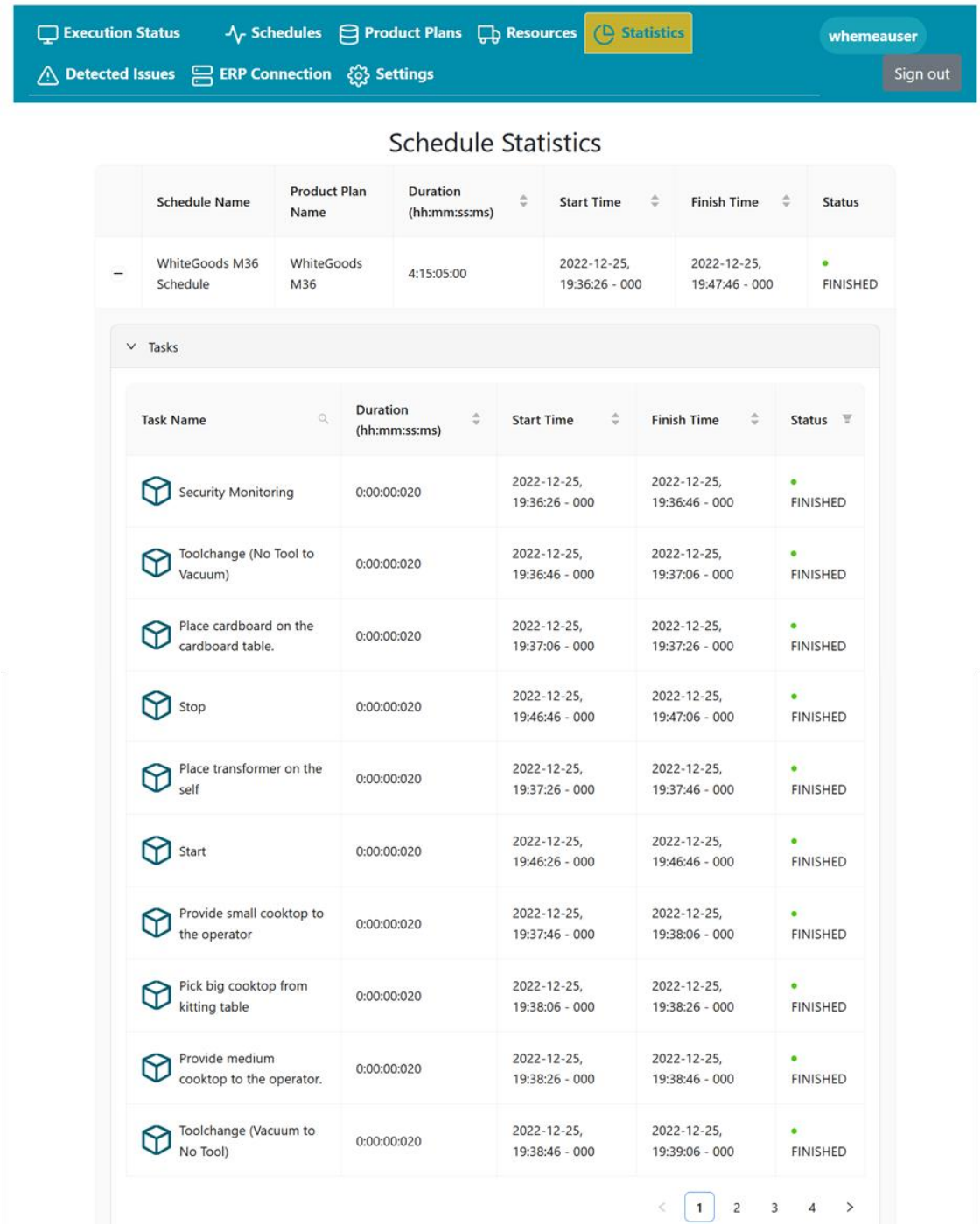

Figure 25: Schedule Statistics

The Statistics View features a metrics collection engine utilized by OpenFlow to present to the User runtime data collected during the execution of a production schedule. Such data include the start time, finish time, duration and status for each Task and Action in separate sections, as well as for the entire production schedule as shown in Figure 25. This provides a useful insight on whether a schedule has been completed successfully or in case of an error which tasks or actions have been completed and which ones were still idle at the time the error occurred.

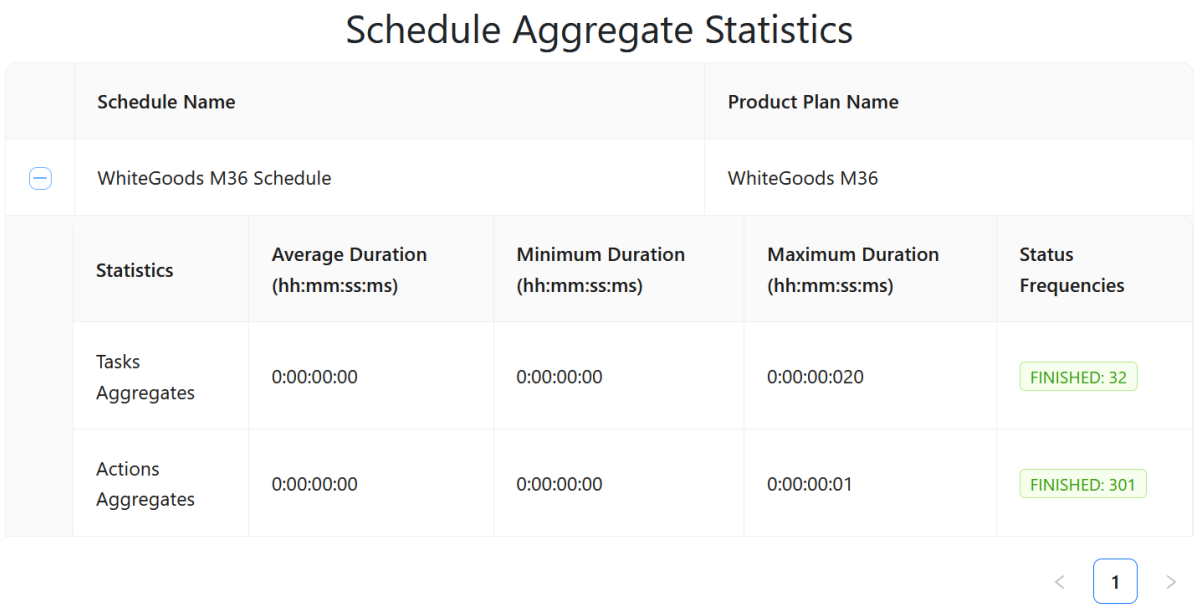

Figure 26: Schedule Aggregate Statistics

Additionally, on the same user interface the User has the option to review aggregate statistics data calculated from data captured during execution time, which include average, minimum, maximum duration times of Tasks, Actions alongside the total outcome of each Status states with quantities for each one (e.g., how many Tasks in total are finished, idle, active).

2.3.7. Detected Issues View

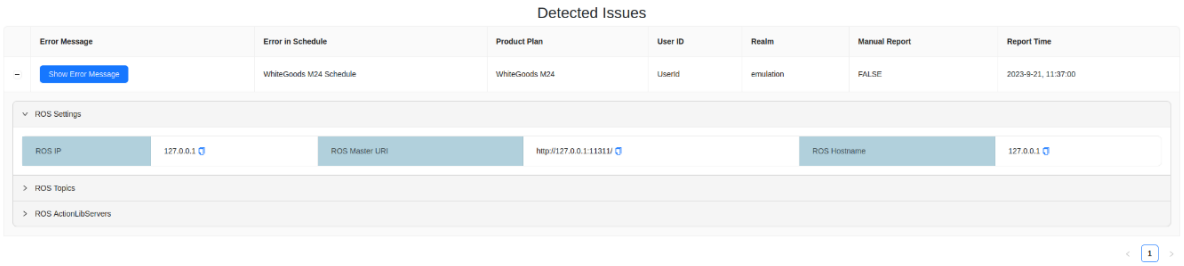

Figure 27: Detected Issues View

Detected Issues present the option of OpenFlow to log errors as they happen during execution and present them alongside system information during the runtime. Captured data include the schedule and product plan that was running when an error happened, the logged error message and captured timestamp as well as ROS environment settings as presented in Figure 27.

Additionally, through expanding the relevant sections in the same View, User can review the ROS network resources that were loaded during the execution when the error happened as presented in Figure 28 and Figure 29 for ROS Topics and ROS ActionLib servers respectively.

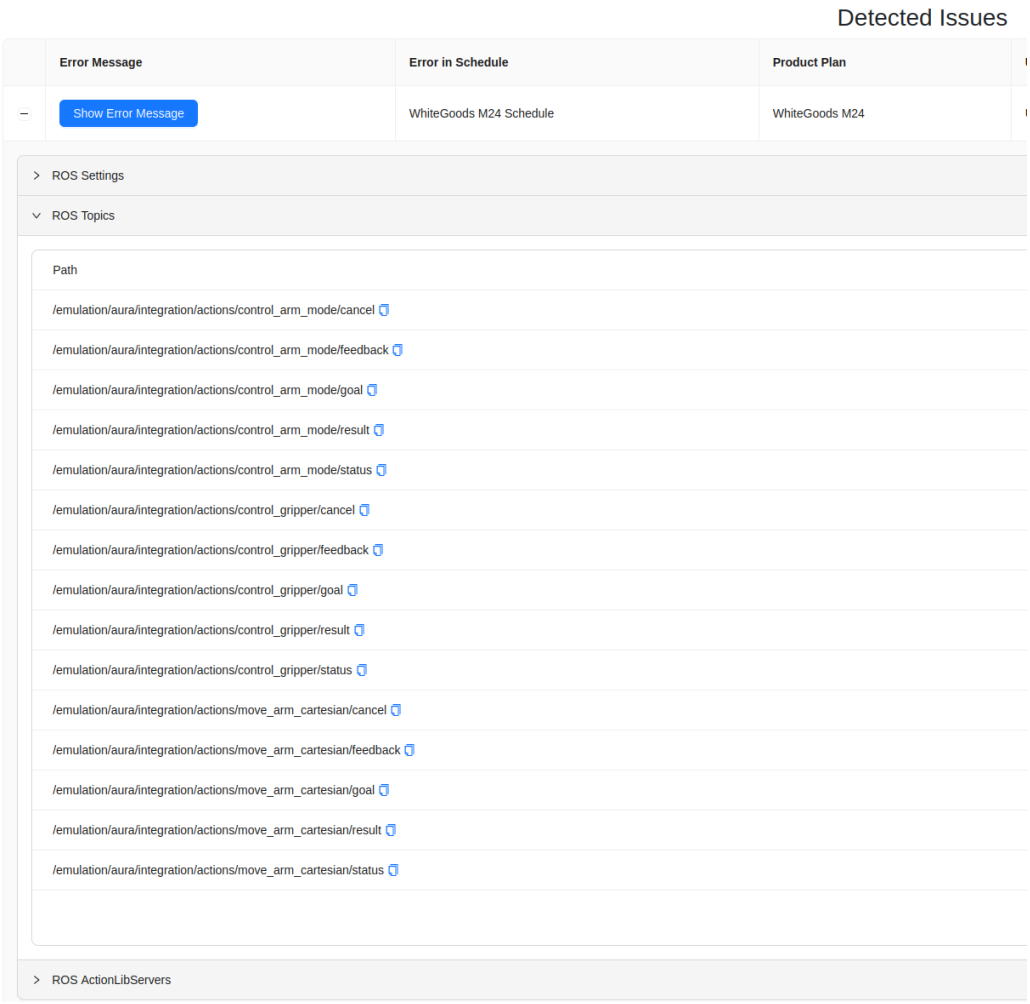

Figure 28: Detected Issues - ROS Topics

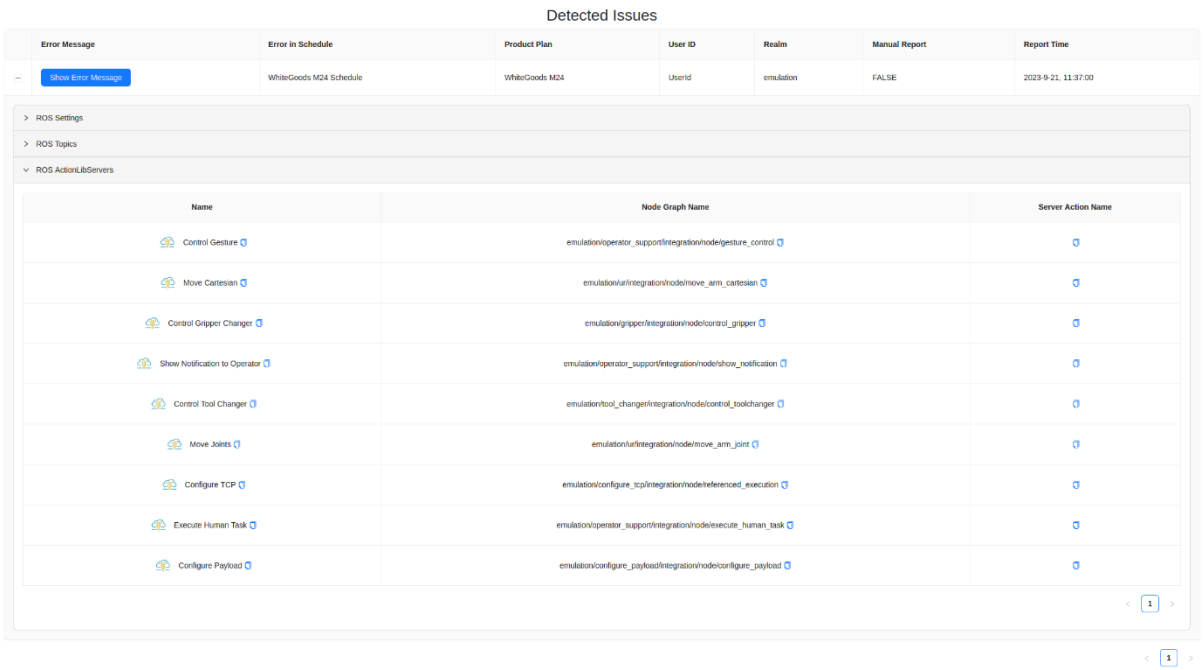

Figure 29: Detected Issues - ROS ActionLib Servers

2.3.8. ERP Connection Details

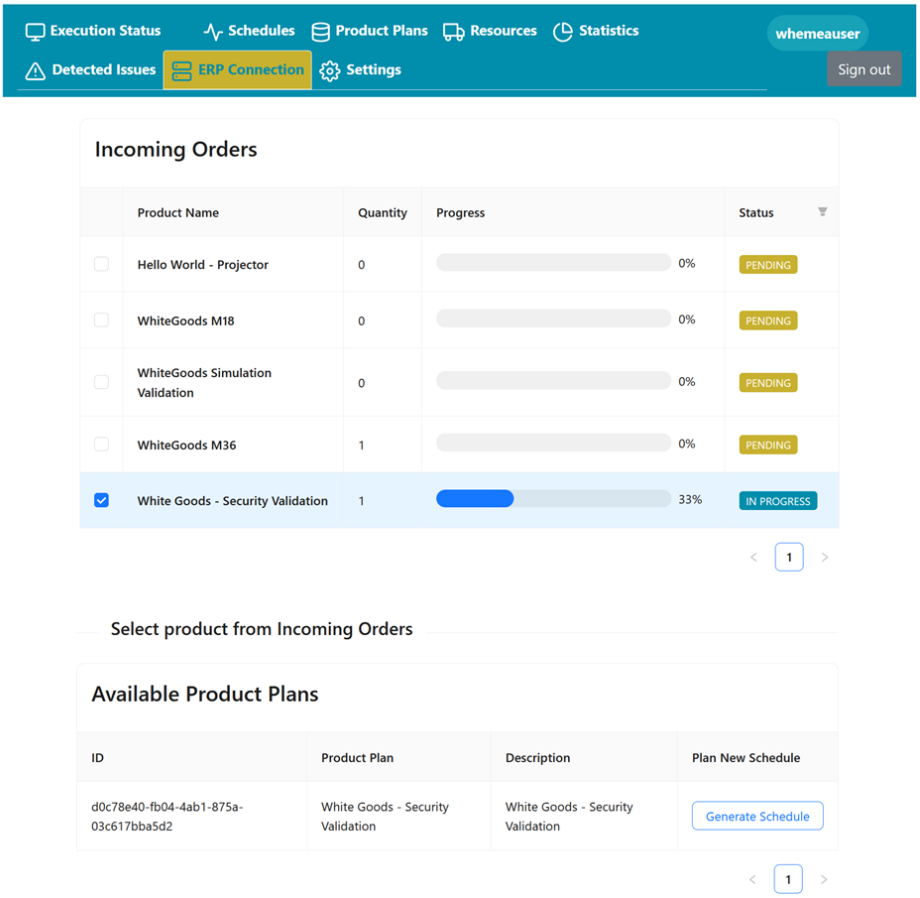

Figure 30: ERP Connection

The OpenFlow can accept incoming orders from external information systems. The incoming orders can be presented in the ERP Connection View. The OpenFlow also supports the process accepting incoming orders from external information systems with a dedicated UI. Figure 30, shows the table of Incoming Orders that presents the different incoming information as they arrive from external systems and the table of Available Product Plans that are used to fulfil the orders.

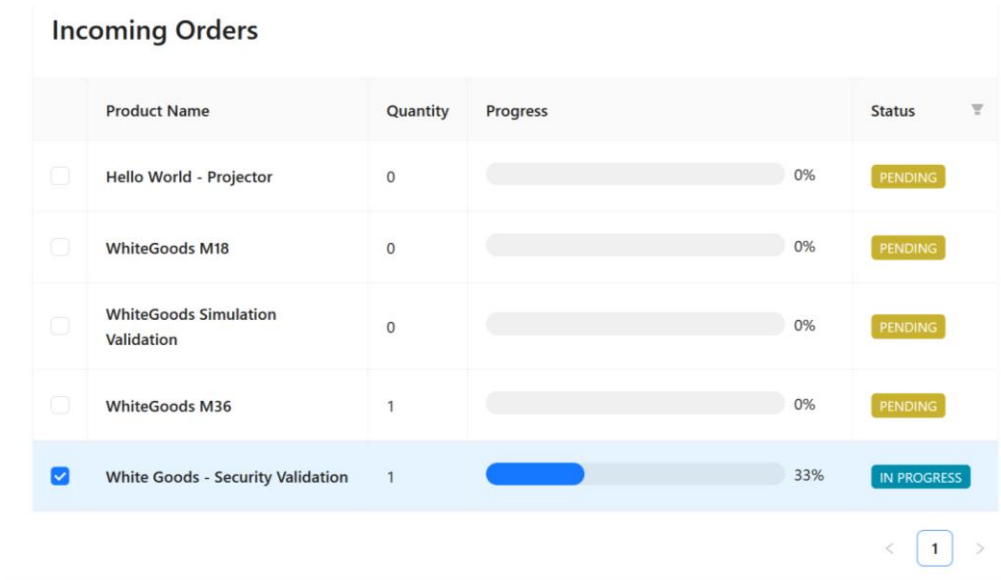

Figure 31: ERP Order Progress

Additionally, the OpenFlow can track and show in the ERP View the progress of the running schedule that are aimed to address specific received orders. This feature is implemented also using a WebSocket to display the status of the running schedule as presented in Execution View and Live Graph.

2.3.9. Digital Resource Descriptions

The OpenFlow has been integrated with the Digital Resource Description module of the ODIN Digital Component. The Digital Resource Description stores general description of production resources; their categorization data; their functions (capabilities/skills) and (HW) interfaces, including technical content; links to datasheets and other sources; images/photos; links to CAD/URDF models. The Digital Resource Description offers functionalities such as searching, filtering, querying, accessing and managing of Resource Description.

The final version of the OpenFlow module is integrated with the Digital Resource Description module. In particular, the OpenFlow backend is able to consume the REST API offered by the Digital Resource Description.

The integration has been implemented through webservices using the OpenAPI Specification for RESTful API design and in particular using the Swagger tool. The OpenAPI Specification (OAS) defines a standard, programming language-agnostic interface description for HTTP APIs. This allows both humans and computers to discover and understand the capabilities of a service without requiring access to source code, additional documentation, or inspection of network traffic. [9] Swagger is a tool that allows the description of the structure of the APIs in machine readable formats. [10]

The integration of the OpenFlow and the Digital Resource Description module takes place in the backend. The OpenFlow user interface server performs the required data exchange with the Digital Resource Description and then the information is used accordingly. The following paragraphs demonstrate how the information exchange is implemented for the Standards Digital Resources.

2.3.9.1. Resources Information

The Digital Resources Description user interface of the OpenFlow offers the user the possibility to view the resource descriptions that are available in the Digital Resources Description module.

The information is displayed in a table format as presented in Figure 32. In addition to the displayed information, the user can click on a link to view more information in the Digital Resource Description user interface.

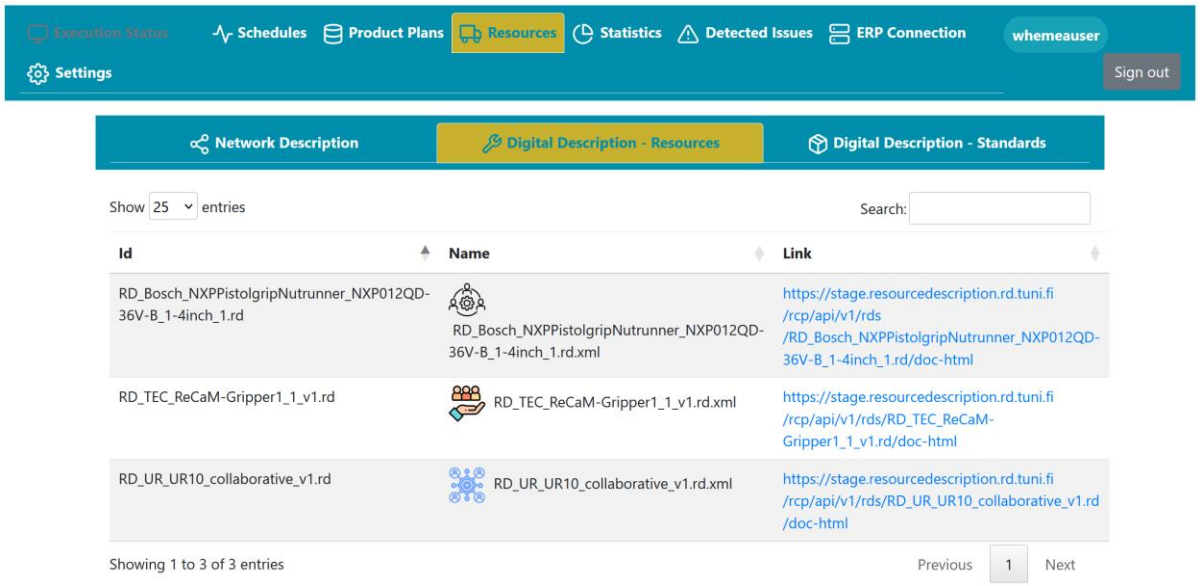

Figure 32: OpenFlow UI - Digital Resource Description Resources Table screenshot

The data flow for the implementation of the Digital Resources Description user interface of the OpenFlow is visualized in Figure 33. The Digital Resource Description Module server requires authentication in order to access the Digital Resources. The authentication is based on JWT (JSON Web Token). JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties [11].

More specifically, the OpenFlow server authenticates with the Digital Resource Description Module server using appropriate credentials and requests Resource Description information in order to respond to requests coming from the OpenFlow user interface. This information is displayed in the "Digital Description – Resources" table of the OpenFlow UI. The clicking of the link redirects the user to the Digital Resource Description Server.

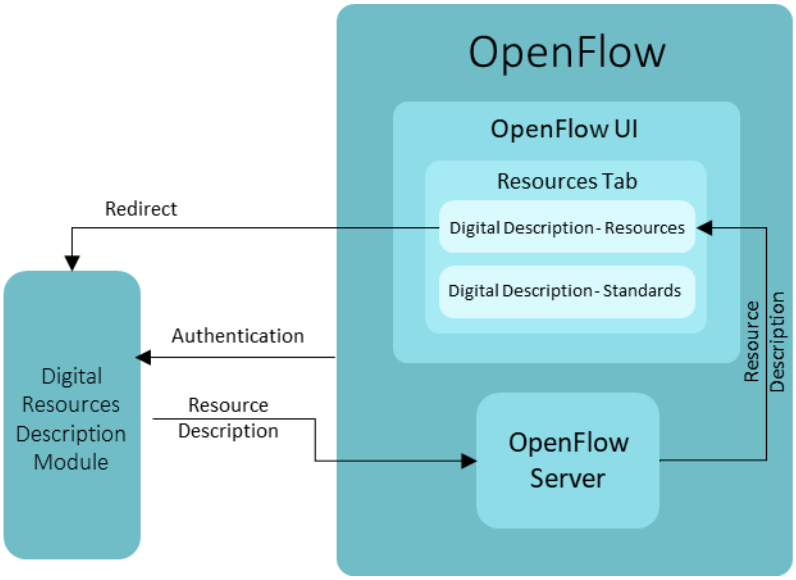

Figure 33: OpenFlow UI - Digital Resource Description Resources Functionality

2.3.9.2. Standards Information

The user interface for the display of standards within the OpenFlow framework allows users to seamlessly view a table of standard related information housed in the Digital Resources Description module. A user is presented with a visually intuitive table format displaying the information, as depicted in Figure 34. In addition to the presented details, users have the added convenience of accessing more in-depth information by clicking on the related link that redirects them to the Digital Resource Description user interface.

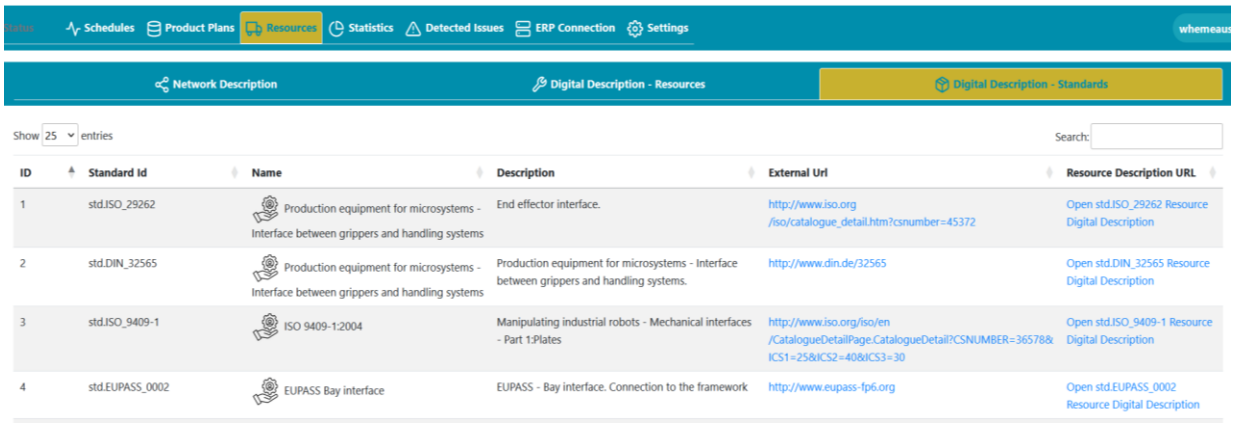

Figure 34: OpenFlow UI - Digital Resource Description Standards Table screenshot

The data flow for the implementation of the standards user interface of the OpenFlow is presented in Figure 35. The Digital Resource Description module server does not require authentication in order to access the standard metadata but may require authentication of the user once the he/she has been redirected in the Digital Resource Description UI.

The flow of information for the implementation of this functionality is as follows. The OpenFlow requests standard related information in order to fulfill requests coming from the OpenFlow user interface. This information is displayed in the "Digital Description – Standards" table of the OpenFlow UI. When the user clicks on the link, the OpenFlow UI redirects the user to the Digital Resource Description Server.

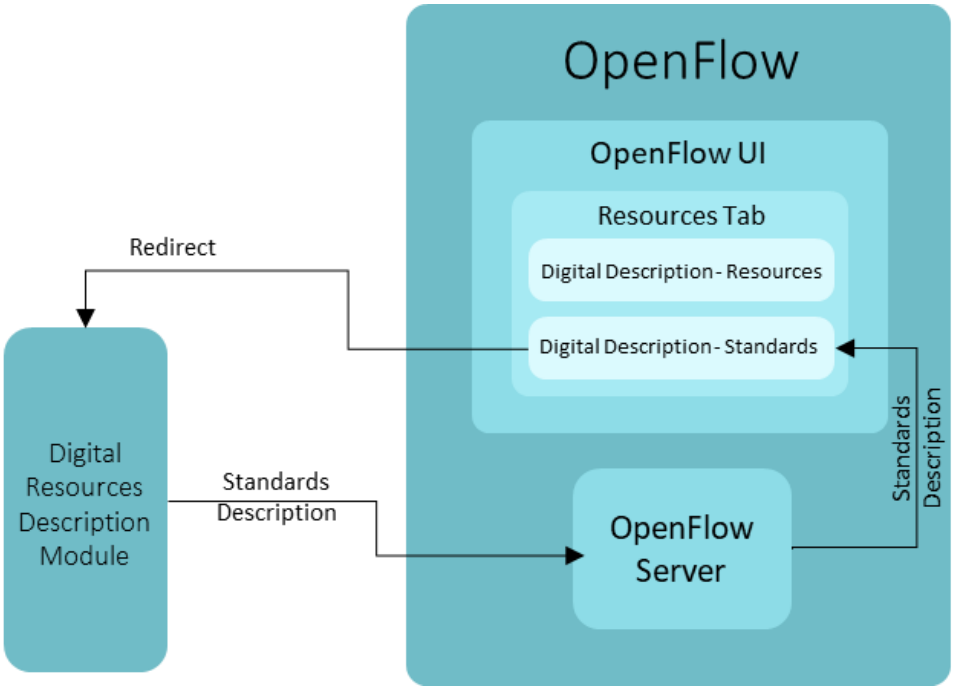

Figure 35: OpenFlow UI - Digital Resource Description Standards functionality

2.4. Technologies & Implementation

This section gives an overview of the design and implementation technologies that have been used to implement, test and validate the system during development.

The OpenFlow software was mainly tested in a PC equipped with an i7-3770CPU @3,40GHz and 16GB of RAM, running Ubuntu 20.04.03 LTS and uses ROS1 Noetic version and Java 17.0.1 64-bit and since September 2023 uses the latest LTS Java version, namely version 21. Information is stored, following the repository pattern, in a MongoDB which is a No-SQL, Document-oriented, database.

The OpenFlow orchestrator is developed in Java using the AKKA framework. The AKKA framework is an implementation of the actor model for the Java Virtual Machine (JVM). For the implementation of the immutable messages, as well as for persistence operations the "Immutables" annotation processors has been used to generate code for immutable object classes [3].

In order to test and validate the OpenFlow orchestrator a series of emulated, simulated and integration validation tests have been performed. D4.4 "ODIN Networked Component validation report – final version" provides a detailed description of the OpenFlow validation.

The integrated AI Task Planner module is also developed in Java. [4] The ROS Java library is used to develop the ROS interfaces for both the OpenFlow Orchestrator and the AI Task Planner modules.

During tests, different equipment has been used, such as a UR10 from Universal Robots [8] or a COMAU Aura Collaborative Robot [7]. Different approaches have been used for the motion planning

also, for instance using the open-source move-it platform [5]. The software developed for control and integration of the robot has been developed in C++. The same software is also responsible for exposing the ROS [6] interfaces for controlling the gripper, the tool changing and configuring the tool center point and the payload.

The AR glasses used are the Microsoft HoloLens 2, the related HMI software has been developed by using the Unity development platform and in the C# programming language.

The OpenFlow has been released in incremental docker images hosted in a private docker repository. Docker is a platform that facilitates building, sharing, and running applications using Docker images and containers. [17] These docker images have been distributed and used by the ODIN partners for different purposes such as integration and validation tests and demonstrations. These incremental versions helped partners integration and validation tests and also allowed the validation of OpenFlow in multiple scenarios.

3. CYBERSECURITY

3.1. Introduction

As a result of the work done in task T4.2 Cybersecurity and data processing in autonomous production environments, a private repository (https://github.com/ODIN-PROJECT-EU/odin-cybersecurity) has been created in the private ODIN Project GitHub Organization , where the developed components have been added, together with scripts for their easy installation and configuration. These components are properly documented so that the end users of the module can deploy and operate them by analyzing the cybersecurity of their environments.

The different components developed have been grouped into modules depending on the virtual machines where they will be installed. The modules developed are the following: Monitored Endpoint, SIEM and SOAR. The components that compose each of these modules are the ones that can be viewed in the Figure 36 and will be detailly presented in the next sections.

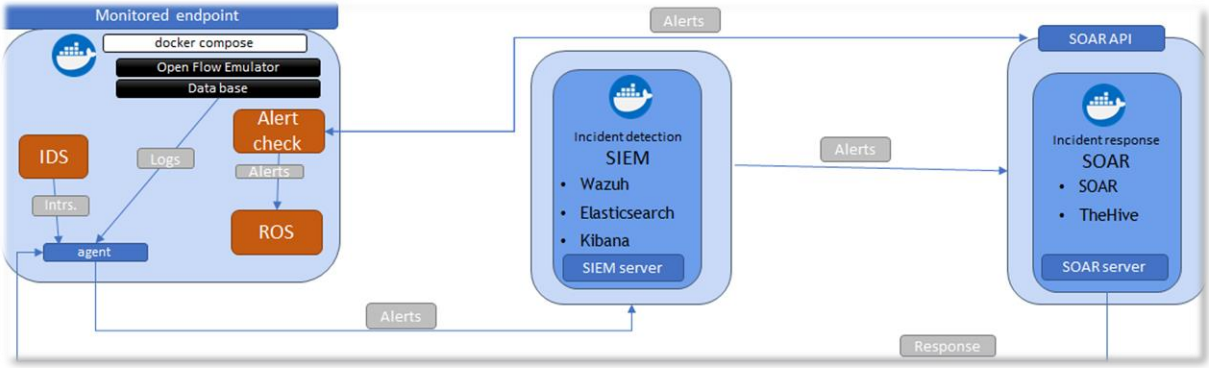

Figure 36: ODIN cybersecurity module architecture (final version)

3.2. Monitored endpoint prototype

The component that we have called monitored endpoint is the target component where cyber security will be analyzed. Within the project and in WP4, it is defined that the cybersecurity module developed in T4.2 will analyze the OpenFlow component described above. For this reason, and as can be seen in Figure 37, the monitored endpoint has been established on the machine where the OpenFlow emulator is installed.

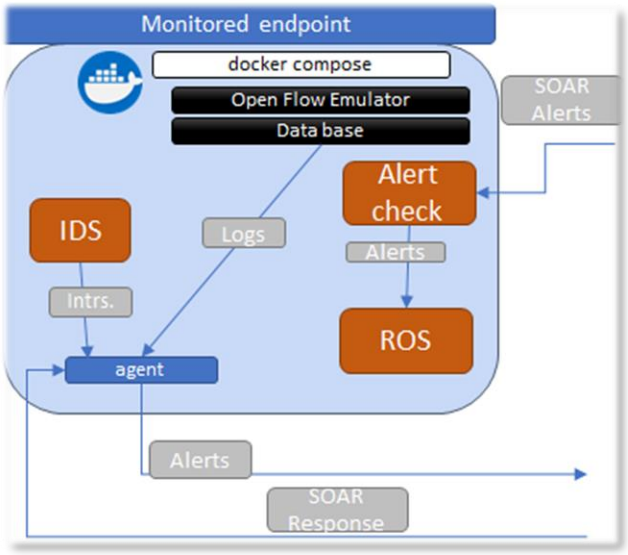

Figure 37: Monitored endpoint architecture (final version)

3.2.1. Design

A script has been developed that installs all the necessary software on the machine and configures the machine so that the cybersecurity of the monitored endpoint is analyzed. this script which is executed as `sudo python3 Start.py` installs and configures the following components:

- SIEM agent (wazuh-agent)
    - Configuration of rules for SIEM localrules and decoders and configure rsyslog (`/etc/rsyslog.conf`) and suricata to report cybersecurity alert
- IDS suricata
- ROS environment (set up roscore)
    - Expose ROS to the network

As a design assumption and due to the OpenFlow environment which is built upon docker for demonstration purposes, the design of the monitored end point components has been built upon docker so the following design principles are required:

- Docker and docker-compose (if they are not installed)
- Docker images for Open-Flow emulator

Finally, to connect SIEM and SOAR environment with OpenFlow and ROS, a design assumption was made in order to communicate both software modules via alert_checker.

- Alert checker

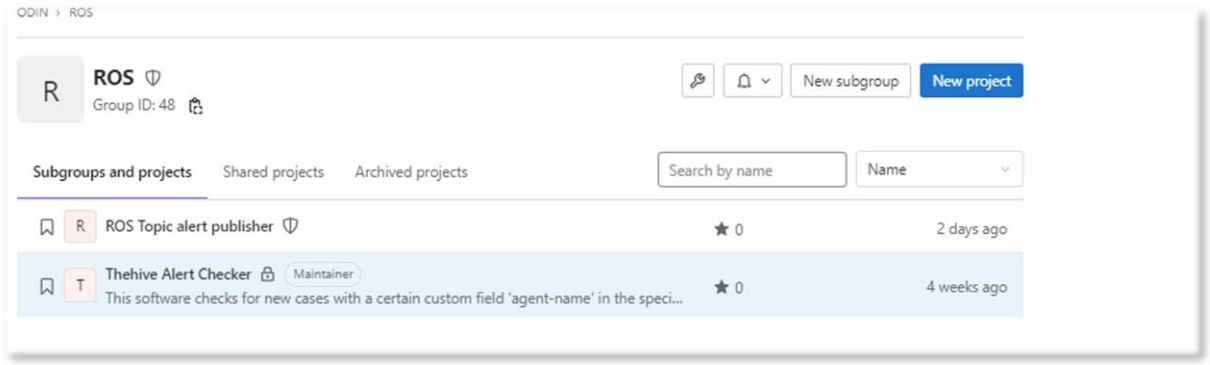

Figure 38: Alert checker modules

3.2.2. Functionalities Overview

This section describes the functionality of the different components that make up the Monitored Endpoint. OpenFlow's monitor requires the implementation of the components described in the following subchapters:

3.2.2.1. SIEM Agent

The SIEM agent is responsible for collecting, normalizing, parsing, and later transmitting security events and log data from various sources to SIEM server component. Host-Based IDS features are also provided, such as: real time OS authentication errors, real time monitoring on software installation and file monitoring among others.

3.2.2.2. IDS

The main feature of the intrusion detection system (IDS) is the ability to monitor and analyze network or system activities for signs of malicious behavior or security policy violations. IDS play a crucial role in cybersecurity by helping to detect and respond to potential security incidents. Key features of

IDS implemented are: Real-time network monitoring, Anomaly Detection, Signature-Based Detection, Alerts and Notifications and Network ID.

3.2.2.3. Alert check

The primary function of the alert_checker is to construct cybersecurity information by parsing input from a JSON file and making this information accessible by writing it to the ROS security topic, which is implemented by the OpenFlow.

To facilitate the functionality of this module, it is necessary to obtain in a first step the cybersecurity incident reported in the SOAR. In order to be able to get this information, a connector SOAR_alert_checker is designed to inspect new cases featuring a specific custom field, 'agent-name,' within the designated SOAR client or tenant instance. It then stores each case in a .json file located in the specified alerts folder. The software employs a tinydb file database to avoid duplicating alerts. Configuration options can be found in the 'config.py' file, with detailed explanations provided in the configuration section.

3.3. SIEM prototype

The SIEM component is in charge of receiving all the alerts generated by the components installed in the Monitored Endpoint, through the SIEM Agent. In addition to centralizing the alerts, it is responsible for normalizing them, enriching them with additional information and displaying them in a dashboard.

SIEM prototype creates an ecosystem that can monitor, analyze, and respond to security events within the OpenFlow in the industrial robotics framework.

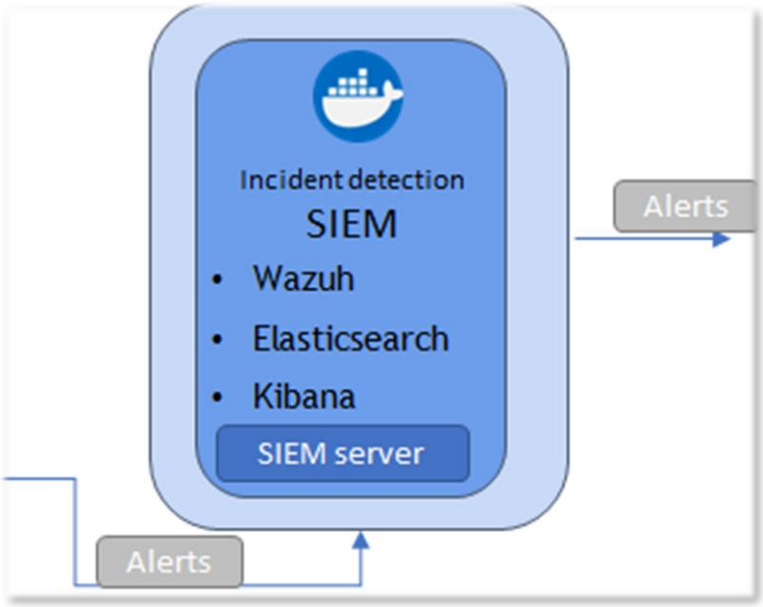

Figure 39: SIEM architecture (final version)

3.3.1. Design

To configure the installation environment of the SIEM component, a script has been developed. This script, `sudo python3 DockSet.py -i`, in particular installs the docker-compose so that the SIEM component can be easily deployed. In turn, the SIEM component has been packaged in docker-compose to facilitate the installation of the SIEM component once the environment is correctly configured.

Finally, for the configuration of the SIEM and the inclusion of the configuration files in it, once it has been deployed, the `python3 StartServices.py -c` script has been developed. This script completes the following phases:

1. Check Docker and docker-compose installation with the tag -c to invoke the DockSet.py and install Docker and docker-compose, if it is installed will skip it and continue.
2. Download docker images and install all needed packages for the SIEM.
3. Generate the ossec.conf, injectc into the docker wazuh-server container.
4. Stablish the rules local_rulex.xml and decoders local_decoder.xml into the docker wazuh-server container.
5. Create the integration between SIEM+SOAR.
6. Restart all services SIEM to apply configuration.
7. Automatic connection to docker kibana server thread-core to detect when the microServices are done and can be used.

After the execution of the script, it is possible to check where the services has been installed as presented in the Figure 40.

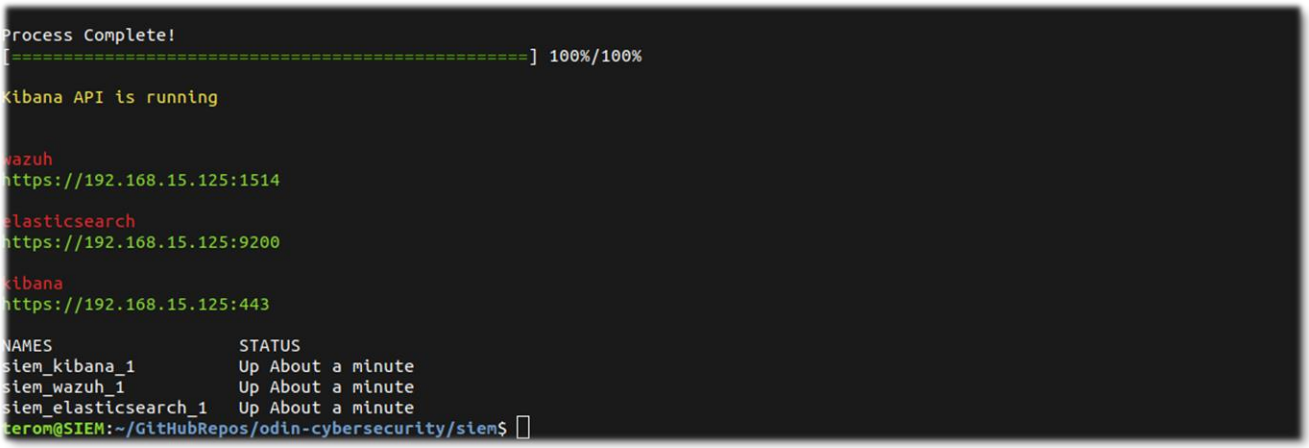

Figure 40: SIEM installation and configuration

3.3.2. Functionalities Overview

This section describes the functionality of the different components that make up the SIEM component.

3.3.2.1. Decoders

The alerts generated by the Monitored Endpoint first pass through the decoders. These decoders make it possible to normalize these alerts and index them so that they can be displayed on the SIEM dashboard. In this case, special decoders have been developed for the attacks detected in the ODIN project (logging errors, port scans, DoS and ROS attacks via ROSPenTo).

These decoders have been included in the local_decoder.xml file (Figure 41) so that SIEM can interpret the specific attacks, along with those that Wazuh is able to detect with its default decoders.

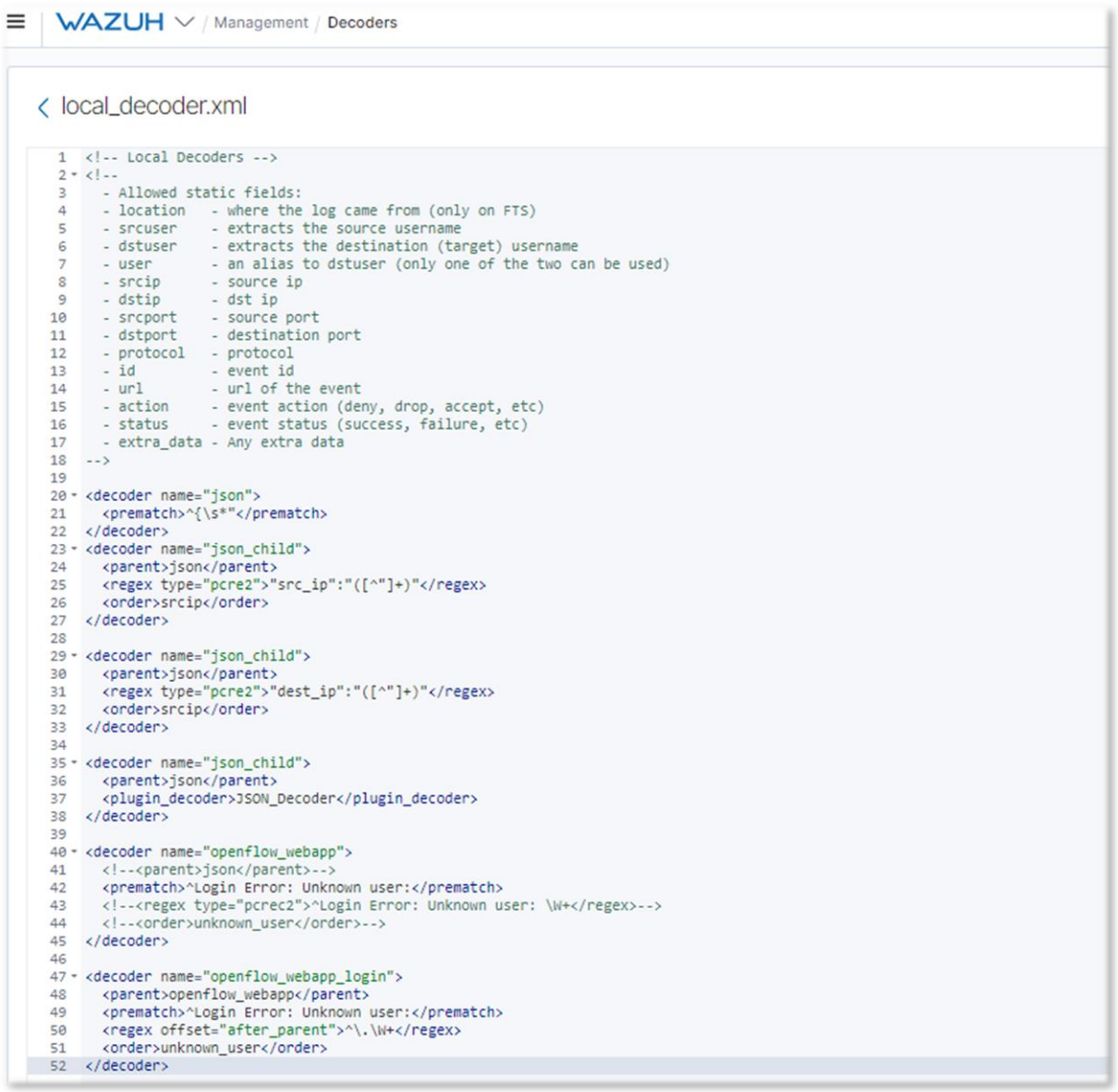

Figure 41: ODIN custom decoders

3.3.2.2. Rules

Once the alerts are detected in the Monitored Endpoint and parsed through the decoders, they must pass through customised rules. These rules allow us to include additional information to the alerts (such as the Mittre ID) by way of enrichment or other aspects (changing the rule ID) that will allow us to analyse these alerts appropriately. Specifically, this rule ID allows us to filter the alerts we are interested in and send only these to the SOAR (avoiding unnecessary noise).

In the same way as the decoders, the specific rules that have been created for the ODIN project have been put into a local_rules.xml file (Figure 42). These rules are added to the default Wazuh rules, giving a specific analysis of the particular attacks that have been selected in ODIN.

These attacks have been selected following the modelling of MagMa MITRE that has been done at the beginning of the execution of task T4.2. In the last year, in the integration of the pilots, it will be analyzed whether these detected attacks are sufficient or it is necessary to create more specific rules and decoders for other attacks that may appear in the pilot environments.

These rules are presented in the following figures, in particular Figure 42 and Figure 43.

Figure 42: ODIN Security – Custom Rules 1/2

```
<group name="openflow_webapp_unknown,">
  <rule id="100204" level="9">
  <if_sid>1002</if_sid>
  <match>Login Error</match>
    <description>OpenFlow Webapp Login Error Unknown user: </description>
    <mitre>
      <id>T1100</id>
    </mitre>
  </rule>
</group>


<group name="openflow_webapp_login,openflow_webapp_unknown_users,">
  <rule id="100205" level="10">
  <if_sid>100204</if_sid>
  <decoded_as>openflow_webapp_login</decoded_as>
  <description>OpenFlow Webapp Login Error Unknown user: $(unknown_user)</description>
  <mitre>
    <id>T1100</id>
  </mitre>
  </rule>
</group>

<group name="syslog,sshd,">
 <rule id="100206" level="12">
    <if_sid>5700</if_sid>
    <match>illegal user|invalid user</match>
    <description>sshd: Attempt to login using a non-existent user</description>
    <mitre>
      <id>T1110</id>
    </mitre>
    <group>invalid_login,authentication_failed,pci_dss_10.2.4,pci_dss_10.2.5,pci_dss_10.6.1,gpg13_7.1,gd
  </rule>
</group>

<group name="ids,suricata">
<rule id="100207" level="12">
    <if_sid>86600,86601</if_sid>
    <field name="event_type">^alert$</field>
    <match>SCAN Nmap Scripting</match>
    <description>Nmap scripting engine detected. TEST</description>
    <mitre>
      <id>T1595</id>
    </mitre>
  </rule>
</group>
```

Figure 43: ODIN Security – Custom Rules 2/2

3.3.2.3. SOAR integration

This component is responsible for collecting alerts from the SIEM and integrating them into the SOAR. As there is no default integration between SIEM Wazuh and SOAR The Hive, this component had to be specially designed and developed within the project. In the Figure 44 the developed two Python scripts can be seen.

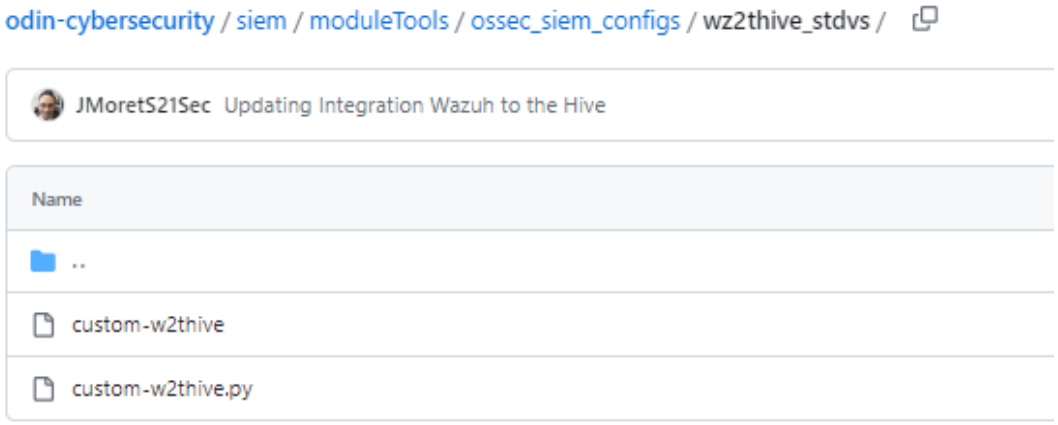

Figure 44: Custom SOAR integration

The functionality of this module (programmed in Python) is to detect the alerts that we are interested in for the project (those that represent that there has been an attack of those that are being monitored in the project) through the Wazuh API, parse, format and integrate them in The Hive. This module is also responsible for introducing additional data to the alerts (by way of enrichment) before they are

integrated into The Hive, providing interesting data that can facilitate the operator's decision-making process.

3.4. SOAR prototype

Once the alerts have been detected in the Monitored Endpoint and analyzed in the SIEM, they must be integrated into the SOAR so that a cybersecurity analyst can visualize them and provide the necessary response to resolve the incident. The following architecture (Figure 45) describes the SOAR module:

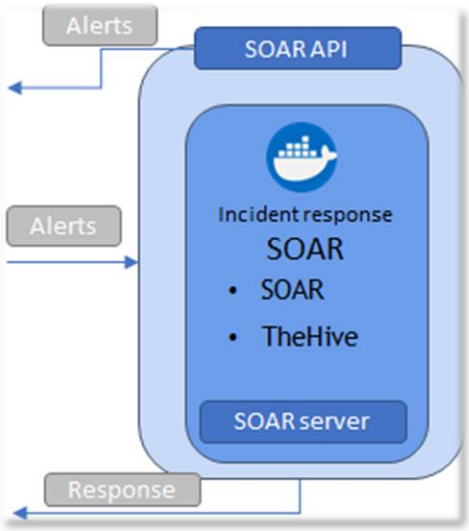

Figure 45: SOAR architecture (final version)

3.4.1. Design

The design of the SOAR component is equal to the SIEM based on a docker-compose environment. The following high-level services are implemented and orchestrated with docker-compose.

```
version: "3.8"
services:
  elasticsearch: ···

  kibana: ···

  cortex: ···

  cassandra: ···

  thehive:
    image: 'thehiveproject/thehive4:latest'
    #image: 'thehiveproject/thehive4:4.1.21
    container_name: thehive
    #restart: unless-stopped
    restart: on-failure
    depends_on:
      #- cassandra
      cassandra:
        condition: service_healthy
    ports:
      - '0.0.0.0:9000:9000'

    volumes:
      - ./thehive/application.conf:/etc/thehive/application.conf
      - ./vol/data:/opt/data
      - ./vol/index:/opt/index
    command: '--no-config --no-config-secret'
```

Figure 46: SOAR design

3.4.2. Functionalities Overview

The main functionality is that SOAR module is able to process alerts from the SIEM and manage a cybersecurity automation and response lifecycle for an incident that can be modelled using MITRE ATT&CK to detect and response.

3.4.2.1. Observables

Observables allow us to automatically extract key information from the alert that has arrived at the SOAR in order to be able to respond to the incident.

Observables refer to artifacts or entities that are analyzed and processed to gather information during the investigation and response to security incidents. It can be IP addresses, domain names, URLs, file hashes, etc.

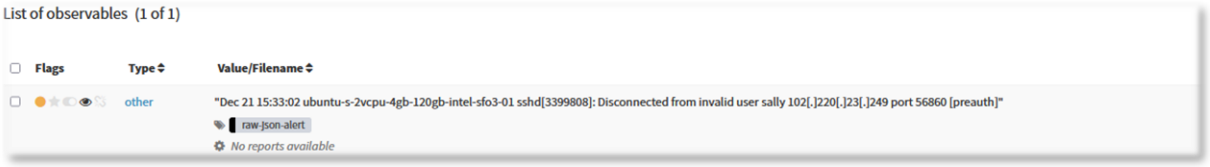

Figure 47: Example of observable type IP address

3.4.2.2. Responders

Responders refers to automated actions or scripts that can be executed as part of the incident response process. Responders play a crucial role in Security Orchestration, Automation, and Response (SOAR) by allowing security teams to automate repetitive and predefined tasks in response to security incidents. The following responders have been deployed.

- Automated Actions
- Script Execution
- Integration with Other Tools

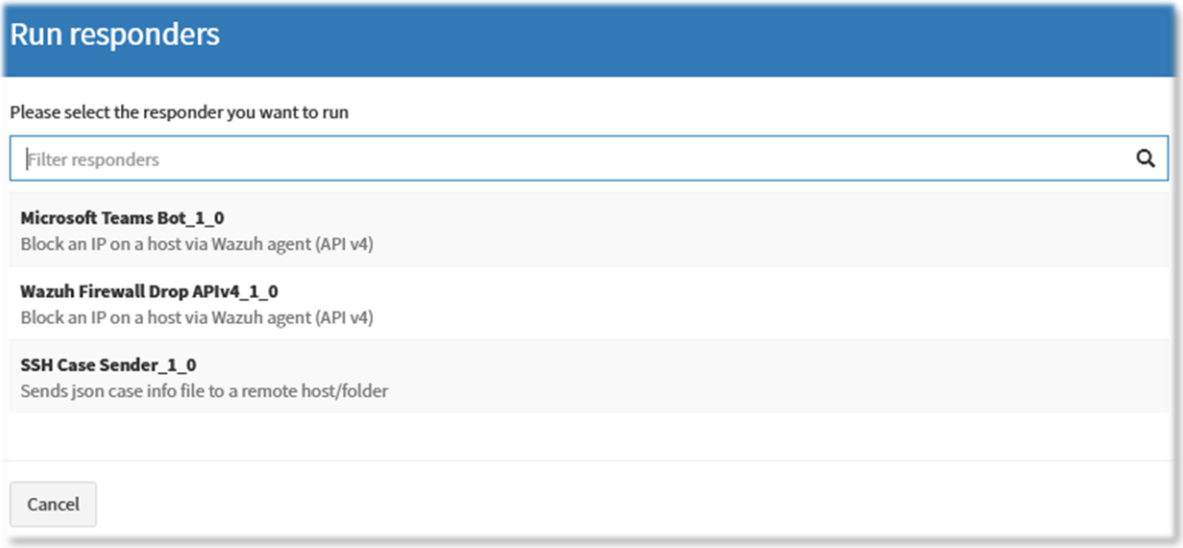

Figure 48: Responders implemented

4. CONCLUSIONS

The final prototype of the ODIN Networked Component has been completed successfully. This document presented the final prototype, that is comprised of two modules. Namely the OpenFlow and Cybersecurity modules have been described in this document. The ODIN Networked Component has also been submitted to a thorough validation process, that aims to validate the suitability of the software for the purpose of industrial usage in the framework of the industrial Pilot Cases of WP5. The evaluation results have been reported in ODIN Deliverable D4.4, titled "Networked Component validation report – final version".

The OpenFlow final prototype provides all the required functionality to integrate, orchestrate and facilitate the execution of a production schedule. Furthermore, the OpenFlow final prototype has been thoroughly evaluated. The evaluation results have been reported in ODIN Deliverable D4.4, titled "Networked Component validation report – final version".

The OpenFlow final prototype offers a full suite of SOAR and SIEM services combined towards the final ODIN Networked Component, allowing it to detect and respond to different kind of threats.

The next step for the ODIN Networked Component is to participate in the developments of WP5 and become part of the ODIN Industrial Component. In the context of WP5, the ODIN Networked Component modules will be responsible for orchestration and integration of the software modules of the Pilot cases, as well as for adding cybersecurity services to the system.

5. GLOSSARY

AI	Artificial Intelligence
API	Application Programming Interface
AR	Augmented Reality
DB	Database
DDD	Domain Driven Design
ERP	Enterprise Resource Planning
HRC	Human Robot Collaboration
IEC	International Electrotechnical Commission
IP	Internet Protocol
ISA	Industry Standard Architecture
HMI	Human Machine Interface
HRC	Human Robot Collaboration
KR	Knowledge Repository
MES	Manufacturing Execution Systems
OSINT	Open-Source Intelligence
OT	Operational Technology
PLM	Product Lifecycle Management
ROS	Robot Operating System
SCADA	Supervisory Control and Data Acquisition
SOA	Service Oriented Architecture
SOAR	Security Orchestration, Automation and Response
SIEM	Security Information and Event Management
SOC	Security Operation Centre
UI	User Interface
URL	Uniform Resource Locator
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
JWT	JSON Web Token

6. REFERENCES

1. Chryssolouris, G., Manufacturing Systems: Theory and Practice, 2nd Edition, Springer-Verlag, New York, New York, (2006)
2. S. Koukas, N. Kousi, S. Aivaliotis, G. Michalos, R. Bröchler, S. Makris, "ODIN architecture enabling reconfigurable human – robot based production lines", Procedia CIRP, Volume 107, pg 1403-1408, (2022)
3. Immutables, accessed online https://immutables.github.io/.
4. Evangelou G, Dimitropoulos N, Michalos G, Makris S. An approach for task and action planning in human-robot collaborative cells using AI. 8th CIRP Conference on Assembly Technologies and Systems, 2021;97:476-481.
5. Sucan I, Chitta S. "MoveIt", available at moveit.ros.org.
6. ROS, “ROS - Robot Operating System”, available at www.ros.org
7. COMAU Aura Collaborative Robot, accessed online https://www.comau.com/en/competencies/robotics-automation/collaborative-robotics/aura-collaborative-robot/
8. Universal Robots, accessed online https://www.universal-robots.com.
9. The OpenAPI Specification, Github. Accessed Online: https://github.com/OAI/OpenAPI-Specification/tree/main
10. What is swagger? Swagger site. Accessed Online: https://swagger.io/docs/specification/2-0/what-is-swagger/
11. JSON Web Token, Internet Engineering Task Force, Accessed Online: https://datatracker.ietf.org/doc/html/rfc7519
12. N. Kousi, S. Koukas, G. Michalos, S. Makris:"Scheduling of smart intra – factory material supply operations using mobile robots", International Journal of Production Research, Volume 57, Issue 3, pg. 801-814, (2018)
13. N. Kousi, S. Koukas, G. Michalos, S. Makris, G. Chryssolouris, "Service oriented architecture for dynamic scheduling of mobile robots for material supply", CIRPe2016 , Procedia CIRP, 5th CIRP Global Web Conference-Research and Innovation for Future Production Volume 55, pp. 18-22 (2016)
14. S. Papanastasiou, N. Kousi, P. Karagiannis, C. Gkournelos, A. Papavasileiou, K. Dimoulas, K. Baris, S. Koukas, G. Michalos, S. Makris, "Towards seamless human robot collaboration: integrating multimodal interaction", The International Journal of Advanced Manufacturing Technology, Volume 105, pg. 3881-3897, (2019)
15. G. Michalos, N. Kousi, P. Karagiannis, C. Gkournelos, K. Dimoulas, S. Koukas, P. Mparis, A. Papavasiliou, S. Makris,"Seamless human robot collaborative assembly – An automotive case study", Mechatronics, Volume 55, pg 194-211, (2018)
16. S. Makris, P. Karagiannis, S. Koukas, A. S. Matthaiakis, "Augmented reality system for operator support in human–robot collaborative assembly", CIRP Annals - Manufacturing Technology, Volume 65, Issue 1, pp. 61-64 , (2016)
17. Official Docker site, Docker, https://www.docker.com/ accessed online 2021-09
18. MaGMa: https://www.betaalvereniging.nl/en/safety/magma/
19. MITRE ATT&CK: https://attack.mitre.org