Title:	*Trustworthy & agile operation of smart IoT systems – Final version*
Authors:	*Franck Dechavanne (CNRS), Felix Feit (UDE), Nicolas Ferry (CNRS), Thibaut Gonnin (CNRS), Stéphane Lavirotte (CNRS), Wissam Mallouli (Montimage), Andreas Metzger (UDE), Edgardo Montes de Oca (Montimage), Ruben Mulero Martinez (TECNALIA), Phu Nguyen (SINTEF), Sarah Noyé (TECNALIA), Alexander Palm (UDE), Vinh Hoa La (Montimage), Gérald Rocher (CNRS), Jean-Yves Tigli (CNRS).*
Editors:	*Alexander Palm (UDE), Andreas Metzger (UDE)*
Reviewers:	*Arnor Solberg (TellU), Modris Greitans (EDI)*
Identifier:	*Deliverable # D3.3*
Nature:	*Other*
Date:	*30 November 2020*
Status:	*v1.0*
Diss. level:	*Public*

Executive Summary

Deliverable D3.3 reports on the final versions of the four enablers developed in WP3, namely (i) Online-Learning, (ii) Context Monitoring and Behavioural Drift Analysis, (iii) Root Cause Analysis, and (iv) Orchestration and deployment. This document thereby complements the deliverable of type OTHER, which comprises the executable code and documentation of the enablers provided in the ENACT online repository (https://gitlab.com/enact).
For the online learning enabler, this deliverable covers conceptual approaches to automate manual activities during design time enabling a software system to adapt itself during operational time and to incorporate structure of the adaptation space to guide exploration. A corresponding tool follows these approaches to enable a DevOps engineering to apply Reinforcement Learning algorithms to RL problems identified in arbitrary self-adaptive system with the ability to configure two well established RL algorithms and monitor the learning process

For the context monitoring & behavioural drift analysis enabler, this deliverable covers a visual analysis approach that allows Dev and Ops teams to study symptoms related to SIS behavioural drifts by highlighting differences between the expected behaviour (Dev) and the one observed in the field (Ops). For the root cause analysis enabler, this deliverable covers conceptual approaches to collect different system traces and to extract potentially relevant data attributes to build a labelled training dataset. Additionally, techniques to evaluate the importance of the attributes and to select the most relevant ones based on different Machine Learning techniques are provided. A knowledge-based approach to identify the root cause of detected incidents by analysing the symptoms and calculating the similarity of new events with the learned ones in the historical data concludes this enabler's main contributions.
The final version of the Orchestration and deployment enabler, comprises a common language that can be used by Dev and Ops teams to specify and to adapt a deployment. The language is platform and technology independent supporting the specification of adaptation in deployment over IoT, Edge and Cloud infrastructure. A final version of the GeneSIS models@run.time based engine is used to enact a deployment. Finally, a set of APIs for third parties to trigger an adaptation and manipulate a model is included.
The validation of the WP3 enablers using the ENACT use cases is reported in deliverable D1.5.

Members of the ENACT consortium:

SINTEF AS	Norway
BEAWRE	Spain
MONTIMAGE	France
EVIDIAN SA	France
INDRA Sistemas SA	Spain
Fundacion Tecnalia Research & Innovation	Spain
TellU AS	Norway
Centre National de la Recherche Scientifique	France
Universität Duisburg-Essen	Germany
Istituto per Servizi di Ricovero e Assistenza agli Anziani	Italy
Baltic Open Solution Center	Latvia
Elektronikas un Datorzinatnu Instituts	Latvia

Revision history

Date	Version	Author	Comments
07/09/2020	V0.2	Alexander Palm (UDE) / Andreas Metzger (UDE)	Draft of Table of contents and document structure
26/10/2020	V0.3	UDE, CNRS, MONTIMAGE, SINTEF	Initial content for the enabler sections
26/10/2020	V0.4	Alexander Palm (UDE) / Andreas Metzger (UDE)	First integrated version of the document (content of MONTIMAGE missing) for WP-internal review

27/10/2020	V0.5	Alexander Palm (UDE) / Andreas Metzger (UDE)	2nd integrated version for WP-internal review
11/11/2020	V0.6	UDE, CNRS, MONTIMAGE, SINTEF	3rd integrated version. Updated according to WP-internal review and ready for project-interal review
30/11/2020	V1.0	Andreas Metzger (UDE) / Alexander Palm (UDE)	Final version

Contents

1 Introduction

This deliverable describes the final results of the ENACT work package WP3 "Trustworthy & agile operation of smart IoT systems". WP3 developed enablers for the operational part of the DevOps life cycle (see Figure 1). To this end, the WP3 enablers facilitate Smart IoT Systems (SIS) with novel capabilities to (1) monitor and analyse when their behaviour drifts away from what is expected, (2) identify the root cause of the observed problems, (3) automatically learn when and how to perform self-adaptation at run time, (4) enact the dynamic adaptations of the deployed systems.

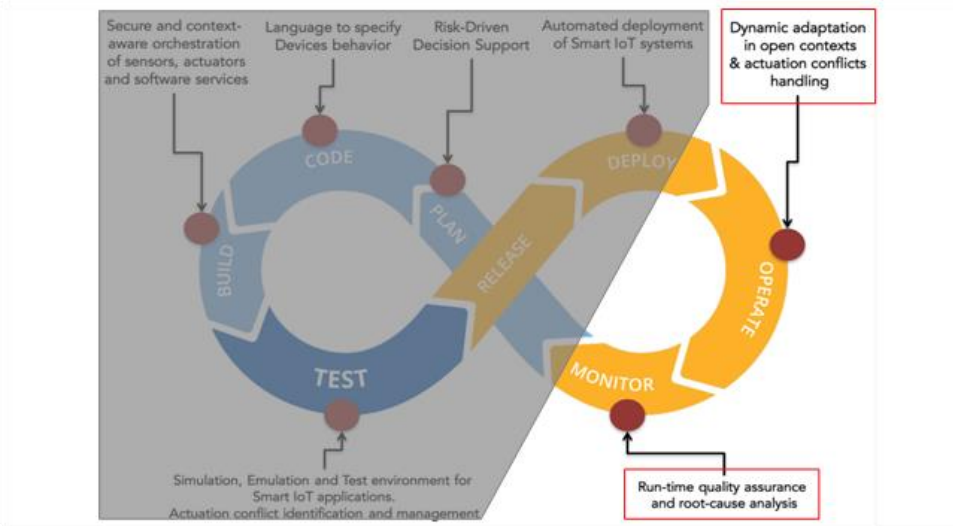


Figure 1: Focus on the Ops of the DevOps cycle

1.1 Architectural Overview

WP3 developed four main enablers which are depicted in Figure 2. This architectural overview also shows at which layer of the SIS the enablers are located and depicts their interplay among each other and with the Development activities of the DevOps life cycle.

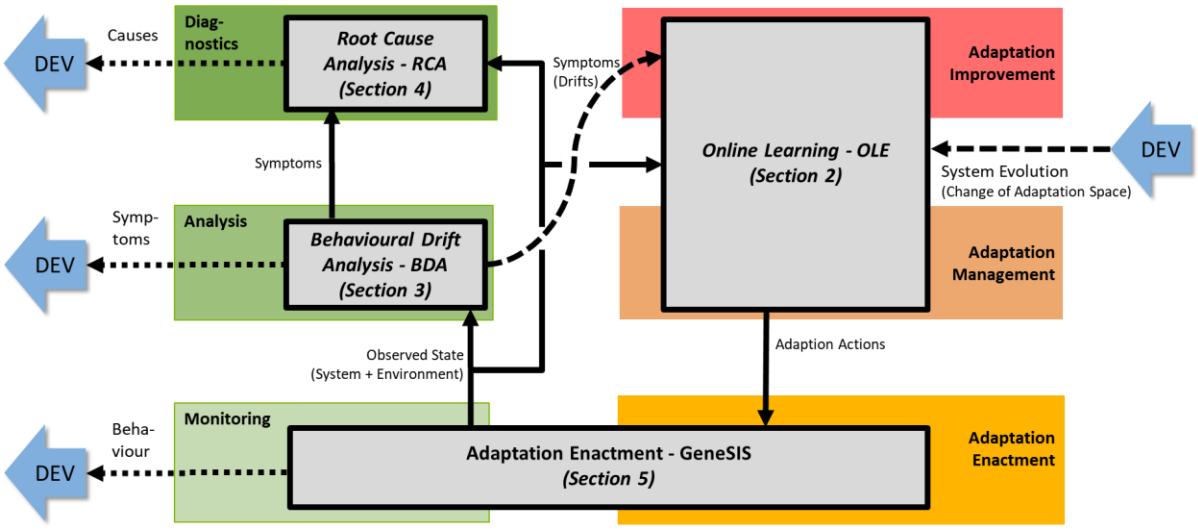


Figure 2: WP3 Architecture

The four enablers developed by WP3 are:

- **Online Learning Enabler:** Because anticipating all possible context situations that SIS may encounter during their operation is not possible, it is difficult for software developers to determine how a run-time adaptation of the system may impact the satisfaction of the system behaviour and of the interactions with the environment. To address this challenge, this enabler applies online learning techniques to improve the way a SIS adapts during its operation. Online learning means that learning is performed at run-time, taking into account observations about the actual system execution and system context. Online learning incrementally updates the SIS's knowledge base; *e.g.*, its adaptation rules or the models based on which adaptation decisions are made.
- **Behavioural Drift Analysis Enabler:** SIS, by interacting with the physical environment through actuators, are complex systems difficult to model. Formal verification techniques carried out at design-time are often ineffective in this context and these systems have to be quantitatively evaluated for the extent to which they behave as expected (effectiveness). This evaluation is achieved by confronting a model of the effects they have to produce in different contexts to the ones produced in the field. However, this evaluation is not informative on the symptoms of drifts in effectiveness. This enabler provides designers with tools for evaluating SIS effectiveness and analysing drifts that may occur (i.e., understanding the symptoms of the drifts and the underlying behaviour of the SIS).
- **Root Cause Analysis Enabler:** When anomalous conditions start to arise in a complex system, determining related part and anomalies to focus attention on is crucial to reduce the mean time to resolution (MTTR). Thus, the root-cause analysis enabler sensibly group anomalies related to the same problem and compute likely culprits of that problem with the least amount of human involvement possible. Since the number of open incidents in a large deployment can be large, it prioritizes as well the different grouped problems by their root-causes, potential impacts and recommended mitigation actions, based on past experience.
- **Orchestration and Deployment Enabler:** DevOps promotes an iterative and incremental approach enabling the continuous evolution of software systems. A key feature to enable such continuous evolution is to support the adaptation of the system with minimal impact on the system already delivered and under operation. The Orchestration and Deployment Enabler, which is part of GeneSIS our solution for the continuous deployment of smart IoT systems, provides the necessary mechanisms, interfaces, and abstractions for the Dev (for adapting as a result of a new development cycle) and Ops (for adapting to improve how the system operates) adaptation of the deployment and configuration of a SIS.

1.2 Change log from deliverable D3.3

Table 1 below provides the main changes and additions compared with deliverable D3.2.

Change	Section in D3.2	Section in D3.3	Rationale
Updated collection of achievements according to the final outcomes of WP3	Section 1.2	Section 1.3	Summarizes the achievements offered by ENACT WP3 tools described in this document in regards with ENACT WP3 objectives on SIS system security and privacy support.
New section	N/A	Section 1.4	Summarizes the support offered by ENACT Trustworthiness tools described in this document in regards with the properties of SIS system Trustworthiness.

Updated and restructured corresponding to final version of respective enabler	Sections 2–5	Sections 2–5	Work on respective enabler is now finished and evaluated. Prior work is summarized and final results were added.
Updated conclusions and future work	Section 7	Section 7	Section with final conclusions and planned research lines beyond ENACT project end.

Table 1: Overview of changes from D3.2

1.3 Achievements

This deliverable provides the final versions of the WP3 enablers with the main progress from the previous deliverable summarized in Table 2 below. **The executable code of the implementations is provided in the ENACT online repository (https://gitlab.com/enact) and a link to each subproject of each enabler is provided in Table 1. Documentation on how to use the enablers is provided in the respective readme-files of the subprojects in GitLab.** This technical documentation and code are complemented by a conceptual description of each enabler as well as the evaluation results for the enablers described in Sections 2–5 below. Additional integrations between different tools are described in Section 6.

Enabler	Progress from D3.2
Online Learning (https://gitlab.com/enact/online-learning-enabler)	The final version of the Online Learning enabler comes with the following main contributions: • Conceptual approaches to automate manual activities during design time enabling a software system to learn adapting itself during operational time and to incorporate structure of the adaptation space to guide exploration. • A corresponding tool following the mentioned approaches to enable a DevOps engineering to apply Reinforcement Learning (RL) algorithms to RL problems identified in arbitrary self-adaptive system with the ability to configure two well established RL algorithms and monitor the learning process. Compared to the initial release of the enabler in D3.2, the main evolutions are: • Evaluation of the proposed approaches and publishing results to several conferences as well as evaluation of the enabler in the Smart Building use case. • Update to the Graphical User Interface (GUI) of the corresponding tool providing further means for monitoring of the learning process. • Extension of the tool by Deep Q-Network (DQN) algorithm. • Highlighting means to improve the learning behaviour of RL techniques in an online setting by properly initializing a policy offline. • Providing proper APIs to enable the interconnection with other tools.

Context Monitoring & Behavioural Drift Analysis (https://gitlab.com/enact/behavioural_drift_analysis)	The final version of the Context Monitoring and Behavioural Drift Analysis enabler comes with the following main contribution: • A visual analysis approach that allows Dev and Ops teams to study symptoms related to SIS behavioural drifts by highlighting differences between the expected behaviour (Dev) and the one observed in the field (Ops). Compared to the initial release of the enabler in D3.2, the main evolutions are: • The behavioural drift analysis tool is now based on a new and generic clustering-based model learning algorithm. • Besides this algorithm, the tool has been complemented with a new algorithm that makes clear the differences between the model of the expected behaviour and the model learned from observations gathered from the field. • A dedicated GUI has been developed and integrated into the project GUI.
Root Cause Analysis (https://gitlab.com/enact/root_cause_analysis)	The final version of the Root Cause Analysis (RCA) enabler comes with the following main contributions: • Conceptual approaches to collect different system traces and to extract potentially relevant data attributes to build the labelled training dataset. • The techniques to evaluate the importance of the attributes and to select the most relevant ones based on different Machine Learning techniques. • A knowledge-based approach to identify the root cause of detected incidents by analysing the symptoms and calculating the similarity of new events with the learned ones in the historical data. Compared to the initial release of the enabler in D3.2, a prototypical implementation of the enabler has been developed and tested in generic cases as well as adapted for the ITS (Intelligent Transport Systems) use case, including the following features: • Data collecting (subscribing to MQTT messages, reading logs in json/ csv, capturing and extracting IoT-6LoWPAN traffic). • Data processing (normalization, attribute selection, similarity calculation). • Reaction and Visualization (publishing analysis results to MQTT exchange, displaying on GUI dashboards).

Adaptation enactment (GeneSIS execution engine)
 	
(https://gitlab.com/enact/GeneSIS)
	The final version of the Orchestration and deployment enabler (modelling language and execution environment) has been delivered. The main contribution of the enabler from the adaptation support perspectives are:
	• The same language can be used by Dev and Ops teams to specify and to adapt a deployment. The language is platform and technology independent supporting the specification of adaptation in deployment over IoT, Edge and Cloud infrastructure. This includes:
	  ○ The specification and enactment of rolling deployment (including redeployment) strategies in a platform independent way.
	  ○ The specification and (re)deployment of security mechanisms as part of the system in a platform independent way.
	• The GeneSIS models@run.time based engine is used to enact a deployment with the following benefits:
	  ○ During a redeployment/adaptation only the part of the system that needs to are adapted, minimizing the impact of a redeployment for better availability.
	  ○ The deployment model is maintained up-to-date, ensuring subsequent DevOps cycles to always work on up-to-date version of the deployment.
	  ○ Deployment models are always checked and validated before a deployment or redeployment is performed.
	  ○ Deployment activities can be delegated to deployment agents enabling the deployment on devices in local network with no or limited access to Internet.
	• A set of APIs for third parties to trigger an adaptation and manipulate a model.
	Compared to the initial release of the enabler in D3.2, the main evolutions are:
	• Support for blue/green (rolling) deployment and redeployment for maximum reliability and availability. Blue/green deployment can be specified in a platform-independent way.
	• Extending support for deployment of security mechanisms toward DevSecOps.
	• Refined model checking and comparison mechanism for finer grained adaptation and better reliability.
	• Introduction of the concept of Monitoring agent.
	• Plug-in mechanism to (i) dynamically load new component types and (ii) extend the GeneSIS deployment engine.

Table 2: Overview of achievements per enabler

1.4 Support to Trustworthiness in SIS

This section describes the characteristics of the support offered by ENACT WP3 tools to the five trustworthiness properties of SIS. Table 3 summarizes how each of the final tool prototypes developed in WP3 addresses each of the properties.

WP3-tool	Security	Privacy	Reliability	Resilience	Safety
Online Learning Enabler	No	No	Yes (adaptation and improvement of adaptation logic to keep working in unseen environment situations)	**No.**	**No.**
Behavioral Drift Analysis	No	No	Yes (monitoring and analysing drifts to trigger dev)	No	No
Root Cause Analysis	Yes (detecting cause of security flaws)	No	Yes (root cause of previously known performance problems)	Yes (root cause of previously unknown performance problems supported by ML techniques)	No
GeneSIS	Yes (specification of security requirements + deployment of security mechanisms)	Yes (via security)	Yes (Yes, via mechanisms such as blue/green deployment)	No	No

Table 3: ENACT WP3 methods and tools' support to trustworthiness in SIS at the time of D3.3 delivery

1.5 Structure of the document

The remainder of the document is structured as follows. After a brief introduction, Sections 2-5 provide for a comprehensive description of each of the enablers of WP3 highlighting the evolution since D3.2. Section 6 describes how the interplay among the WP3 enablers was realized. Section 7 concludes the document.

2 Online Learning for Adaptation Self-improvement of Smart IoT Systems

In this chapter we summarize the main achievements of Task 3.1 that has been made since the first review, as well as provide a description of the corresponding Online Learning tool. In the end of the chapter we give a brief outlook on what might be future work on our achievements beyond the ENACT project.

2.1 Overview and Main Achievements

The Online Learning Enabler offers means to apply Reinforcement Learning (RL) machine learning techniques to control problems whose underlying structure can be formulated as a RL problem. Thereby a tool is provided which relies on publicly available implementations of two popular RL algorithms, with an interface enabling to interconnect these algorithms with an arbitrary system via MQTT or HTTP. The tool has a graphical user interface (GUI) to enable a DevOps engineer to properly configure the tool and to provide several means to monitor the learning process. Furthermore, APIs are provided to interconnect the tool with other tools of the ENACT Framework.
Apart from the main achievements already described in D3.2 the following achievements have been made since then:

- The graphical user interface (GUI) of the Online Learning tool has been migrated from mineral.ui[1] (used in the first prototype) to ant.design[2]. (cf. Section 2.2.2)
- The possibility of using different Reinforcement Learning (RL) algorithms than Proximal Policy Optimization has been integrated. In particular the possibility to use value-based RL technique (Deep Q-Network [1]) has been added.
- A possibility has been added to load and save models (capturing a learned value-function and policy) of an experiment. This introduces the opportunity to do an offline-training within a simulated environment before applying and revising the learned policy in a runtime environment. This can heavily improve the initial performance of a policy in an online-setting (cf. Section 2.3.3).
- Further means for monitoring the learning process has been integrated, giving a developer the opportunity to mark changes in the running system (e.g. on a hardware level by changing a device) in the monitoring pane via REST API. This eases the interpretation of possible changes in the learning behaviour of the applied policy used by the RL algorithm. Furthermore, the enabler GUI offers opportunities to improve the visualization of variables as the amount of data points used for plotting diagrams can be adjusted which impacts performance of the GUI.
- Opportunities to visualize external variables (e,g, sensor data that might have impact on the learning behaviour or can be used to explain the system behaviour e.g. BDA) can now be fed into the online learning tool and be visualized accordingly. Signals can be fed into the tool via REST API or via MQTT as a topic.
- The online learning enabler has been evaluated using the Smart Building Use Case by using it to learn a proper Heating Ventilation Air-Conditioned system (HVAC) control strategy within a simulation of the KUBIK building (cf. Section 2.3).

In addition, several conceptual works have been done resulting in three publications [2] [3] [4]

- We motivated the application of policy-based RL as a concrete machine learning technique for the realization of self-adaptive software system (e.g. self-adaptive smart IoT systems) via online learning. Concomitant we proposed an approach for the integration of policy-based RL into the MAPE-K reference architecture and evaluating this approach with a cloud computing and a business process monitoring approach. (cf. [2] [3])

[1] https://mineral-ui.netlify.app/
[2] https://ant.design/

- Proposal for leveraging RL-technique to explore feature space in feature-oriented software (cf. [4])
- Ongoing work on reward decomposition as an approach towards explainable RL, that has not been published yet, but may result in a fourth publication

Regarding the status of all the features to be delivered at the end of the project as described in the "Extra Document"[3] are delivered as summarized in Table 4.

Feature	Description	Status at M22	Status at M35
Ease manual steps during design-time	See Sections 2.4.1, 2.4.2 & D3.2	Completed	Completed
Provide proper documentation	See Section 2.2.2 & D3.2	Completed	Completed and revised
Improve the behaviour of the system during operation using RL algorithms	See Sections 2.3.2.2, 2.4.2 & D3.2	Completed	Completed and revised
Enable the user to supervise/monitor the learning process	See Sections 2.2.1, 2.2.2 & D3.2	Completed	Completed
Provide ways to configure the corresponding tool	See Section 2.2.2 & D3.2	Completed	Completed and revised
Efficient usage of experience for policy updates	See Section 2.3.2.3	Ongoing	Completed
Provide further means for monitoring	See Sections 2.4.4 & 2.2.2	Ongoing	Completed
Handle changes during development time	See Section 2.4.3	Ongoing	Completed
Take behavioural drift into consideration	See Sections 2.2.1.1 & 6.1	Ongoing	Completed

Table 4: Summary of delivered features of the Online Learning Enabler

2.2 Description of Enabler

In this section we describe the technical implementation and main capabilities of the corresponding online learning tool. The tool implementation is available via GitLab (https://gitlab.com/enact/online-learning-enabler/) and via DockerHub (https://hub.docker.com/repository/docker/enactproject/online_learning_enabler).

2.2.1 Architecture of the tool (Backend)

In the following chapter we described the technical architecture of the corresponding online learning tool. An overview of the architecture is given in Figure 3.

[3] "Extra Document": Plans for development and evaluation for 2020 and until the end of ENACT. Submitted to the Project Officer in January 2020 as an additional document to the planned deliverables all the promised features

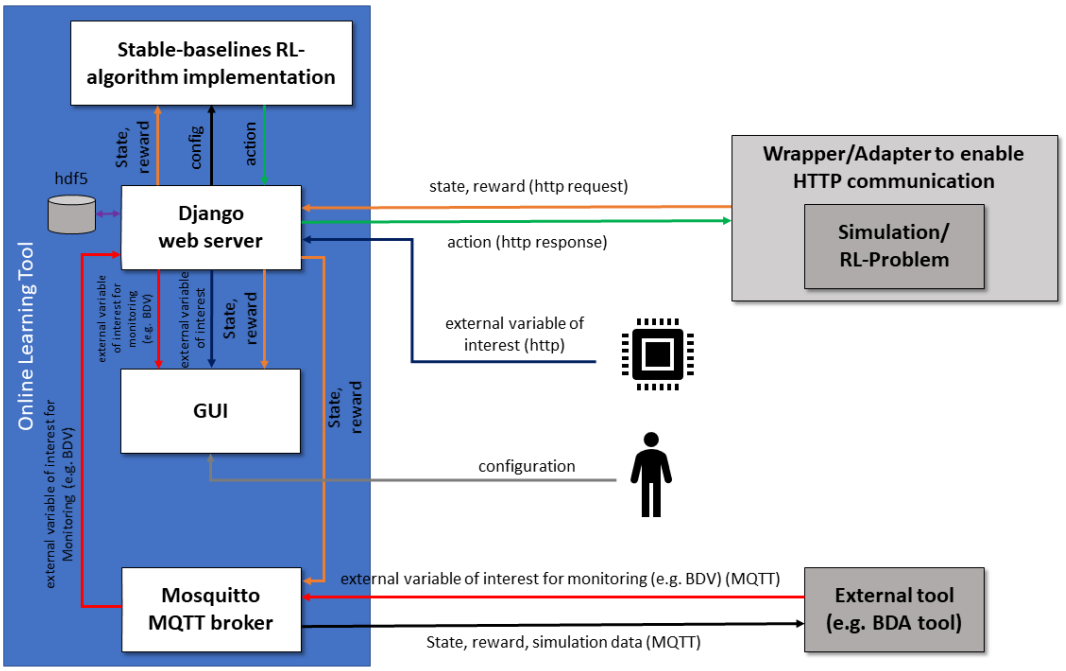


Figure 3: Architecture of the Online Learning Enabler

The tool is designed to allow arbitrary environments, which can be formalized as a Markov Decision Process (MDP), to be attached and solved with state-of-the-art RL algorithms. To do so, the environment must be wrapped into a script, which transfers the current state variables and the reward for the last state at each time step via an HTTP request. In response to this request, the script receives an action generated by the algorithm, which must then be executed in the environment.
While an environment is connected, it is possible to monitor the learning progress and configure the algorithm in a GUI. In addition to the standard RL metrics, external variables can also be displayed there. These have to be provided via a separate HTTP endpoint.
As a single-page application, the graphical user interface is a stand-alone program that is statically served and that communicates with the actual web server via HTTP.
The core of the tool is a Django web server, which offers the aforementioned HTTP endpoints and persistently stores all training data in HDF5 format. Besides that, the server contains a middleware that runs third-party implementations [5] of common algorithms in a thread and communicates with them via sockets.
Besides the possibility to connect external tools via HTTP, MQTT can also be used. For this purpose, the provided Docker Container runs an MQTT broker (Mosquitto) on which the observation variables as well as the last received reward is published at each time step. External data published in the broker is displayed in the GUI.

2.2.1.1 Interfaces

In addition to the graphical user interface, the tool offers two machine-to-machine interfaces that can be used to attach the environment of interest and to connect external tools. One of these interfaces is a web API that provides HTTP endpoints and the other is an MQTT broker.

Web API:

Method	Resource	Content-type	Description
POST	/api/submit_configuration	JSON	This endpoint is used to modify the configuration of the algorithm. This includes (1) the type of the algorithm, (2) observation-space and action-space, and (3) the hyperparameter of the chosen algorithm.

POST	/api/get_action	JSON	This endpoint is used to transmit the current state variables and the last reward to the algorithm. The response of the request contains the action to be executed.
GET	/api/get_environmental_log_data	text	This endpoint is used to obtain logs of the learning process. The query string can contain the parameter first_timestep, which causes only logs after this timestep to be returned.
GET	/api/external_data/	text	External data, for example from a connected tool, can be obtained by a GET request. For this purpose, as described above, the query string can contain the parameter first_timestep.
POST	/api/external_data	JSON	This endpoint can be used to transmit data of external sensors.
GET	/api/annotations	JSON	This endpoint can be used in two ways. Firstly, all diagram annotations can be obtained.
POST	/api/annotations	JSON	New annotations can be transmitted.
POST	/api/upload_model	JSON with Base64 string	This endpoint is used to load an existing model and its configuration into the tool. To do so the ZIP file generated by the save()method of Stable-Baselines 3 must be POSTed as a Base64 encoded string inside of a JSON.

MQTT: At each time step the observations are published in the MQTT-Broker. These can be found under `<use-case name>/observations/<observation variable>`. Data published from the outside can be displayed in the GUI.

2.2.2 Main capabilities of the Frontend

The frontend of the tool is divided in three panes: documentation, configuration and monitoring.

1. The documentation pane (Figure 4: Screenshot of the documentation pane of Online Learning Tool GUI) gives a full documentation about the tool summarizing also some conceptual work that motivates the usage of the tool. Furthermore, an example of a certain RL problem is given to introduce the underlying structure of problems that can be solved by means of RL and guiding a potential user of the tool through the configuration process of the integrated Reinforcement Learning algorithms (Proximal Policy Optimization (PPO) and Deep Q-Network (DQN)). Furthermore, the monitoring capabilities of the tool are explained.

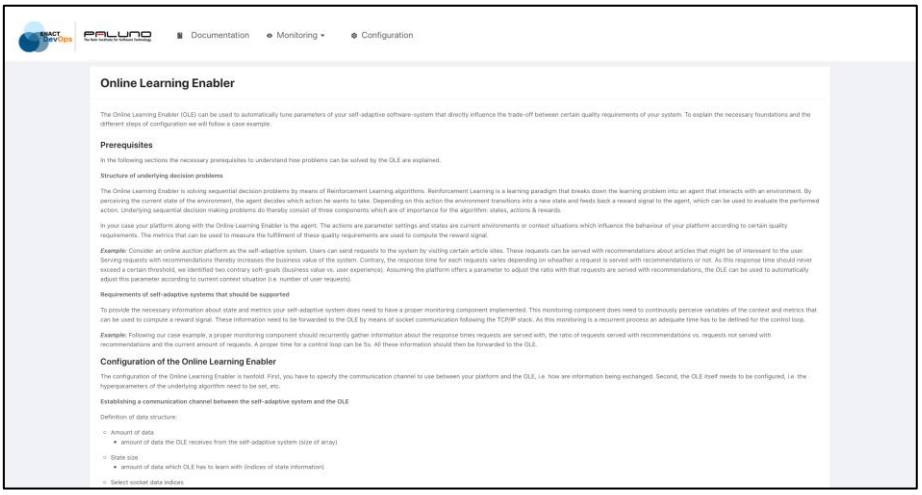


Figure 4: Screenshot of the documentation pane of Online Learning Tool GUI

2. The configuration pane (Figure 5: Screenshots of the configuration pane of Online Learning Tool GUI) enables the user to select a proper RL algorithm, define its hyperparameters, define the underlying learning problem as a RL problem and setup an interface for communication between the self-adaptive system and the RL algorithm managed by the Online Learning tool.

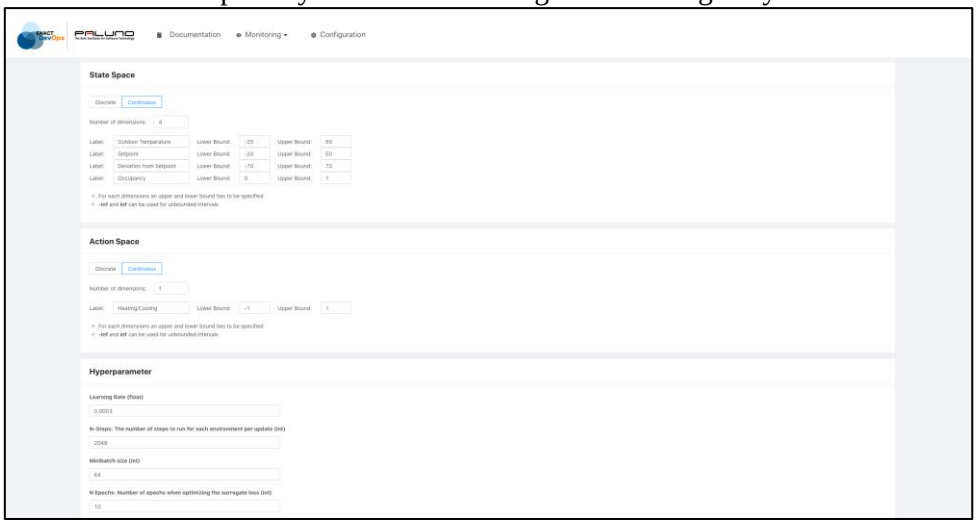


Figure 5: Screenshots of the configuration pane of Online Learning Tool GUI

3. The monitoring pane (Figure 6) gives the opportunity to supervise the learning process. Hereby it is possible to monitor the evolution of state and actions variables from the RL perspective and system and context variables from the domain perspective of the self-adaptive system. By adjusting the configuration of the RL algorithm and/or revising the underlying learning problem, new insights concerning an optimal control policy or at least how existing control policies might be improved can be identified. The monitoring pane can also be used to visualize additional variables computed by external tools (e.g. BDA) when a proper communication was set up in the backend.

Figure 6: Screenshot of the monitoring pane of Online Learning Tool GUI

2.3 Evaluation

To evaluate the Online Learning Enabler (OLE) it has been applied to a RL problem found as part of the "Smart Building" use case managed by TECNALIA. The RL problem basically comprises the process of finding a strategy to control an HVAC system in a certain room of the KUBIK building, that maximizes user comfort and minimizes energy consumption. This RL-problem is solved by means of a certain RL solution method: policy-based RL (with PPO serving as an algorithm candidate). A simple thermostat implementation thereby served as a baseline.

2.3.1 Description of simulation developed for the sake of the technical evaluation of the enabler

As it is not feasible to demonstrate the capabilities of the Online Learning Enabler by applying it to the physical facilities of TECNALIA without blocking these for an undefined amount of time while the facilities are used to run several other experiments in parallel (that would influence our experiments), we decided to setup a simulation based on the specification of the physical facilities. The simulation is written in Python programming language using the OpenAI Gym framework and can be found here: https://gitlab.com/enact/online-learning-enabler/-/tree/master/HVAC-Simulation.
The simulation is based on the following sensors and actuators from the KUBIK building (cf. Table 5):

Ground Floor - Reflected Ceiling Plan										
Device				Location				System		Signal
ID	Sensor	Actuator	Device type	Bedroom	Kitchen	Living Room	Corridor	IoT Smart Space Z-Wave wireless	Building Control PLC wired	Measure/Command type and Unit
9	X		Ceiling Multisensor 1		X			X		Motion (binary): YES/NO
9	X		Ceiling Multisensor 1		X			X		Temperature: degrees Celsius
9	X		Ceiling Multisensor 1		X			X		Light: 0 lux to 1000 lux
9	X		Ceiling Multisensor 1		X			X		Relative humidity: 20% to 95%
10	X		Ceiling Multisensor 2			X		X		Motion (binary): YES/NO
10	X		Ceiling Multisensor 2			X		X		Temperature: degrees Celsius
10	X		Ceiling Multisensor 2			X		X		Light: 0 lux to 1000 lux
10	X		Ceiling Multisensor 2			X		X		Relative humidity: 20% to 95%
11	X		Ceiling Multisensor 3			X	X	X		Motion (binary): YES/NO
11	X		Ceiling Multisensor 3			X	X	X		Temperature: degrees Celsius
11	X		Ceiling Multisensor 3			X	X	X		Light: 0 lux to 1000 lux
11	X		Ceiling Multisensor 3			X	X	X		Relative humidity: 20% to 95%
12	X		Ceiling Multisensor 4	X				X		Motion (binary): YES/NO
12	X		Ceiling Multisensor 4	X				X		Temperature: degrees Celsius
12	X		Ceiling Multisensor 4	X				X		Light: 0 lux to 1000 lux
12	X		Ceiling Multisensor 4	X				X		Relative humidity: 20% to 95%
13	X		Smoke Detector 1		X	X	X	X		Smoke alarm state (binary): ON/OFF
14	X		Smoke Detector 2	X				X		Smoke alarm state (binary): ON/OFF
15		X	Ceiling Light 1		X				X	Light state (binary): ON/OFF
16		X	Ceiling Light 2			X			X	Light state (binary): ON/OFF
17		X	Ceiling Light 3	X					X	Light state (binary): ON/OFF
18		X	Fan Coil 1		X				X	Operation state (binary): ON/OFF
18		X	Fan Coil 1		X				X	Temperature setpoint: Celsius
19		X	Fan Coil 2			X			X	Operation state (binary): ON/OFF
19		X	Fan Coil 2			X			X	Temperature setpoint: Celsius
20		X	Fan Coil 3			X			X	Operation state (binary): ON/OFF
20		X	Fan Coil 3			X			X	Temperature setpoint: Celsius
21		X	Fan Coil 4			X			X	Operation state (binary): ON/OFF
21		X	Fan Coil 4			X			X	Temperature setpoint: Celsius
22		X	Fan Coil 5				X		X	Operation state (binary): ON/OFF
22		X	Fan Coil 5				X		X	Temperature setpoint: Celsius
23		X	Fan Coil 6	X					X	Operation state (binary): ON/OFF
23		X	Fan Coil 6	X					X	Temperature setpoint: Celsius
24		X	Fan Coil 7	X					X	Operation state (binary): ON/OFF
24		X	Fan Coil 7	X					X	Temperature setpoint: Celsius

Table 5: Sensors & actuators of the KUBIK building used in our simulation (green: used in the simulation, red: not considered in the simulation)

In addition to these sensors and actuators, the current outdoor temperature is also recorded by a meteorological station on the roof of KUBIK. For the implementation historic weather data from year 2017 is used.

A sketch of the room which served as a template for the implementation can be seen in Figure 7 which is an actual excerpt of the KUBIK building specification provided by TECNALIA. This room has two fan coil units. To facilitate the implementation, a single fan coil unit with an equivalent capacity is considered. Since the detailed dynamic within the room is not modelled, only one sensor is considered for each type of measurement.

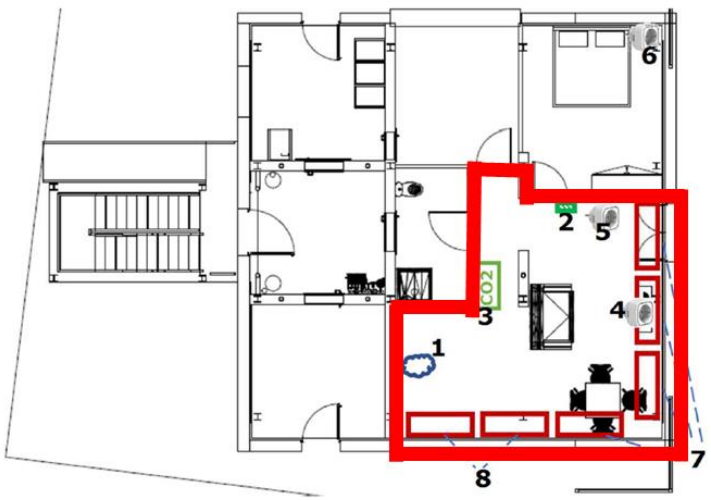


Figure 7: Excerpt of the KUBIK specification showing the room used for the simulation

2.3.1.1 Variables and parameters of the simulation

The simulation of the thermal behaviour of the room is realized based on the sensor variables:

- Outdoor temperature (T_{out}),
- Indoor temperature (T_{in})

and mainly influenced by the following actuator

- Fan coil.

Furthermore, we simulate the following variables for our RL problem:

- Occupancy,
- Predicted Occupancy.

The following parameters remain constant throughout our experiments and have been defined in cooperation with TECNALIA to be as close as possible to the properties of the KUBIK building (W=Watts, K=Kelvin, m=meter, Kg = kilogram, m²=square meter, m³ = cubic meter):

- Room dimensions: 6.14m (length), 6.891m (width), 2.388m (height);
- Window area: 29.56 m²;
- Outer wall area (excluding windows): 1.56 m² (the inner walls are neglected as they are not relevant for thermal exchange with the exterior);
- U-value wall: 1.7 $\frac{W}{m^2*K}$;
- U-value window: 0.19 $\frac{W}{m^2*K}$;
- Thermal resistance wall (R_{wall}): 2.652 $\frac{K}{W}$;
- Thermal resistance window (R_{window}): 0.178 $\frac{K}{W}$;
- Thermal resistance room (R_{room}): 2.83 $\frac{K}{W}$;
- CP-value air (CP_{air}): 1005.4$\frac{\mathrm{J}}{\mathrm{kg*K}}$;
- Air density: 1.225 $\frac{kg}{m^3}$;
- Internal air mass (M): 123,77 kg.

The fan coil unit is the controllable asset in our use case and its operating modes and conditions were defined as show in Table 6:

Fan coil mode	**Air flow rate (AFR; $\frac{m^3}{min}$)**	**Temperature (°C)**
heating low	260 / 60	38
heating medium	350 / 60	39
heating high	450 / 60	40
cooling low	260 / 60	17
cooling medium	350 / 60	16.5
cooling high	450 / 60	16
off	0	-

Table 6: Different modes of fan coil unit in KUBIK

2.3.1.2 Computation of thermal behaviour

We specified our simulation for a timestep of one minute and relied on common thermal equations to compute the increase in indoor temperature for each timestep. Hereby we computed the indoor temperature delta for each time step as follows:

$$\Delta T_{in} = \frac{1}{M*CP_{air}} * (Heatingflow + Coolingflow - Heatingloss).$$

Depending on the action selected the heating flow is computed as follows:

$$if\ action = heating\ \&\ T_{heating} > T_{in}: Heatingflow = (T_{heating} - T_{in}) * AFR * CP_{air}$$
$$else\ Heatingflow = 0.$$

Depending on the action selected the cooling flow is computed as follows:

$$if\ action = cooling\ \&\ T_{cooling} < T_{in}: Coolingflow = (T_{cooling} - T_{in}) * AFR * CP_{air}$$
$$else\ Coolingflow = 0.$$

Independent of the selected action, the heating loss is computed as follows:

$$Heatingloss = \frac{T_{in} - T_{out}}{R_{room}}$$

The **outdoor temperature** is updated according to the provided historic weather data. The original data set has hourly measurements. The data has been linearly interpolated to have weather data giving information for every minute.
For the sake of our RL-problem we also simulated the user **occupancy** for each timestep based on a fixed pattern following the scheme: Weekend from Friday 8pm to Monday 8am; Working days from Monday to Friday 8am to 8pm. Based on this basic pattern, we slightly shifted each change in the occupancy based on a gaussian distribution ($\sigma = 7.5$, $\mu = 0$) to simulate deviations of up to 30 minutes. Furthermore, we provided the additional variable **predicted occupancy** which basically is a probability of the person leaving or entering the room within the next 30 minutes.

2.3.2 Experimental results with simulation

In the use case scenario for the evaluation of the online learning enabler, we want to show that it is possible to learn a control strategy for the simulated HVAC system of the KUBIK building by means of policy-based RL. The learned control strategy should control the HVAC system in such a way that thermal comfort is achieved whenever the controlled room is occupied and energy consumption is minimized otherwise. The runtime of each experiment corresponds to one year of simulation, while one timestep of an experiment corresponds to one minute (i.e. total timesteps of one experiment: 524000). After several iterations of code improvements of the simulation we were able to simulate between 150 and 1000 timesteps per seconds depending on the hyperparameter (especially the size of the neural net) of the underlying algorithm.
To visualize the results of the different experiments, we plotted the outdoor temperature curve and the indoor temperature curve showing the moving average of both variables. For the moving average we averaged the last 128 values to reduce the noise inside the diagram and improve the interpretability of the results. Furthermore, we plotted the average indoor temperature per occupancy phase. This gives us the option to see whether the learned control strategy is able to avoid heating or cooling actions in occupancy phases where no user is present and to keep the indoor temperature close to the user setpoint whenever a user is present. Apart from this domain-related metric, we used the moving average reward of the last 10000 timesteps as a metric to evaluate the learning process from a RL-perspective. It is important to note that the values of the average reward are scaled to fit into the temperature diagram. The scaling factor is neglectable, as it is only the evolution of the reward curve that is important in this case. The exact evolution of the cumulative reward is shown in different plots throughout the evaluation chapter, when we compare the results of the RL approach to the baseline approach.
As a baseline we implemented a simple on/off-controller, that heats or cools the room whenever the indoor temperature is not close enough to the user setpoint and a user is present. If no user is present the thermostat controller remains inactive.
For the evaluation, in a first step we formulated the problem of learning an HVAC control strategy as a RL problem. The underlying Markov Decision Process (MDP) is thereby specified as follows:
State-space: The main state variables are stemming from the from simulation variables (e.g. indoor temperature). Furthermore, we created some crafted features resulting in variables relying on main state

variables (e.g. deviation from setpoint). The variable predicted occupancy returns the probability that the room gets occupied within the next 30 min. This variable is computed based on the underlying occupancy pattern, as we assume that such variables might exist in modern HVAC systems (cf. [6]).

Action-space: As the main task of the control strategy is to properly control the HVAC device, the action-space comprises 7 discrete actions, where only one action could be selected at a time. The actions correspond to the different modes of the fan coil for heating and cooling, as well as a turning he device off. The different modes relate to different fan speeds resulting in different air flow rates and different temperatures concerning the integrated fluids for heating or cooling respectively.

Transition-dynamics: The transitions between the different states (depending on the selected action) are computed by the simulation according to the underlying thermal equations. As we employ model-free RL algorithms, the control strategy does not have access to the environmental model resulting from the simulation. It is learning through pure interaction with raw experience. The simulation could be treated as a real environment, with the main difference that with a real environment the run time of an experiment would be much longer.

Reward: The main part of the MDP driving the algorithm in a direction to learn the right control strategy is the reward function. We defined the reward in such a way, that the algorithm gets a positive feedback whenever its control strategy keeps the indoor temperature close (i.e. within bounds of 1°C) to the user setpoint when a user is present and penalized otherwise (i.e. negative feedback corresponding to the deviation from the setpoint). As the control strategy should minimize energy consumption, the algorithm gets penalized according to the strength of the current action, whenever it performs an action other than turning the HVAC system off, if no user is present in the room (based on the simulated occupancy).

A more comprehensive listing of the MDP-elements can be found in Table 7.

State s = (T_{out}, T_{int}, SP_user, Dev_sp, o, o_{pred})	Outdoor temperature: $T_{out} \in \mathbb{R}$ Indoor temperature: $T_{in} \in \mathbb{R}$ User set point: $SP_{user} \in \mathbb{R}$ Deviation from set point: $Dev_{sp} \in \mathbb{R}$ Occupancy: $o \in \{0, 1\}$ Predicted occupancy: $o_{pred} \in [0, 1]$
Action a $\in \{0, 1, 2, 3, 4, 5, 6\}$	0: Heating off & cooling high 1: Heating off & cooling medium 2: Heating off & cooling low 3: Heating off & cooling off 4: Heating low & cooling off 5: Heating medium & cooling off 6: Heating high & cooling off
Reward $R = f(a, o) * g(Dev_t, o)$	$f(a_t, o) = 0\ if o = 0\ \&\ a_t = 0$ $f(a_t, o) = -33\ if\ o = 1\ \&\ a_t \in \{1,\ 4\}$ $f(a_t, o) = -66\ if\ o = 1\ \&\ a_t \in \{2,\ 5\}$ $f(a_t, o) = -100\ if\ o = 1\ \&\ a_t \in \{3,\ 6\}$ $g(Dev_t, o) = 0\ if\ o = 0$ $g(Dev_t, O_t) = 100\ if\ o = 1\ \&\ Dev_t < \lvert Tin_t - SP_t \rvert$ $g(Dev_t, O_t) = -\lvert Tin_t - SP_t \rvert^2\ if\ o = 1\ \&\ Dev_t \geq \lvert Tin_t - SP_t \rvert$
Transition-dynamics	Based on simulation and weather data file, occupancy pattern, etc.

Table 7: Underlying MDP of the HVAC use case

2.3.2.1 Optimizing hyperparameters for Proximal Policy Optimization (PPO) while applying it to the HVAC use case:

Corresponding to our work in [2] we decided to use Proximal Policy Optimization (PPO) as a candidate for policy-based RL due to its robustness against different hyper parameter settings [7]. PPO thereby is an on-policy actor-critic algorithm maintaining a parameterized policy (actor). Using its policy PPO is selecting actions based on the current state. After gathering a certain amount of samples it updates its policy by perturbing the corresponding parameters in such a way that the probability of selecting actions that led to higher rewards is increased. This is done by computing the advantage of each selected action through the critic component (i.e. value-function). Both, policy and value-function are represented as neural networks, However, as first initial tests on our simulated environment have shown that it was not easy to reproduce the results, we did further investigate the impact of the hyperparameters of the underlying algorithm and how they influence the performance of our solution approach and the overall reproducibility of our experiments. Based on [8] we decided to take the stochastic nature of the random initialization of the underlying neural network representing the value-function and policy network also into account, so that we did perform 5 runs for each experiment using the exact same list of five different seeds for the initialization process (463276947, 903450155, 1377776711, 2522572000, 3952781742). First, we tried to figure out which type of hyperparameters are responsible for the quite high variance between the different experiment runs. During this tuning we performed 150 experiments where, in each experiment, we perturbed a single hyperparameter in comparison to the default parameters and repeated each experiment five times with different seeds for initialization (Table 8). The different tested configurations (Table 22) and the results (Figure 60, Figure 61, Figure 62, Figure 63, Figure 64, Figure 65, Figure 66, Figure 67 and Figure 68) which can be found in the appendix, showed that the default parameters do indeed have an adequate performance comparing the cumulative overall reward, energy reward and thermal comfort reward, especially after the learning has converged, compared to other configurations. This aligns with the statement of the authors of PPO [7].

Parameter:	**Values:**
layer_size	64, 128
layer_numbers	2, 3, 4, 5
shared_layers	None, 1
activation_fn	tanh, relu
learning_rate	0.00025, 0.0003, 0.003
n_steps	128, 256, 512, 1024, 2048
batch_size	4, 8, 16, 32, 64
n_epochs	4, 10
gamma	0.99
gae_lambda	0.95, 0.99, 1
clip_range	0.2, 0.1, 0.02
clip_range_vf	None
ent_coef	0.0, 0.01
vf_coef	0.5
max_grad_norm	0.5
target_kl	None, 0.01

env_seed[4]	463276947, 903450155, 1377776711, 2522572000, 3952781742
reward_scaling	1, 0.01, 0.1
obs_space_norm	false, true

Table 8: Configuration space for the first round of our hyperparameter tuning

However, the variance concerning the cumulative reward of the different runs with the default parameters was quite high (variance: 1178377,3203; standard deviation: 1085,5309). Next, we decided to have a closer look at the network architecture used for the policy and value-function, assuming that the initialization of the neural networks weights mainly impacts the variance and thus the reproducibility of our experiments. For this, we performed another set of experiments where we tested different architectures for the value-function and policy. An overview of the tested architectures can be found in Table 9. During this second round we tested all 80 different combinations of the values given in Table 9 leading to a total of 400 experiments. Table 23 showing all specific architecture parameters of each configuration can be found in the appendix.

Parameter:	**Values:**
layer_size	4, 8, 16, 32
layer_number	1, 2, 3, 4, 5
shared_layers	None, 1
activation_fn	tanh, relu

Table 9: Values for architecture parameters

As can be seen in
Figure 8, Figure 9 and Figure 10, there is one architecture configuration (61) which always (throughout the whole experiment duration and after the learning has converged) performs better in terms of cumulative reward than all other architecture configurations (and especially than the default network architecture). This architecture configuration defines a network comprising only one hidden layer with 32 neurons and no shared layer. Furthermore, RelU is used as the activation function instead of the default activation function tanh. Figure 69, Figure 70, Figure 71, Figure 72, Figure 73 and Figure 74 comparing the energy reward and thermal comfort reward can be found in the appendix.

[4] These seeds are used for the generation of the probabilistic occupancy pattern of the simulation. Not to be mixed up with the seeds used for the initialization of the network weights.

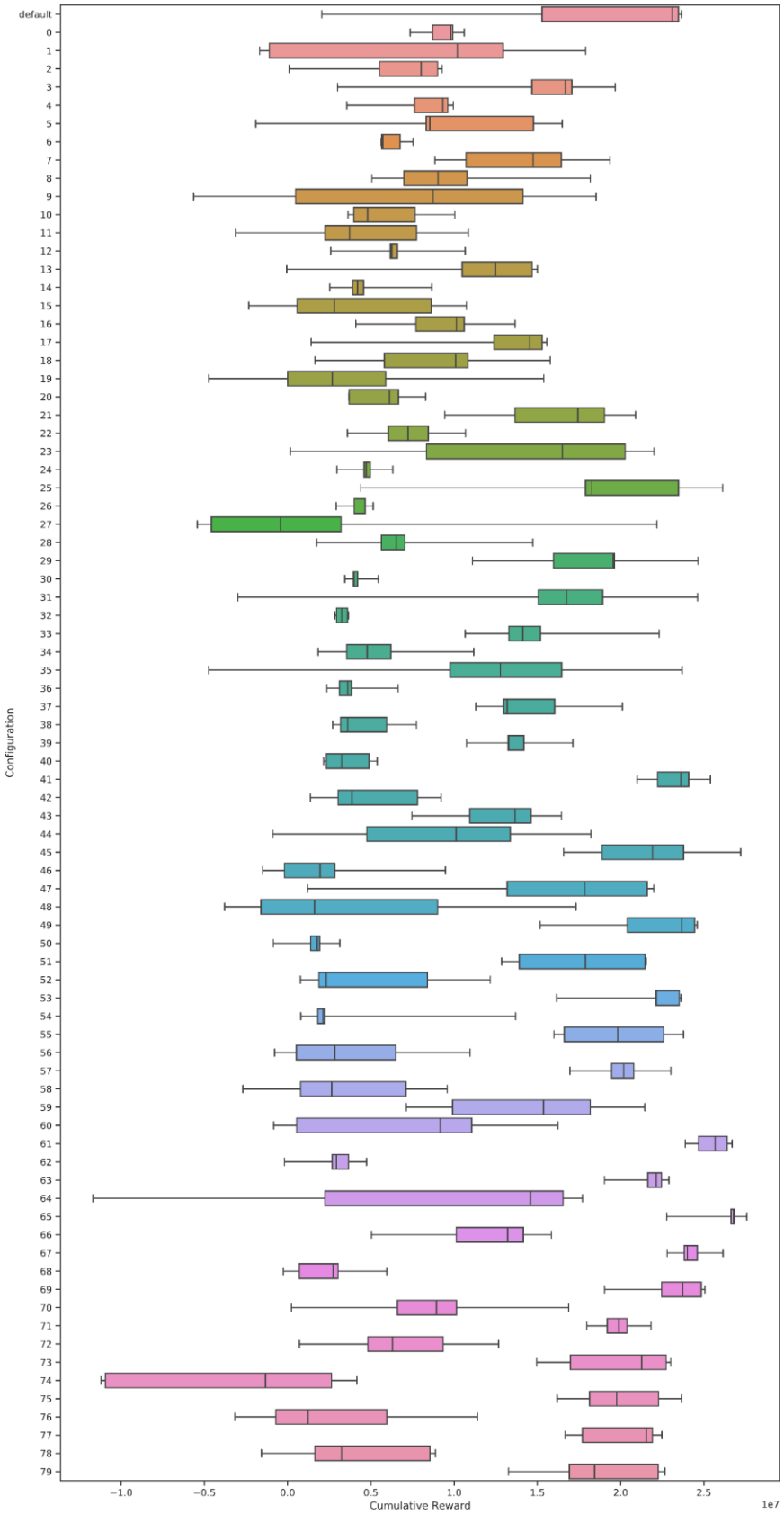


Figure 8: Cumulative Reward of different architectures

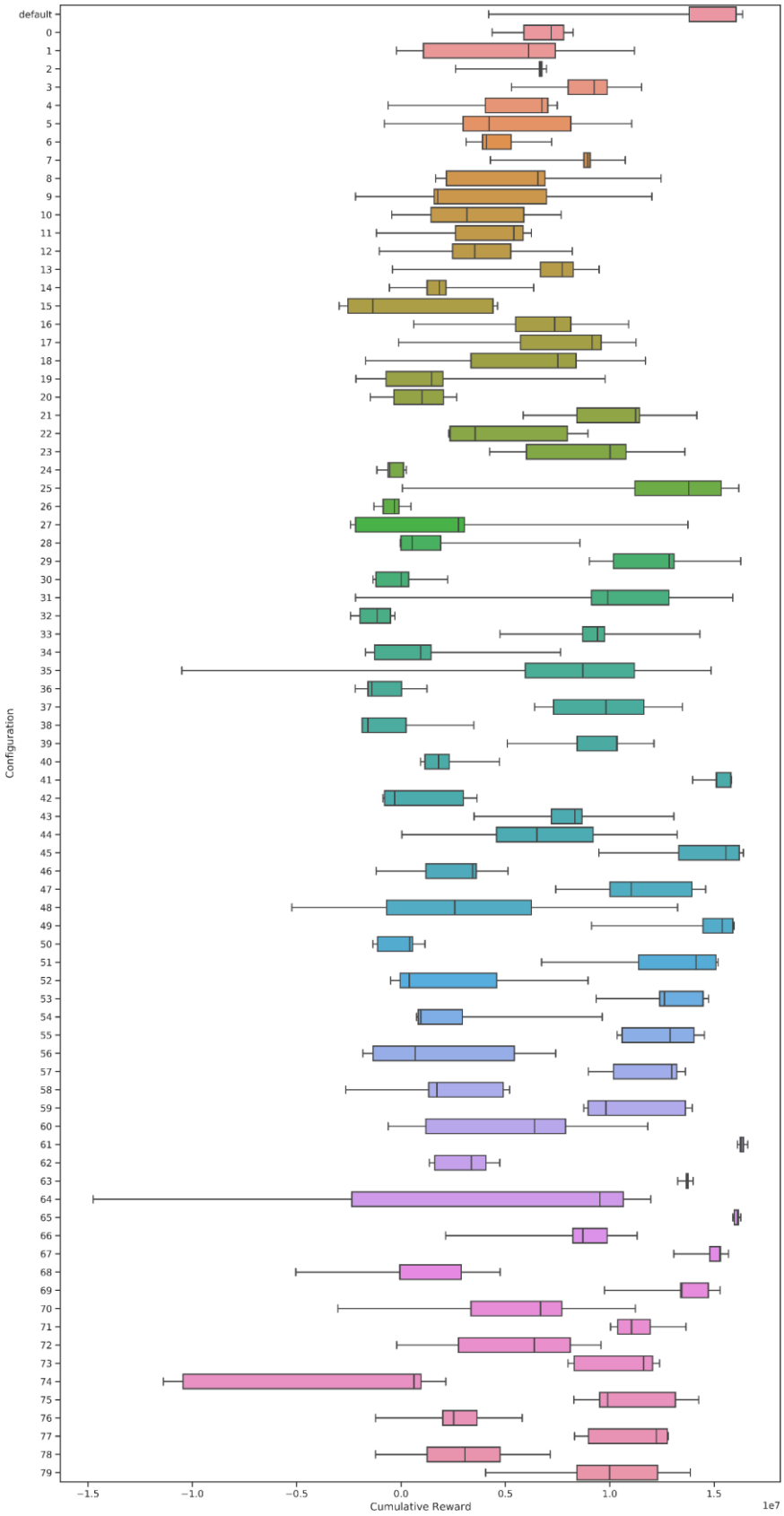


Figure 9: Cumulative Reward of different architectures (after 250.000 timesteps)

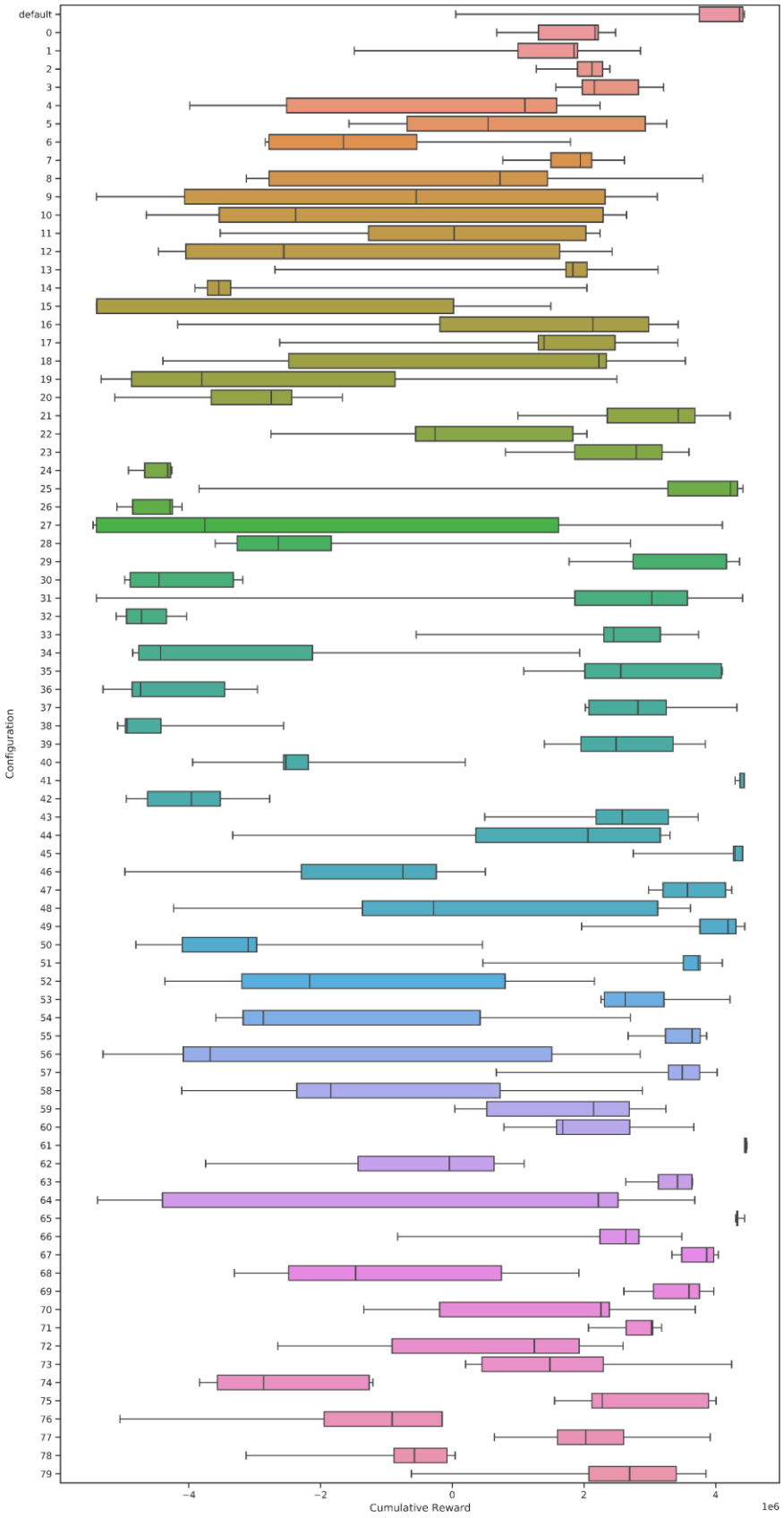


Figure 10: Cumulative Reward of different architectures (after 450.000 timesteps)

This led us to the conclusion that the authors of PPO [7] only talk about the algorithmic parameters responsible for the update of the network, but not the network parameters (responsible for the

architecture) itself. Based on our findings, we decided to stick with the default parameters concerning the PPO configuration, but used the following parameters for the network architecture instead:

Parameter:	**Values:**
layer_size	32
layer_number	1
shared_layers	None
activation_fn	relu

Remarks concerning the hyperparameter tuning:

- We used an implementation of Proximal Policy Optimization provided of the repository "stable-baselines-2" codebase (cf. [9]). We did not further investigate on other baseline-implementations as it was not our focus to benchmark different implementations. However, using a different implementation might have had influence on the performance of our approach, the presented results and consequently the choice of optimal hyperparameters.
- The hyperparameter tuning was not exhaustiive. This would have led to an immense amount of time we would have needed to finish our final experiments. As this was not feasible (and also not the main focus) during the ENACT project, we did our best to gain as much insight into the impact of the different hyperparameters on the learning performance of PPO (in our use case) as possible.

2.3.2.2 General capability of learning a HVAC control strategy with RL:

Figure 11 shows the averaged result of five experiments where we applied the policy-based RL algorithm "Proximal Policy Optimization" (PPO) to our previously defined RL-problem (HVAC use case) leveraging the implemented simulation of the KUBIK building. It can be seen that the algorithm is capable of learning an adequate control policy to control the indoor temperature. During the first 250.000 timesteps the algorithm learns that it needs to keep the indoor temperature within the setpoint boundaries when a person is present in the room in order to maximize its cumulative reward and to avoid heating or cooling actions if no person is present in the room. This can be interpreted from the increasing reward curve. After around 250.000 timesteps the reward converges which results from the algorithm now being able to properly perform the heating and cooling actions, as can be seen from the periodic spike in the indoor temperature. These spikes result from the algorithm only taking the NOP action (0) if the room is not occupied. This leads to an increase or decrease of the indoor temperature depending on the current outdoor temperature (during the winter period starting from timestep 400.000 these spikes become more visible in the plot).
Furthermore, the algorithm is able to proactively perform heating or cooling actions in such a way that the indoor temperature is already within the setpoint boundaries of +/- 1°C when a person is in the room, based on the predicted occupancy variable (non-occupancy phases 3, 4 & 5). This can be seen in Figure 12, which is an excerpt of the experiment showing the behaviour throughout one single week, after learning has converged (and without averaging the single variables). However, as the isolation of the KUBIK building is pretty efficient, the temperature is not falling below 18°C (in the excerpt). This makes no extensive preheating necessary, as only one minute of heating already increases the indoor temperature by approximately 0.5°C. It is important to note that the excerpt is based on the data of a single experiment.

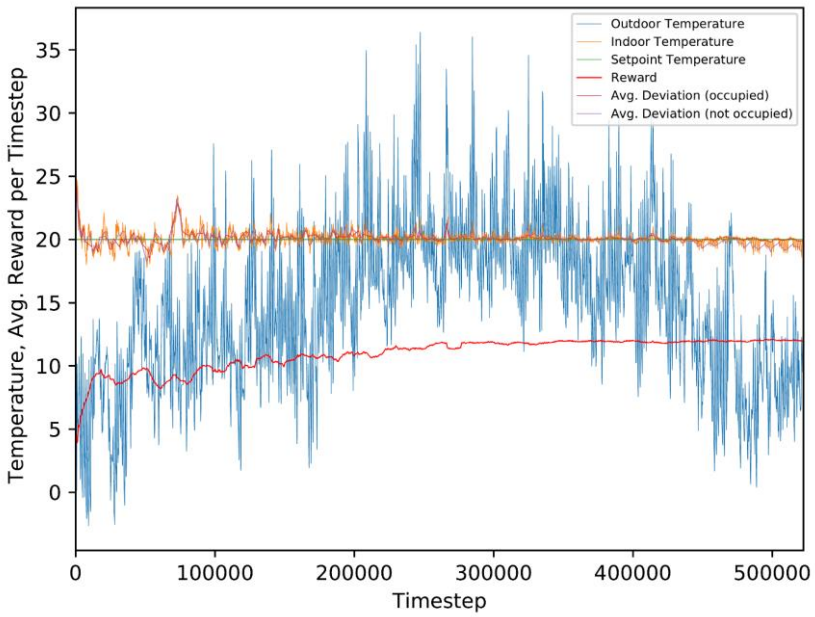


Figure 11: Evolution of temperature & reward averaged over five experiments

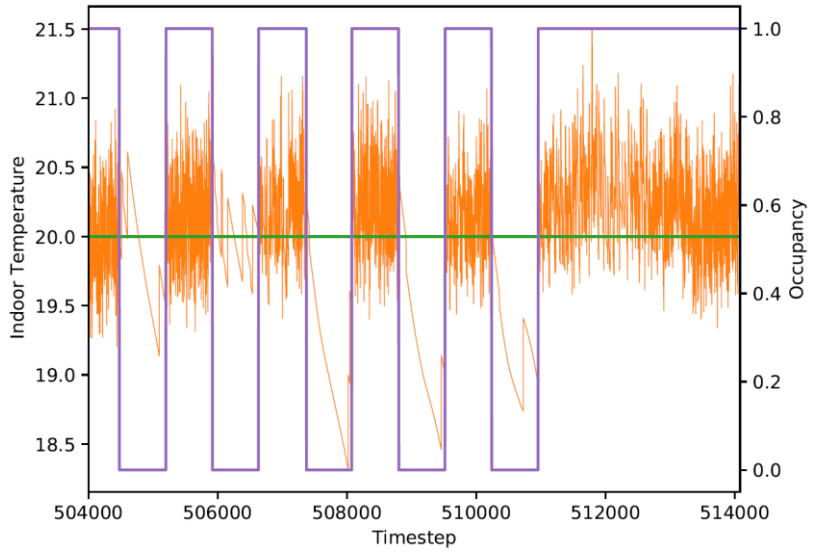


Figure 12: Excerpt of one week showing the proactive heating/cooling of a single experiment (seed: 463276947)

Results of all single experiments with their individual seeds can be found in the appendix (Figure 75, Figure 76, Figure 77, Figure 78, Figure 79). It can be seen that the learning convergence behaves slightly different according to the curves, but the overall trend of all curves shows that the underlying RL approach is capable of learning an adequate control strategy to steer the indoor temperature in such a way that the thermal comfort is maximized, while energy consumption is minimized in phases where no person is present. However, without tuning the neural network architecture, things look a lot different.
The performance of the RL-approach compared with the thermostat-baseline in terms of overall cumulative reward, cumulative energy reward and cumulative thermal comfort reward can be seen in Figure 13.

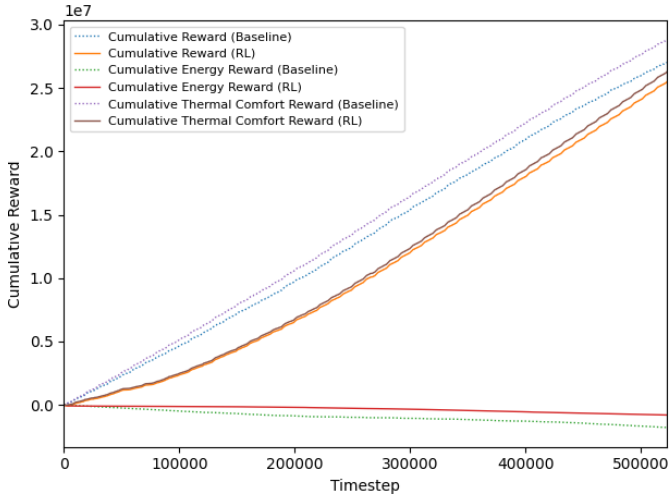


Figure 13: Comparison of overall cumulative reward (RL from scratch vs. baseline)

As the RL-approach has a learning phase throughout which it is not able to behave optimal in terms of maximizing the rewards it receives, the baseline has a better overall cumulative reward. Figure 14 and Figure 15 show the comparison of the cumulative reward after 250.000 & 450.000 timesteps, where the RL-approach has converged.

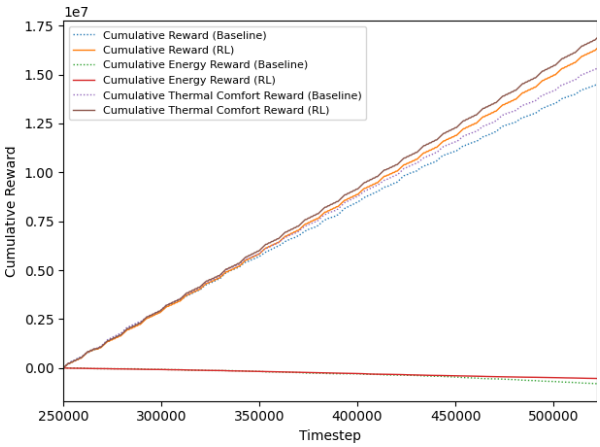


Figure 14: Comparison of cumulative reward after 250.000 timesteps (RL from scratch vs. baseline)

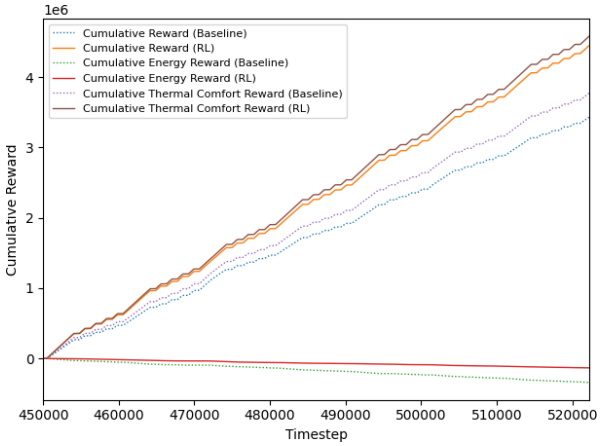


Figure 15: Comparison of cumulative reward after 450.000 timesteps (RL from scratch vs. baseline)

In these diagrams the RL-approach clearly outperforms the baseline. Same as with the temperature diagrams, plots of each single experiment compared to the baseline can be found in the appendix (Figure 80, Figure 81, Figure 82, Figure 83, Figure 84).

Table 10 gives a detailed overview of the improvements made by the RL approach, for each cumulative reward metric.

	Reward	**Reward (250.000)**	**Reward (450.000)**	**Energy**	**Energy (250.000)**	**Energy (450.000)**	**TC**	**TC (250.000)**	**TC (450.000)**
Baseline	27006777	14527144	3432345	-1777270	-801690	-342090	28784047	15328834	3774435
RL (Avg)	-5,71%	12,43%	29,72%	55,48%	33,36%	61,12%	-8,79%	10,04%	21,49%
RL1	-8,63%	10,91%	29,32%	57,46%	29,65%	60,17%	-11,64%	8,79%	21,21%
RL2	-2,26%	12,88%	30,33%	53,76%	36,25%	63,71%	-5,44%	10,31%	21,81%
RL3	-1,15%	14,31%	30,06%	50,05%	32,09%	62,84%	-4,17%	11,89%	21,64%

RL4	-4,94%	12,17%	29,34%	59,58%	37,21%	60,66%	-8,31%	9,59%	21,18%
RL5	-11,58%	11,89%	29,56%	56,56%	31,61%	58,22%	-14,35%	9,62%	21,60%

Table 10: Improvements in cumulative reward, cumulative energy reward and cumulative thermal comfort reward over baseline

2.3.2.3 Improving initial performance with pretraining MDP:

As can be seen in Figure 11 the initial performance of the control strategy leaves room for improvement. Especially when applying a RL approach from scratch in an online setting, its random behaviour (due to the need for exploration, which is one of the basic principles in RL) leads to high user discomfort and low energy efficiency in the beginning of the learning process (cf. Figure 13). To address this problem, we decided to set up a pretraining environment, which could be used for an offline training. The pretraining environment thereby leverages a different MDP, as we assume that an engineer has no complete knowledge about the final MDP of the online setting. This is also the reason why we leverage model-free RL techniques.

The pretraining MDP basically differs from the online MDP in three aspects: (1) The outdoor temperature of the simulation is following a sine wave with setpoint of 17°C, an amplitude of 10°C and a period of 1.440 timesteps (which equals 24 hours), as we assume the engineer has no prior knowledge about the location of the building where the HVAC controller is employed (and thereby only an assumption about the outdoor temperature evolution is available). (2) The occupancy pattern is the basic pattern (which is not shifted in comparison to the occupancy pattern of the online MDP). (3) The thermal equations implying the environment dynamics are simplified, by not considering the isolation etc. of the KUBIK building anymore. Instead, we set up some simple rules, to determine the indoor temperature based on the outdoor temperature and the heating or cooling action performed:

- If the outdoor temperature is higher than the indoor temperature, the indoor temperature is increased by 0.1°C;
- If the outdoor temperature is lower than the indoor temperature, the indoor temperature is decreased by 0.1°C;
- Else the indoor temperature remains constant;
- Additionally, the indoor temperature is increased by 0.5°C times the strength of the heating action (e.g. 0.5°C * 1/3 = 0.166°C) and decreased by 0.5°C times the strength of the cooling action.

All rules are defined within an indoor temperature range of -10.0°C to 50.0°C. However, these bounds are never reached during our experiments.

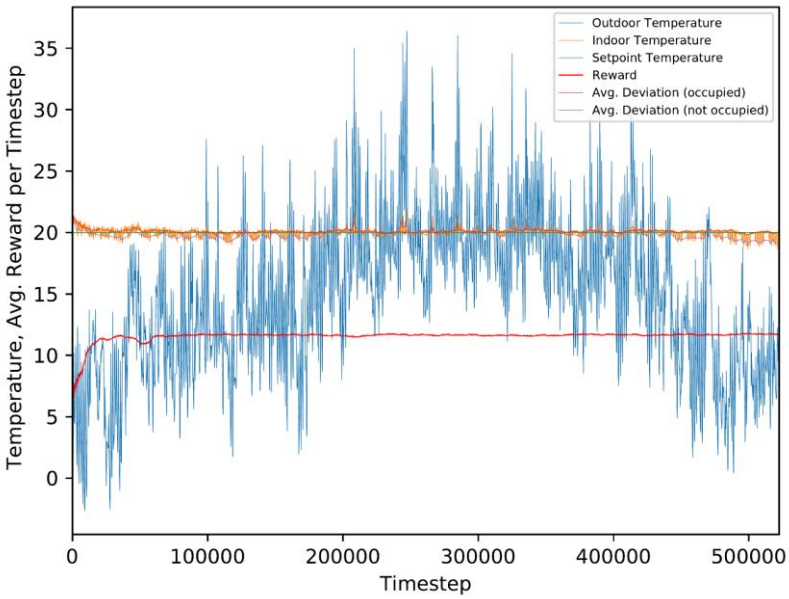
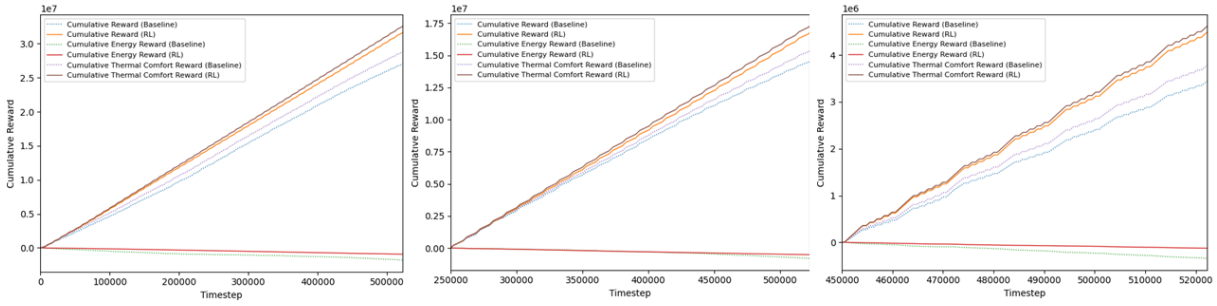


Figure 16: Evolution of temperature & reward averaged over five experiments (with pretrained model)

During a pretraining we apply the RL-approach to the pretraining MDP for 524.000 timesteps and save the learned model (value function and policy network) afterwards. After the pretraining we apply our RL-approach to the online MDP (as before) but with the weights of the value-function and policy network being initialized by the saved model of the pretraining.
As can be seen in Figure 16, the learning phase can be heavily reduced and an adequate control strategy can be observed already after 30.000 timesteps. This also leads to a convergence in the average reward after around 50.000 timesteps. The reason for a short learning phase still being present, lies in the non-stationarity of the MDP when switching from the pretraining MDP to the online MDP. As RL by its nature is able to capture such non-stationarity it still needs some time to adjust the weights of the value-function and policy network to optimize its behaviour in the new MDP. However, with the weights being already biased concerning the structure of the environment dynamics, this adjustment happens a lot faster, than adapting arbitrary weights when learning from scratch.



(a) (b) (c)

Figure 17: Comparison of overall cumulative reward (pretrained RL vs. baseline)

Figure 17 also shows this improvement in the cumulative reward earned (a) throughout the whole experiment, (b) after 250.000 timesteps and (c) after 450.000 timesteps. Due to the much shorter learning phase, now, the RL-approach is able to outperform the thermostat baseline right from start (in comparison to Figure 13). Figure 85, Figure 86, Figure 87, Figure 88 and Figure 89 show the temperature curves and Figure 90, Figure 91, Figure 92, Figure 93 and Figure 94 reward comparison for all single experiments with their individual seeds.
Finally, Table 11 gives a detailed overview of the improvements made by the RL approach with a pretrained model, for each cumulative reward metric.

	Reward	Reward (250k)	Reward (450k)	Energy	Energy (250k)	Energy (450k)	TC	TC (250k)	TC (450k)
Baseline	27006777	14527144	3432345	-1777270	-801690	-342090	28784047	15328834	3774435
RL (Avg)	-5,71%	12,43%	29,72%	55,48%	33,36%	61,12%	-8,79%	10,04%	21,49%
pRL (Avg)	17,11%	15,30%	30,74%	48,25%	35,89%	63,21%	13,07%	12,62%	22,23%
pRL1	17,68%	14,75%	30,32%	46,59%	31,71%	62,64%	13,71%	12,32%	21,90%
pRL2	17,62%	15,33%	31,04%	49,39%	38,09%	61,38%	13,48%	12,54%	22,67%
pRL3	15,50%	15,42%	30,55%	49,08%	36,73%	63,37%	11,51%	12,70%	22,04%
pRL4	18,24%	15,94%	31,37%	52,40%	42,66%	68,86%	13,88%	12,87%	22,29%
pRL5	16,51%	15,04%	30,43%	43,79%	30,28%	59,80%	12,79%	12,67%	22,25%

Table 11: Improvements in cumulative reward, cumulative energy reward and cumulative thermal comfort reward over baseline (with pretrained model)

To conclude, the experiments during the evaluation of the Online learning enabler in the smart buidling use case have shown that the it is indeed an adequate means to use Reinforcement Learning algorithms for improving the self-adaptiveness of the HVAC system in a changing context. Leveraging a proper simulation of the KUBIK building it has been shown that RL algorithms (i.e. PPO) are able to control an HVAC system in such a way that energy consumption can be reduced while thermal comfort can be increased. Furthermore, it has been shown, that this effect can be even improved when properly initializing the underlying algorithm based on a simplified version of the online environment.

2.4 Complementary results (beyond the tool)

In this section we want to briefly summarize results of complementary work (alongside the tool development) that has been done as part of Task 3.1 throughout the ENACT project.

2.4.1 Policy-based Online Reinforcement Learning Approach

As a starting point for our work on the online learning enabler we introduced an approach that uses policy-based RL as a machine learning technique to enable a software-system to adapt itself during runtime that has been published in [2]. As this has been already described in detail in D3.2 we provide just a small summary of the main ideas in this section.
By using policy-based RL we were able to overcome basically two main shortcomings arising from prior approaches leveraging value-based RL:

- Value-based RL approaches require manually fine-tuning the exploration rate, as the policy is otherwise greedily exploiting the value-function. Policy-based RL addresses this problem by using a stochastic policy leading to a probability distribution over the action space which leads to an implicit exploration,
- Value-based RL approaches may require manually quantizing environment states to foster scalability. As policy-based RL heavily rely on function approximators to represent the policy, continuous state variables can be handled directly.

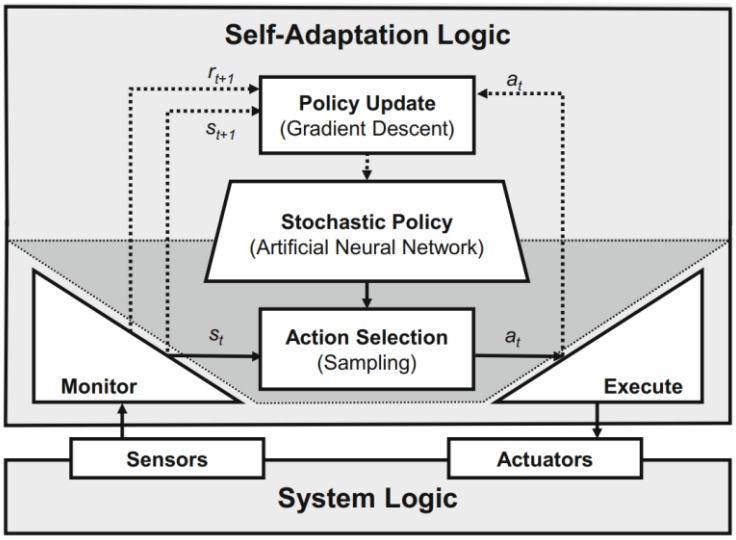


Figure 18: Conceptual architecture of OLE published in [2]

The fundamental idea behind policy-based RL is to directly use and optimize a parametrized stochastic action selection policy [10]. The action selection policy maps states to a probability distribution over the action space (i.e., set of possible actions). This means that actions are selected by sampling from this probability distribution. A learning cycle consists of a predefined number of n time steps. At the end of each learning cycle, the trajectory of n actions, states and rewards are used for a policy update. During a policy update, the policy parameters are perturbed based on the rewards received, such that the resulting probability distribution is shifted towards a direction which increases the likelihood of selecting actions leading to a higher cumulative reward.
Figure 18 depicts the conceptual architecture of our approach, showing how the elements of policy-based RL are integrated into the MAPE-K loop [11]. The dark-grey area indicates where the action selection of RL takes the place of the analyse and plan activities of MAPE-K. The learned stochastic policy takes the role of the self-adaptive system's knowledge base. At run time the policy is used by the self-adaptation logic to select (via sampling) an adaptation action at based on the current state s_t determined by the monitoring activity. Action selection determines whether there is a need for an adaptation (given the current state) and plans (i.e., selects) the respective adaptation action to execute. A policy update utilizes the trajectory of actions a_t, states s_{t+1}, and rewards r_{t+1} to update the policy. In our approach, policy updates are performed via so-called policy gradient methods [10], because the policy is represented as an artificial neural network. In our architecture, rewards are computed by the monitoring activity, as this activity has access to all sensor information collected from the system and its environment.

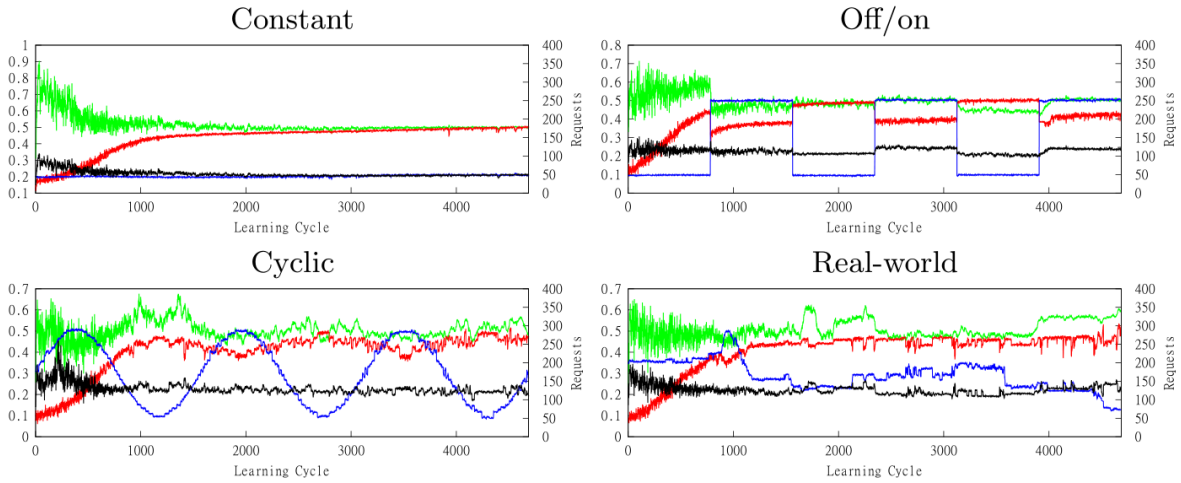
2.4.2 Application beyond IoT use cases (cloud & business process monitoring)

As part of our work to evaluate the online learning enabler, we have also considered application domains beyond IoT (i.e., beyond the use case described above). In particular, we used exemplars from cloud computing and from business process management, which we briefly described below.

Cloud. We use the auction web application Brownout-RUBiS as a subject system. When a user requests a specific item, the application's recommendation engine provides a list of recommended items based on past auctions. Due to the resource needs of the recommendation engine, Brownout-RUBiS must balance two quality requirements: maximizing the user experience by providing many recommendations, while minimizing the user-perceived latency. Therefore, the recommendation engine can be adapted by setting a so-called dimmer value $\delta \in [0,1]$, which represents the per-request probability that the recommendation engine is activated. The dimmer value thus impacts on both quality requirements: A high rate of recommendations increases user experience, but at the same time also increases resource needs and thus may increase latency. We synthesized workloads using different,

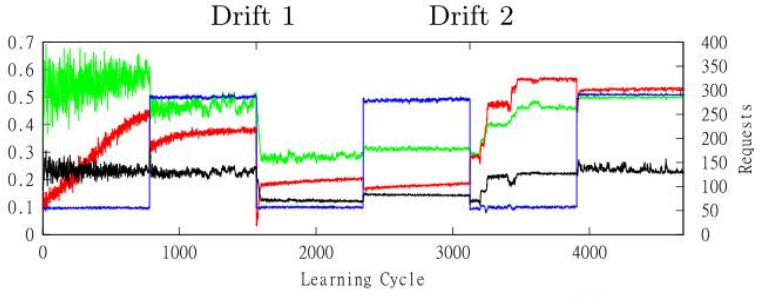
representative workload patterns from the literature: stable (constant number of requests), off/on (reflecting periodic batch processing), and cyclic (workload increases and decreases in periods). We also replayed an excerpt of a real-world workload trace.

The results, presented in [2] show how our approach enables the system to adapt itself. The system automatically adapts the dimmer value depending on the workload, thereby optimizing the balance between latency and user experience (as visible in the increase of cumulative rewards). While at the beginning of the learning process, the adaptation of the dimmer value shows a high variance for all workload patterns, after some time the variance of adaptation actions becomes visibly lower.



Learning behavior for self-adaptive web application; **blue** = workload, **black** = latency; **green** = dimmer value; **red** = reward

As shown in the figure below, our approach automatically captures non-stationary environments. After learning cycle 1562 (“Drift 1”), we reduced the virtual machine compute resources by half. This means that for the same dimmer value the system experiences a higher latency, because less compute resources are available. Our approach learns to decrease the dimmer value so that the latency threshold is not violated. After learning cycle 3125 (“Drift 2”), we increased the resources by 1.5. Again, the dimmer values are set accordingly. Note that our approach is able to capture this non-stationarity without explicitly monitoring the changes in compute resources and without explicitly changing the exploration rate.

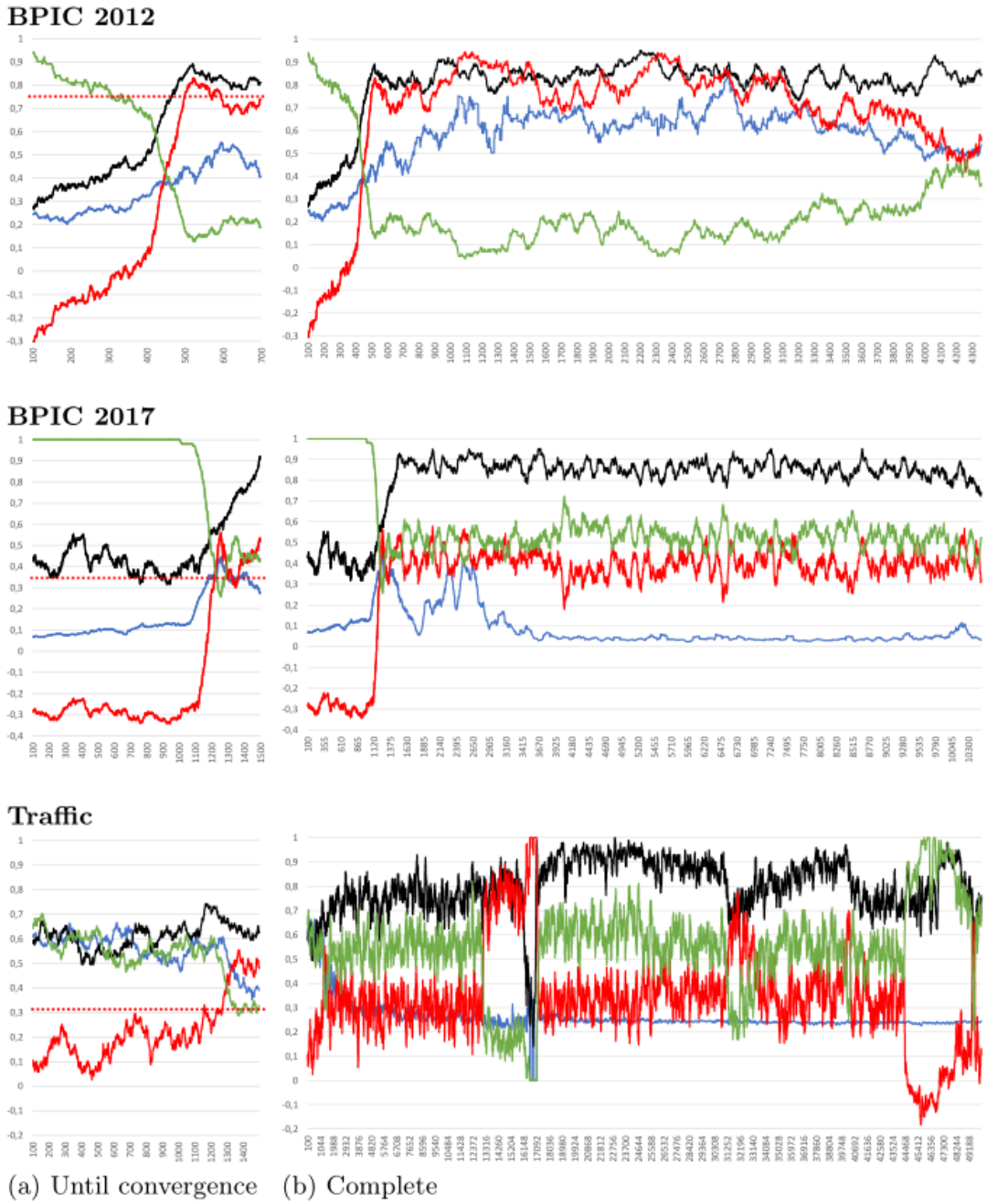


Learning behavior in non-stationary environment; **blue** = workload, **black** = latency; **green** = dimmer value; **red** = reward

Business Process Management. Proactive process adaptation can prevent and mitigate upcoming problems during process execution by using predictions about how an ongoing case will unfold. There is an important trade-off with respect to these predictions: Earlier predictions leave more time for adaptations than later predictions, but earlier predictions typically exhibit a lower accuracy than later predictions, because not much information about the ongoing case is available. An emerging solution to

address this trade-off is to continuously generate predictions and only trigger proactive adaptations when prediction reliability is greater than a predefined threshold. However, a good threshold is not known a priori. One solution is to empirically determine the threshold using a subset of the training data. While an empirical threshold may be optimal for the training data used and the given cost structure, such a threshold may not be optimal over time due to non-stationarity of process environments, data, and cost structures.

In [3] we use online reinforcement learning as an alternative solution to learn when to trigger proactive process adaptations based on the predictions and their reliability at run time. The results show that RL helps learn to find a good balance between earliness and accuracy.



(a) Until convergence (b) Complete

Learning behavior; **green**: rate of adaptations; **blue**: earliness (0 = beginning, 1 = end of process); **black**: rate of correct adaptation decisions; **red**: overall reward/100

Again, our approach is able to capture non-stationarity as is especially visible in the Traffic data set experiments, where learning captures changes in the underlying data (in this case we have identified major differences in prediction accuracy, leading to updates in the learned policy).

2.4.3 Aligning online RL with system evolution (Dev)

RL faces the exploration-exploitation dilemma. To optimize cumulative rewards, actions should be selected that have shown to be suitable, which is known as exploitation. However, to discover such actions in the first place, actions that were not selected before should be selected, which is known as exploration. How exploration happens has an impact on the performance of the learning process.

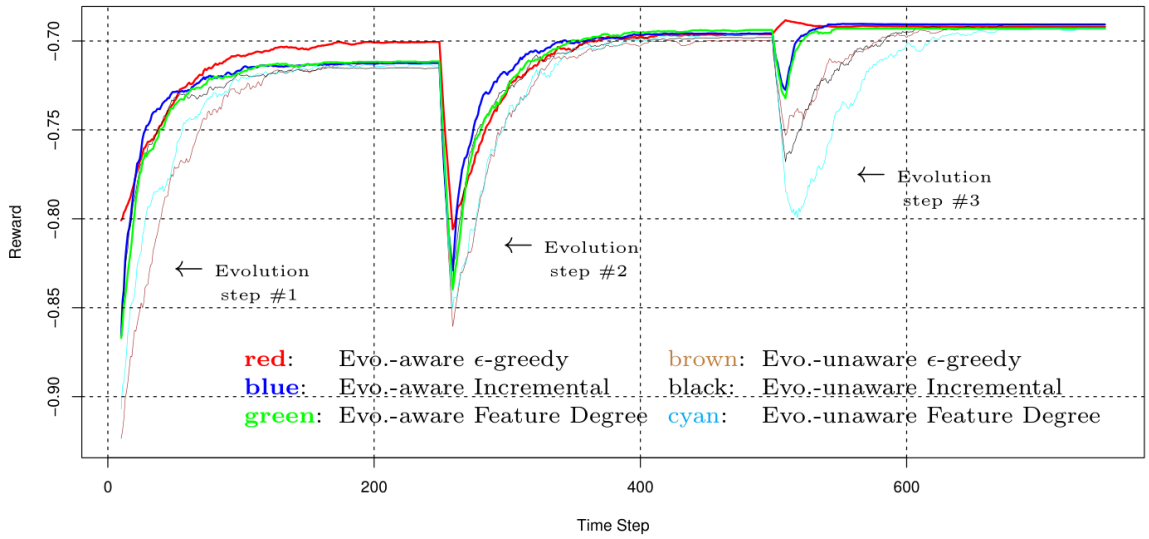
Existing online RL solutions are unaware of service evolution. They do not consider that a self-adaptive service – like any service – may undergo evolution. In contrast to self-adaptation, which refers to the automatic modification of the service by itself, evolution refers to the modification of the service by humans (in the DevOps framework, this refers to the actions performed in the DEV cycle). During evolution, service engineers may modify the service to correct bugs, remove no longer used features, or introduce new features. Service evolution means that the adaptation space may change, e.g., existing adaptation actions may be removed or new adaptation actions may be added. RL algorithms may cope with environments that change over time, so called non-stationary environments (as we have also demonstrated by using policy-based RL in the online learning tool from above).

However, a change of the adaptation space cannot be determined by observing the environment, as the adaptation space is an intrinsic property of the RL agent. As a result, existing solutions may explore new adaptation actions only with low probability (as all adaptation actions have an equal chance of being selected). It may thus take quite long until the new adaptation actions have been explored.

In [4], we introduce exploration strategies for online RL that address changes of the adaptation space due to evolution. Our exploration strategies use feature models from software product line engineering to give structure to the service's adaptation space and thereby leverage additional information to guide exploration. A feature model is a tree or a directed acyclic graph of features, organized hierarchically. An adaptation action is represented by a valid feature combination specifying the target run-time configuration of the service.

Our strategies traverse the feature model to select the next adaptation action to be explored. By leveraging the structure of the feature model, our strategies guide the exploration process. In addition, our strategies detect added and removed adaptation actions by analyzing the change of the feature model due to evolution. Adaptation actions removed as a result of evolution are no longer explored, while added adaptation actions are explored first.

As an initial proof of concept, we implemented our strategies as part of the Q-Learning RL algorithm widely used in the related work. We experimentally assess our strategies using an actual cloud resource management service and compare the learning performance with that of the widely used e-greedy random exploration strategy. Experimental results indicate an average speed-up of the learning process of 61.3% in the presence of service evolution. The improved learning performance in turn led to an average quality of service improvement of 23.7%. The figure below shows the performance of different algorithms and strategies.



Learning curves across evolution steps

2.4.4 Concept for explaining RL-based decision making

In addition to improving learning speed and asymptotic performance, the ability to explain decisions is an important goal dimension in RL. In particular, systems that influence people's lives must offer ways to explain their decisions and provide the possibility of intervention. This capability of AI algorithms is so important that the right to an explanation has even been included in the General Data Protection Regulation in 2016 [12].

Besides preventing such runtime decisions that might be unethical or even harmful, explainability is also useful in the development of RL algorithms. Here, explanation of decisions facilitates debugging, since black boxes by themselves give hardly any clues why an expected behaviour does not emerge.

In this chapter an approach is presented to explain decisions made by value-based RL algorithms. This approach builds on reward decomposition, which was utilized in the past to improve learning speed [13] [14] [15] and which will be introduced in the following section.

2.4.4.1 Reward Decomposition [14]

Reward Decomposition is a method in which several value-based RL agents are trained in parallel on different aspects of an environment. At each time step the knowledge of the agents is aggregated to provide a global decision. To apply this method, it is necessary that the reward function of the examined environment can be decomposed into independent reward channels with different semantics. As a result, instead of returning a scalar value at each time step, the reward function returns a vector where each component reflects the reward on a channel or zero if there was no reward or punishment in that channel.

For each component of the reward vector an independent subagent is trained, which receives the global state as observation and as reward only one component of the reward vector. In order to derive the action of the overall agent, at each time step the action values of all subagents are summed up element-wise and on the basis of these aggregated action values an action is selected (e.g. by using epsilon greedy action selection).

This technique can be applied to a simplified version of the HVAC environment (see Chapter 2.3) where the cooling capability was removed. In this environment the reward can be split into the two channels "thermal comfort" and "energy cost". The first of these channels always contains a negative value, i.e. a penalty if the person is present but the current temperature is not within a certain tolerance around the

desired temperature. On the other hand, the second component "energy costs" contains a constant negative value if action "heating" was chosen in the last step. One value-based RL agent (e.g. DQN) is then trained for each of these two components of the reward vector. At each time step the action values of both agents are summed up for both actions "heating" and "not heating". The greater sum then marks the greedy action of the overall agent.

2.4.4.2 Generating Explanations

To generate explanations for actions, the action-values of the subagents can be put in relation to each other. For this purpose, the difference between the action-value of a particular action and the action-values of all alternative actions is calculated for each subagent. This is repeated for all actions and yields the relative importance per action per subagent. The higher the relative importance of an action, the more influence the subagent has on the selection of this action. By observing the behaviour and relative importance the reasoning of an agent can be deduced.

To further increase the explainability of the approach, the reward function can be broken down into situations instead of channels. Each situation represents a set of special states of the environment. In addition, each situation receives a non-zero reward or punishment value that is always given when the agent is in that situation or zero otherwise. For example, the HVAC environment can be deconstructed into the following four situations:

- occupied and within the tolerance (reward: +1)
- occupied and out of tolerance (reward: -5)
- unoccupied and within the tolerance (reward: -1)
- unoccupied and out of tolerance (reward: +0.1)

For each of these situations, as before, a separate agent is trained and the relative importance is calculated. If the importance is then set in relation to the sum of the relative importance of all actions, the *relative importance of an action in relation to a specific situation* can be calculated. These values can then be summed up to obtain *the relative importance of an action across all situations*. Using these two metrics, the following natural language string can be generated for each time step:

"With my current knowledge I am 97% sure that action 'not heating' is better. Arguments in favour of action 'not heating' are the prevention of situation 'occupied and out of tolerance' (84%), the occurrence of situation 'occupied and within the tolerance' (9%), and the prevention of situation 'unoccupied and within the tolerance' (4%). An Argument in favour of action 'heating' is the occurrence of situation 'unoccupied and out of tolerance' (2%)."

The first sentence of this explanatory string contains the *relative importance of an action across all situations* (97%) and the following sentences make use of the *relative importance of an action in relation to a specific situation* (84%, 9%, 4%, and 2%).

2.5 Beyond ENACT

Based on our work in T3.1 and T3.3 we want to continue our research on the corresponding topics after the ENACT project. This future research can mainly be divided into two different streams:

- Developing a systematic approach for the offline training of self-adaptive software systems leveraging RL as their central adaptation enabler: To do this, we are currently focusing on the state-of-the-art concerning the derivation of uncertainty during the software engineering process of self-adaptive systems. By identifying the sources and types of uncertainty being relevant for RL-approaches we want to further investigate on the quantification of these uncertainties to systematically derive an offline training setup for RL-approaches.
- Extending the approach of reward decomposition to policy optimization algorithms like PPO. Thereby the main challenge is to determine the relative importance of the action of each subagent. In the case of value-based algorithms the importance is implicitly given by their action

values. In contrast, policy optimization algorithms work on the basis of function approximators, which are usually black boxes.

3 Context Monitoring and Behavioural Drift Analysis for Smart IoT Systems

The context monitoring and behavioural drift analysis enabler aims at computing a behavioural drift (i.e., the extent to which SIS behave as expected) and at supporting its analysis (i.e., understanding the symptoms of the drift and the new behaviour of the system).

3.1 Overview and Main Achievements

SIS, by interacting with the physical environment through actuators, face many challenges, pertaining to *reliability* and *resilience* requirements. These challenges are difficult to meet because the physical environment is *complex* and therefore difficult to model, limiting the ability of these systems to predict a priori the effects of their interactions. Without being able to predict the effects of their interactions at design-time, nor to formally verify their conformity to logical properties, whether functional or temporal, it is then necessary to quantitatively assess the *effectiveness* of their actions at run-time [16] (behavioural drift assessment). This assessment is based on a *systemic modelling approach;* it is carried out by confronting a behavioural model of the effects expected to be produced by the SIS in particular contexts to those observed in the field. This model is generally built from the experience of experts, documents describing standards and safety, users' preferences, etc., which are formulated by If-Then rules for their interpretation [17].

In the context of ENACT, we do consider that systemic models are implemented with Input/Output Hidden Markov Models (IOHMMs) [18]. This modelling framework brings numerous advantages in the context herein [19]; it is part of the Dynamic Bayesian Networks (DBN) family; it formalizes conditional dependencies between the effects and their stimuli (a.k.a. *contextual input*, *events*); it takes into account uncertainties, whether they are relative to the intrinsic variability of the natural phenomena observed (*randomness* through probability theory) or to the lack of knowledge on the expected effects (*incompleteness* through, for instance, the imprecise probability framework or the possibility theory). The behavioural drift is obtained by calculating the *likelihood* that the observed effects (outputs) have been generated by the model given the context (contextual inputs) i.e., that the system behaves inside the expected behaviour.

While this quantitative assessment could be leveraged as a SIS performance indicator, it does not help understanding the symptoms of the drifts which, from the systemic model perspective, can be threefold: (1) the system behaves *anomalously* outside the expected behaviour, (2) the systemic model is *incomplete*, the observed behaviour, although legitimate, has not been foreseen a priori in the model and (3), there is an *incipient drift in system parameters* i.e., the observed behaviour, although legitimate and foreseen a priori in the model, behaves slightly outside the expectations.

Whether it is due to anomalous or legitimate behaviours not foreseen a priori in the model or to system parameters drifting over time, designers must be provided with tools to support them in investigating the symptoms of the drifts in effectiveness to further change the systemic model or, for instance, trigger the Root Cause Analysis framework (cf. Section 4). By providing these tools, the Behavioural Drift Analysis (BDA) enabler addresses this problematic and contributes to the trustworthiness of SIS.

While D3.2 laid the conceptual foundations of the proposed approach, it is here formalized in a robust mathematical framework. The contribution is twofold:

1. First, **we propose a generic clustering-based algorithm for learning both IOHMMs structure (states and state-transitions) and distribution parameters.** The algorithm is proposed to be implemented with the HDBSCAN* [20] clustering algorithm and a recent incremental extension denoted by FISHDBC [21],

2. **We propose a framework for analysing symptoms of drifts in effectiveness of SIS from the systemic model perspective, which** takes place in two steps:
    a. The first step consists in generating observations from the systemic model of the effects expected to be produced by the SIS in different contexts. These observations are then fed into a learning algorithm, producing an initial IOHMM model of the expected behaviour,
    b. The second step consists in *incrementally* feeding the algorithm with observations of the SIS in operation within its environment. A second algorithm is then proposed and used to build a *directed dissimilarity graph* that allows to identify:
        i. new states i.e., states observed but not foreseen in the initial model, whether they are anomalous or legitimate,
        ii. new state-transitions and the input values (among those defined) they are triggered by,
        iii. states and state-transitions defined in the initial model yet not covered by the SIS in operation,
        iv. drifts in the initial model distribution parameters,
        v. additional metrics on the states and state-transitions are also provided (e.g., the frequency of occurrence of the states and the state transitions).

Thus, the following achievements have been realized since D3.2:

1. The proposed learning algorithm is new and now generic:
    a. it can accommodate any clustering algorithm (in D3.2, the clustering algorithm was limited to Gaussian Mixture Model clustering algorithm),
    b. it can also accommodate any systemic model of the effects expected to be produced by the SIS as far as this model is a *generative model* i.e., one can generate observations from the model,
2. The dissimilarity graph computation algorithm is new. In addition to making clear differences between the expected and the observed behaviours, it provides designers with valuable statistics (e.g., the thickness of the states and the state transitions represents their frequency of occurrence),
3. The tool allows designers to replay observations gathered from the field, making dissimilarities to clearly appear as they arrive,
4. Finally, the graphical user interface of the enabler has been enhanced to integrate the new provided functionalities.

Regarding the status of all the features to be delivered at the end of the project as described in the "Extra Document"[5] all the promised features are delivered as summarized in Table 12.

Feature	Description	Status at M22	Status at M35
DevOps features			
Specify behavioural drift analysis model	See section 3.3.1& D3.2	Completed	Completed
Generate code corresponding to specified behavioural drift analysis, to help developer deploying	See D3.2	Completed	Completed
Context aware monitoring	See D3.2	Completed	Completed

[5] "Extra Document": Plans for development and evaluation for 2020 and until the end of ENACT. Submitted to the Project Officer in January 2020 as an additional document to the planned deliverables.

Compute a behavioural drift analysis value used to monitor observed behaviour	See D3.2	Completed	Completed
Compute a behavioural drift model corresponding to the observed behaviour	See section 3.2.3	Ongoing	Completed
Compare the specified behavioural drift model with the one observed to help developers to understand the symptoms of the drift	See section 3.2.4	Ongoing	Completed
Trustworthiness features			
For resilience: observing behavioural drifts of the SIS in unexpected situations	See D3.2	Completed	Completed

Table 12: Summary of delivered features of the context monitoring and behavioural drift analysis enabler

3.2 Technical Description of Enabler

This paragraph provides the reader with the technical and conceptual explanation of the enabler. The paragraph 3.2.1 presents the conceptual overview of the architecture of the enabler. The paragraph 3.2.2 introduces the IOHMM model. The IOHMM structure and parameters learning algorithm is described in paragraph 3.2.3 and, relying on this learning algorithm, the behavioural drift analysis framework is described in paragraph 3.2.4.

3.2.1 Architecture

The Figure 19 presents a conceptual overview of the architecture of the enabler. The blue parts represent the elements of the architecture implemented at design time (Dev), those in orange, the elements implemented at run time (Ops).

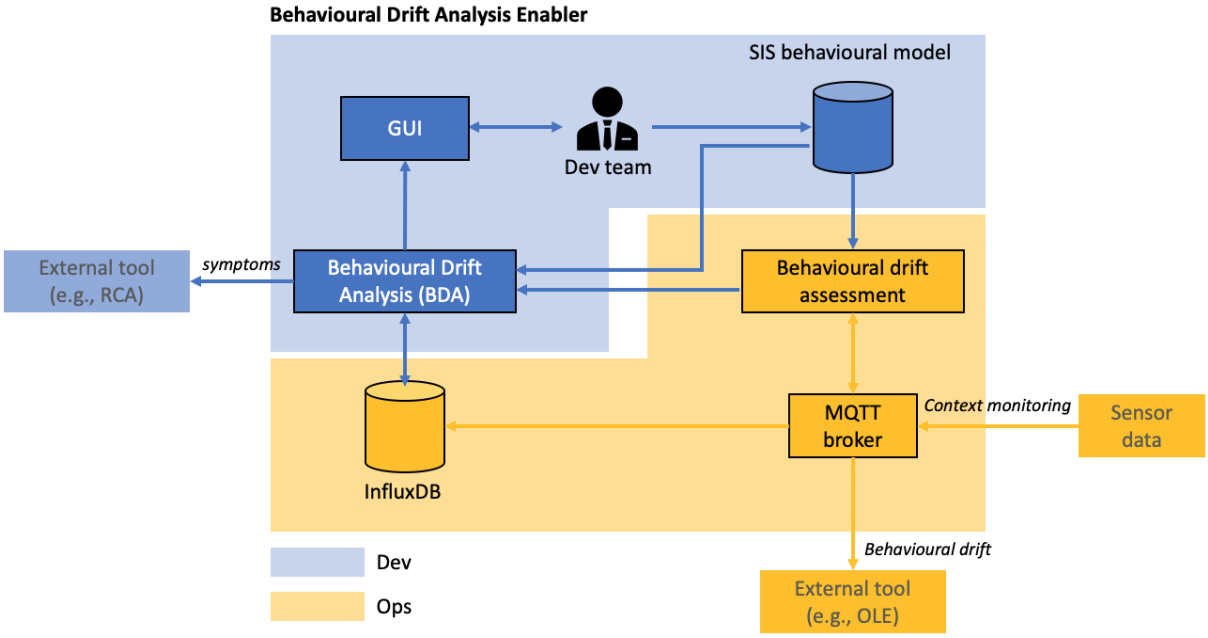


Figure 19 : Architecture of the BDA Enabler

First, the Dev team develops a model of the expected behaviour of the SIS and store this model into a database. The model corresponds to an Input/Output Hidden Markov Model (IOHMM), as described in paragraph 3.2.2 and represented through the dot[6] language. An example is given in paragraph 3.3.1. Then, this model is used at run-time to assess the effectiveness of the SIS from observations gathered

[6] https://en.wikipedia.org/wiki/DOT_(graph_description_language)

from sensors deployed in the field (context monitoring) and transmitted through a MQTT broker. Observations are recorded into an InfluxDB database. At this point, behavioural drift values can be leveraged by the OLE enabler as explained in section 6.1.
Next, any drift in effectiveness is likely to trigger the use of the BDA tool by the Dev team. The BDA tool, on the basis of the model of the expected behaviour of the SIS and the observations recorded into the InfluxDB database, computes a dissimilarity graph between the expected behaviour and the one observed in the field that leads to behavioural drift. The BDA tool and its underlying algorithms is detailed in paragraph 3.2.4. A dedicated GUI is made available to the Dev team, displaying the dissimilarity graph which makes clear the differences between the expected and the observed behaviours. An example is given in paragraph 3.3.1. Such a representation helps Dev team to study the symptoms of the drifts in effectiveness, which may trigger changes in the model of the expected behaviour or the use of the Root Cause Analysis tool described in section 6.2.

In the sequel, the models and algorithms underlying the BDA enabler are described.

3.2.2 Input/Output Hidden Markov Model

IOHMMs [18] model a pair of stochastic input/output processes $(\mathcal{U}, \mathcal{Y})$ as a result of an underlying stochastic markovian process $\mathcal{X}$ that cannot be observed (it is hidden). From a generative point of view, a sequence of input/output continuous observations $\left(\vec{u}_{(k)}, \vec{y}_{(k)}\right)_{k=1}^{K}, K \in \mathbb{N}, \vec{u}_{(k)} \in \mathbb{R}^n, \vec{y}_{(k)} \in \mathbb{R}^m$, is the outcome of a path along the states of $\mathcal{X}$; $\vec{u}_{(k)}$ and $\vec{y}_{(k)}$ are then instances of the random variables $\mathcal{U}_{(k)}$ and $\mathcal{Y}_{(k)}$ whose distributions are respectively governed by *density functions* determined by the states and the state transitions along the path.

Formally, a continuous density discrete state space (discrete time) IOHMM, is defined by the tuple $\langle Q, \vec{\pi}, A, \vec{B} \rangle$ where:

- $Q = \{x_1, x_2, \dots, x_N\}, N \in \mathbb{N}$, is the finite set of hidden states,
- $\vec{\pi} = (\pi_1, \pi_2, \dots, \pi_N)^T$ is the initial state distribution vector where $\pi_i, 1 \leq i \leq N$, denotes the likelihood of the state i to be the first state of a state sequence,
- A is the $N \times N$ state transition matrix, where each element $a_{ij}, 1 \leq i, j \leq N$, of the matrix is a n-dimensional input distribution; $a_{ij}(\vec{u}) = p\left(x_{(k+1)} = j \mid x_{(k)} = i,\ \vec{u}_{(k)} = \vec{u}\right)$ denotes the likelihood of transitioning to state $x_{(k+1)} = j$ at time $k+1$, given the current state $x_{(k)} = i$ and the contextual input vector $\vec{u}_{(k)} = \vec{u} \in \mathbb{R}^n$ at time k,
- $\vec{B} = (b_1, b_2, \dots, b_N)^T$ is the state emission vector where each element $b_i, 1 \leq i \leq N$, is a m-dimensional output distribution; $b_i(\vec{y}) = p\left(\vec{y}_{(k)} = \vec{y} \mid x_{(k)} = i\right)$ denotes the likelihood of observing the output vector $\vec{y}_{(k)} = \vec{y} \in \mathbb{R}^m$ at time k while being in the state $x_{(k)} = i$.

The structure Φ of an IOHMM is defined by the number of states N and the elements a_{ij} of the state transition matrix A such that there exists an input $\vec{u}$ that leads a transition from the state i to the state j (i.e., $\exists \vec{u}$ such that $a_{ij}(\vec{u}) > 0, \forall\ 1 \leq i, j \leq N$). The parameters $\curlywedge$ of an IOHMM correspond to the input and output distribution parameters.

This model serves as a basis for efficient solutions to several inference problems [22]:

1. **The problem of inferring the likelihood of an observation sequence (on which is based the behavioural drift assessment), solved by the Forward algorithm,**
2. The problem of inferring the most likely sequence of hidden states that led to the generation of the observation sequence, solved by the Viterbi algorithm,
3. The problem of learning the parameters of the model, solved by the Baum-Welch algorithm.

3.2.3 IOHMM structure and parameters learning

The first contribution to the enabler concerns an online IOHMM structure and parameters learning. **To date, no learning algorithm has been proposed in the literature for this model, other than those dedicated to parameter learning** (e.g., Baum-Welch).

Discrete state space IOHMMs are characterized by their ability to model stochastic processes whose states are hidden, i.e., they can only be inferred from continuous observation sequences. Learning the structure Φ and parameters λ (i.e., *identifying* the model) is first and foremost about segmenting the observation space into a finite number of relevant *regions* such that each region represents a discrete state i.e., it is assumed that observations are stochastically correlated with the system conditions, thereby taking advantage of understanding its structure [23]. A region, in this context, refers to "*a state of nature that governs the observation generation process. It can be viewed as a source of observations whose distribution is governed by a density function specific to the region*" [24].

Observation space segmentation into regions can be achieved using unsupervised *clustering algorithms*. "*These algorithms try to group observations so that the regions thereby obtained reflect the different generation processes they are governed by*" [24]. On this basis, we consider a two steps generic algorithm for learning the IOHMM structure Φ and parameters λ from a continuous observation sequence $\left(\vec{u}_{(k)}, \vec{y}_{(k)}\right)_{k=1}^{K}$.

The algorithm is described below (Figure 20):
The first step consists in segmenting the output observation space $\left(\vec{y}_{(k)}\right)_{k=1}^{K}$ into regions (discrete states in the model) using a clustering algorithm defined by the function $f_o: \mathbb{R}^m \longmapsto \mathbb{N}^*$. Each region is associated with all output observations belonging to it ($co_{(i)}$, line 2). The set of output regions CO (line 3) provides us with Q (line 4) and the elements b_i of the vector $\vec{B}$ (i.e., the output distribution parameters) are computed by fitting the most appropriate (multivariate) distribution with the set $co_{(i)}$ of output observations associated with the region i (line 7)[7].

The second step consists in segmenting the input observation space $\left(\vec{u}_{(k)}\right)_{k=1}^{K}$ into regions using a clustering algorithm defined by the function $f_i: \mathbb{R}^n \longmapsto \mathbb{N}^*$. Here again, each input observation is associated with the region it belongs to ($ci_{(i)}$, line 8). The set of input regions is defined by CI (line 9). Then, input and output observation spaces are classified according to the identified regions, i.e., each observation in the sequences $\left(\vec{u}_{(k)}\right)_{k=1}^{K}$ and $\left(\vec{y}_{(k)}\right)_{k=1}^{K}$ is associated to its corresponding region. One obtains the sequence of input regions (SI, line 10) and the sequence of states (SO, line 11) from which is built the state transition matrix A. The

```
Input  : Instances f_i and f_o of the clustering algorithm
Input  : An observation sequence (u_(p), y_(p))_{p=1}^{P}
Output : The IOHMM tuple M =< Q, A, B >
Output : The sequence SI of input clusters
Output : The sequence SO of output clusters
Output : The set CI of input clusters
Output : The set CO of output clusters
   // In the case incremental clustering algorithm is used
1  K ← f_i.GetNbElements() + P
   // 1st step : get Q and B from output space clustering
2  co_(i) ← {y ∈ (y_(k))_{k=1}^{K} | ∃i ∈ [0, K − 1], f_o.GetCluster(y) = i}
3  CO ← {co_(i) | i ∈ [0, K − 1]} // set of output clusters
   // Hidden state space Q
4  Q ← {x ∈ ℕ | 0 ≤ x ≤ |CO| − 1}
   // Set elements of the emission vector B
5  B : (b_i)_{0≤i≤|Q|−1}
6  for i in range(|Q|) do
7      B_i ← BestFit(CO_i)
   // 2nd step : get A from input space clustering
8  ci_(i) ← {u ∈ (u_(k))_{k=1}^{K} | ∃i ∈ [0, K − 1], f_i.GetCluster(u) = i}
9  CI ← {ci_(i) | i ∈ [0, K − 1]} // set of input clusters
10 SI ← (f_i(u_(k))_(k))_{k=1}^{K} // sequence of input clusters
11 SO = (f_o(y_(k))_(k))_{k=1}^{K} // sequence of output clusters
   // Set elements of the transition matrix A
   // Elements initialized to null
12 A : (a_{i,j,k})_{0≤i,j≤|Q|−1, 0≤k≤|CI|−1}
13 for i ← 2 to K do
14     λ ← BestFit(CI_{SI_(i−1)})
15     if a_{SO_(i−1), SO_(i), CI_{SI_(i−1)}} is null then
16         a_{SO_(i−1), SO_(i), CI_{SI_(i−1)}} ← λ
17 Return(Q, A, B, SI, CI, SO, CO)
```

Figure 20: IOHMM structure and parameters learning

[7] With the number of states given by Q, one can alternatively compute the parameters λ using an Expectation-Maximization algorithm (e.g., Baum-Welch), though it assumes observations whose distributions are Gaussian.

sequence of states SO determines elements a_{ij} to be populated (lines 15 and 16) with input distribution parameters (line 14).

Let us consider the example depicted in the table below:

k	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**…**
SI	1	1	1	2	2	2	2	0	0	…
SO	…	0	0	0	1	1	1	1	2	2

Output observations first belong to the state 0 (as defined for $k = 2$ in SO), then there are state transitions from the state 0 to the state 0 ($k = 2 \rightarrow 3, k = 3 \rightarrow 4$), then a state transition from the state 0 to the state 1 ($k = 4 \rightarrow 5$), then state transitions from the state 1 to the state 1 ($k = 5 \rightarrow 6, k = 6 \rightarrow 7$,...), etc. Following this sequence of states, elements a_{00}, a_{01}, a_{11}, a_{12} and a_{22} of the state transition matrix A have to be populated with input distribution parameters. Recall from the IOHMM model that the state at time k depends on the input at time $k-1$. Thus, $a_{k-1,k} \stackrel{\text{def}}{=} CI_{SI_{(k-1)}}$ i.e., $a_{k-1,k}$ contains the distribution parameters obtained by fitting the most appropriate (multivariate) distribution with the set of input observations associated with the input region $SI_{(k-1)}$ at time $k-1$. Thus, considering $k = 2 \rightarrow 3$, a_{00} contains distribution parameters associated with input observations of the input region 1 (as defined in SI at time $k = 2$), a_{01} and a_{11} contains those associated with input observations of the input region 2, etc.

The proposed algorithm accommodates any unsupervised, possibly incremental, clustering algorithm, where incremental has to be understood as the ability to process continuous observation sequences as they arrive [25].

As far as clustering is concerned, we do assume the following hypotheses:

- The amount of states $|Q|$ is not known a priori,
- The processes underlying the observations are unknown, so neither their distribution, their density, nor their law of generation (shape) are known,
- Observation could be noisy and, considering incremental learning, incomplete.

To address these hypotheses, we do consider the Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN*) clustering algorithm [20] and a recent incremental extension denoted by Flexible, Incremental, Scalable, Hierarchical Density-Based Clustering for Arbitrary Data and Distance algorithm (FISHDBC [21]). HDBSCAN* and FISHDBC are clustering algorithms for exploratory data analysis that extends DBSCAN algorithm; they can operate correctly up to 100-dimensional data[8]. The main advantages of the HDBSCAN* algorithm, in the context of the BDA enabler, lie in the fact that:

- It does not require to specify a priori the number of clusters in the data,
- It can find non-linearly separable clusters of varying shapes and densities,
- The ordering of the data does not matter,
- It supports *outlier* (or noise) assignments as being observations isolated in sparse regions.

Among the aforementioned advantages, the notion of outlier is particularly relevant when considering model learning. Let us consider two types of outliers:

- The *Intrinsic outliers* depend on how *conservative* one wants to be in learning the model, governed by the HDBSCAN* *min_samples* parameter. The larger the value of *min_samples*,

[8] https://hdbscan.readthedocs.io/en/latest/faq.html

the more conservative the clustering i.e., clusters will be restricted to progressively more dense regions with, as a consequence, a higher number of outliers[9],

- The *Extrinsic outliers* depend on the observations (noise) and are particularly interesting in the context of incremental learning. Indeed, as depicted in Figure 22, while observation sequences arrive progressively, some might reinforce the density of an existing region while some others might form a sparse region not yet dense enough to be considered as a cluster (i.e., there is a lack of knowledge leading these observations to be *temporarily* considered as outliers and discounted from the clustering process).

Whether intrinsic or extrinsic, outliers are discounted from the clustering process. Assuming that they are associated to a dummy cluster $c_{(i)}, i < 0$, the algorithm depicted in Figure 20 is modified as depicted in Figure 21.

```
1  ...
12 A : (a_{i,j,k})_{0≤i,j≤|Q|−1,0≤k≤|CI|−1}
13 for i ← 2 to K do
       // Skip outliers
14     if SO_{(i−1)} ≥ 0 and SO_{(i)} ≥ 0 and SI_{(i−1)} ≥ 0 then
15         λ ←BestFit(CI_{SI_{(i−1)}})
16         if a_{SO_{(i−1)},SO_{(i)},CI_{SI_{(i−1)}}} is null then
17             a_{SO_{(i−1)},SO_{(i)},CI_{SI_{(i−1)}}} ← λ
```

Figure 21: Outliers removal

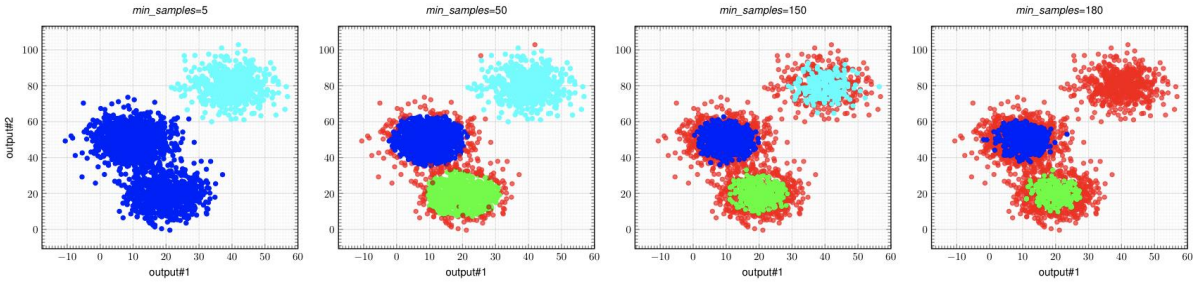


Figure 22: Example of clustering results for different values of min_samples.

3.2.4 Behavioural drift analysis framework

The second contribution to the enabler lies in the development of an algorithm that consists in identifying IOHMMs structure and parameter dissimilarities between the model learned from input/output observations gathered from the field and the model of the effects expected to be produced in different contexts. In the literature, the dissimilarity between two HMMs is mainly characterized by a *measure*. In [26], authors present a method that computes a variant of the Hellinger distance between two HMMs representing nominal and observed behaviours. In [27], authors use the Wasserstein distance. **While these quantitative assessments are useful as a performance indicator, they do not explain the structural and parametric differences between the models. Moreover, the aforementioned measures provide a global distance between the distributions of probability and, thereby, are limited to only providing dissimilarity on the parameters.**

The approach proposed for identifying both IOHMMs structure and parameter dissimilarities is based on the learning algorithm depicted in Figure 20, implemented with the FISHDBC algorithm. The main idea is first to learn an initial model from input/output observations generated from the model of the effects expected to be produced by the SIS in different contexts. Then, at some point in time, input/output observations gathered from the field are *incrementally* fed into the algorithm, leading the initial model

[9] https://hdbscan.readthedocs.io/en/latest/parameter_selection.html

to evolve over time. The following scenarios, denoting dissimilarities (symptoms) between expected and observed behaviours, are then likely to occur:

- **(Structure related)** Output observations gathered from the field lead to the creation of new clusters. These new clusters define new states (either legitimate or anomalous) not foreseen in the initial model,
- (**Structure related**) Input observations gathered from the field lead to the occurrence of new state transitions (either legitimate or anomalous) not foreseen in the initial model,
- **(Parameters related)** Input and output observations gathered from the field lead distribution parameters (e.g., mean & standard deviation) to be slightly modified, denoting a drift in the initial parameter values,
- **(Structure & Parameters related)** observations gathered from the field do not cover the states and state transitions defined in the initial model. This situation implies that either not enough observations have been gathered from the field (e.g., rare events) or that the initial model is somehow wrong in the sense that it expects a different behaviour than the one just observed.

IOHMMs being graphical models, one can easily depict graphically the aforementioned identified dissimilarities, thereby helping designers in investigating on the changes. In this context, the algorithm depicted in Figure 23 and Figure 25 builds a dissimilarity graph as follows:

(lines 3 and 4) First, using the online learning algorithm depicted in Figure 20, implemented with FISHDBC clustering algorithm, an initial model is learned from observations generated from the systemic model of the effects expected to be produced in different contexts. The initial model is then incremented with observations gathered from the field.

(line 5) On the basis of the sets of clusters CI and CO and the sequences of clusters SI and SO, obtained from the previous step, one can compute, using the algorithm depicted in Figure 24, the vector $\vec{B}_w$ and the matrix A_w of the frequency of occurrence of each state and state transitions respectively.

```
Input  : Expected obs. sequence (u⃗_(v), y⃗_(v))_{v=1}^{V}
Input  : From the field obs. sequence (u⃗_(w), y⃗_(w))_{w=1}^{W}
Input  : An instance of a graph data structure G
Output : The dissimilarity graph
   // Instances of the clustering engine
1  f_i ←FISHDBC(min_samples=n)
2  f_o ←FISHDBC(min_samples=n)
   // IOHMM corresponding to the expected behaviour
   // (See algorithm 1)
3  Q, A, B⃗, SI, CI, SO, CO ←Learn(f_i, f_o, (u⃗_(v), y⃗_(v))_{v=1}^{V})
   // Increment f_i and f_o with observations from the field
4  Q, A, B⃗, SI, CI, SO, CO ←Learn(f_i, f_o, (u⃗_(w), y⃗_(w))_{w=1}^{W})
5  A_w, B⃗_w ← GetWeights(SI,CI,SO) // (See algorithm 4)
   // COMPUTE DISSIMILARITY GRAPH NODES
   // Obs. from the field start at index V + 1
6  for i ← V + 1 to len(SO) do
7      if SO_(i) ≥ 0 then
8          if not G.NodeExists(SO_(i)) then
9              if SO_(i) in range(max((SO_(i))_{i=1}^{V})) then
                   // This state is expected
10                 G.AddNode(SO_(i), label=B⃗_{SO_(i)}, line width
                     = B⃗_{wSO_(i)}, color=blue)
11             else
                   // This state is not expected
12                 G.AddNode(SO_(i), label=B⃗_{SO_(i)}, line width
                     = B⃗_{wSO_(i)} color=red)
13 for i ← 1 to V do
14     if SO_(i) not in range(max((SO_(i))_{i=V+1}^{V+W+1})) then
           // This expected state has not been covered
15         if not G.NodeExists(SO_(i)) then
16             G.AddNode(SO_(i), label=B⃗_{SO_(i)}, line width =
                 B⃗_{wSO_(i)}, color=orange)
```

Figure 23: Dissimilarity graph computation (part#1/2)

(lines 6 to 16) The first V elements of SO correspond to the expected states computed from $\left(\vec{y}_{(v)}\right)_{v=1}^{V}$. The remaining $V + W + 1$ elements correspond to the states obtained from output observations $\left(\vec{y}_{(w)}\right)_{w=1}^{W}$. gathered from the field. The idea is to parse states associated to the latter and verify if they are present or not in the former. A node is created in the dissimilarity graph for each state whose colour depends on whether the state is present (blue, meaning the state is expected) or not (red, the state is not expected). The reverse process is done (lines 13 to 16) to detect states that are present in the former but not present in the latter (orange, an expected state is not covered). The width of the states depends on their frequency of occurrence given by $\vec{B}_w$.

(lines 17 to 21) This part of the algorithm is devoted to compute the transition matrix A' corresponding to the first V elements of SI and SO associated to the expected behaviour.

(lines 22 to 30) Transition matrices A and A' are compared for dissimilarities. Edges are added into the dissimilarity

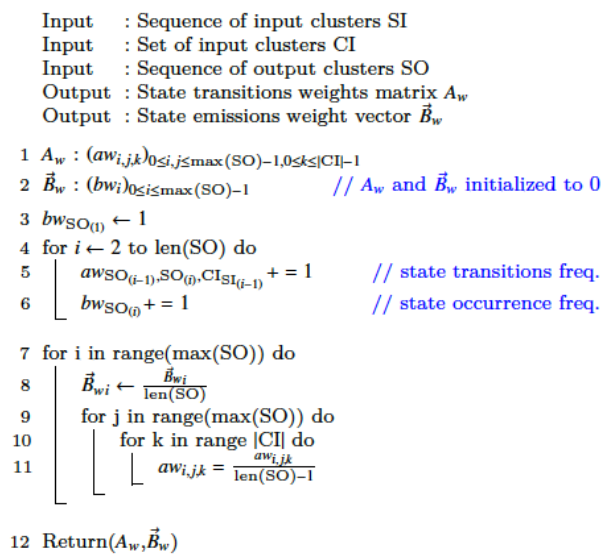


Figure 24: States and state-transitions weights computation

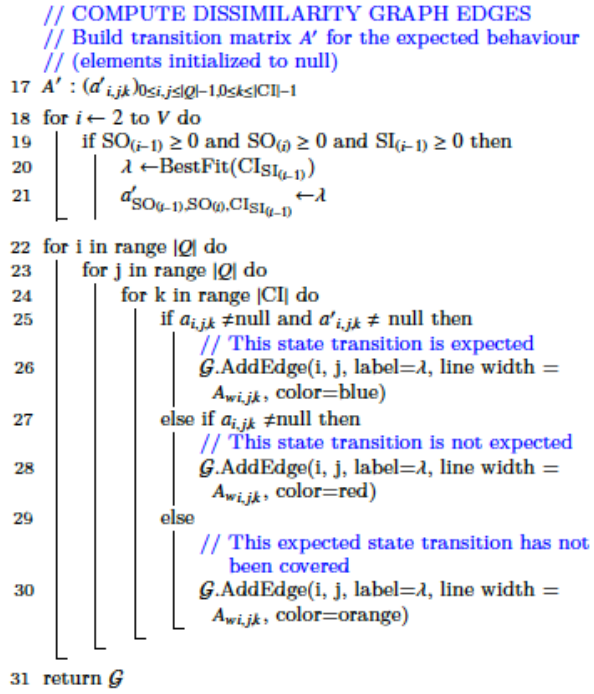


Figure 25: Dissimilarity graph computation (part#2/2)

graph for each state transition whose colour depends on whether an expected (blue) or unexpected (red) state transition occurred. Expected state transition not covered are associated to edges coloured in orange in the dissimilarity graph. The width of the state transitions depends on their frequency of occurrence given by A_w.

3.3 Evaluation

The principles of the proposed approach having been laid, this paragraph is devoted to the description and the validation of the behavioural drift analysis tool implemented into the project. First, in Section 3.3.1 the tool is introduced through a simple synthetic example. Then, in Section 3.3.2 we present the results of drift analysis obtained on a more complex real data set.

3.3.1 Presentation of the tool on a synthetic use-case

First experimentation is carried out on a synthetic use-case reproducing a smart-home scenario. The scenario considers a SIS whose purpose is to maintain a certain level of luminosity that depends on the presence or not of a person in the room.

To start with, the model of the expected effects to be produced in different contexts has to be provided to the tool. It is based on the dot language[10], a textual graph description language. This format is largely supported by WYSIWYG editor tools (e.g., edotor[11], GraphViz[12]). The model corresponding to the aforementioned scenario is given below:

```
digraph {
        // States where physical effects to be produced is relative to the
        // luminosity level
        0 [label="State#0 lum->['100.0, '3.5']"] //<feature>->[<mean>,<stdev>]
        1 [label="State#1 lum->['0.0' , '3.8']"]
```

[10] https://en.wikipedia.org/wiki/DOT_(graph_description_language)
[11] https://edotor.net
[12] https://dreampuf.github.io/GraphvizOnline/

```
        // State transitions triggered on a presence sensor value
        0 -> 0 [label="pres->['20.0', '3.5']"]
        0 -> 1 [label="pres->['0.0' , '3.5']"]
        1 -> 0 [label="pres->['20.0', '3.5']"]
        1 -> 1 [label="pres->['0.0',  '3.5']"]
}
```

Besides this model, the tool expects the data gathered from the field corresponding to input/output observations of the SIS operating in its environment, corresponding to a period in which a behavioural drift was observed. Data are provided as a comma separated file (csv) retrieved, for instance, from a database. An example is given here below.

```
lum;pres
0.926894614;1.914098955
8.205926368;-1.822739541
-3.058076756;-0.694019514
2.677860266;-2.295476298
2.550909248;3.05934991
…
```

Once these inputs have been entered into the tool (see the two fields 'Load dot' and 'Load observations' on top of Figure 26), the analysis can be started (see the button 'Analyze' on the top right of the Figure 26). Following the behavioural drift analysis framework described in Section 3.2.4, the first step is to learn an initial IOHMM model from input/output observations corresponding to the expected behaviour; the first part of the dataset depicted in Figure 26 (from observation 0 to 370) corresponds to the expected luminosity level (output) to be produced depending on the presence or not of a person in the room (contextual input). The data is here automatically produced from the dot file. The systemic model, learned from these observations, is the one depicted in Figure 26.

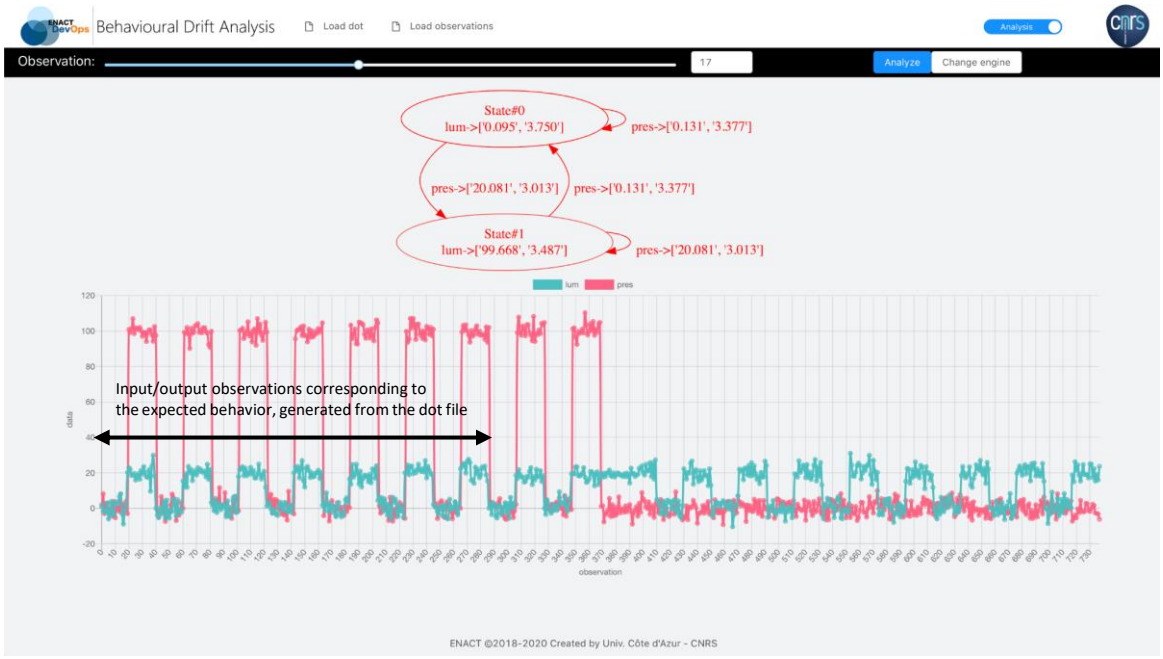


Figure 26: A model of the expected effects to be produced by the SIS is first learned from observations generated from the dot file and feed into the algorithm depicted in Figure 20.

Then, from observation 371 the initial model learned is incrementally complemented with the behaviour of the SIS observed from the field, provided in the csv file, characterized by observations #371 to #730 in Figure 26. The model is updated as new observations are presented to the learning algorithm. A slider makes it possible to replay observations.

Using the dissimilarity graph algorithm depicted in Figure 23 and Figure 25, differences in the behaviour (symptoms) are presented to the designers (see Figure 27). States and state transitions coloured in green characterize the behaviour of the SIS observed from the field that corresponds to the expected behaviour. Any differences between the expected behaviour of the SIS and the one observed from the field are coloured in red. **Here, the symptom highlighted implies that the light is broken; whatever the presence in the room, the luminosity level remains low.**

Figure 27: Differences between the behaviour of the SIS observed from the field and the one expected are depicted in red.

This makes the tool an ideal companion of the RCA enabler described in Section 4. By providing designers with symptoms of drifts in effectiveness, it may help them to better understand their root-causes and implement mitigating actions.

3.3.2 Evaluation on a real use-case

The second experimentation is carried out on a real dataset gathered in a smart home. It is meant to highlight the mechanics of the clustering algorithm underlying the IOHMM learning algorithm; thus, the results are depicted outside the tool presented in the previous section.

The objective is to learn a behavioural model of a set of devices from sound features. The devices considered for this experiment are a TV and the Amazon Echo smart speaker. While both devices can produce sound into the environment, the expected behaviour, from user's perspective, is that they do not produce it simultaneously. In this example, the initial model is learned directly from observations gathered from the environment (i.e., not generated from a dot file) and provided to the learning and dissimilarity graph algorithms through the MQTT protocol.

Output observations (expected effects) are characterized by two standard sound features extracted from a micro-phone signal i.e., the Mel-Frequency Cepstral Coefficients[13] (MFCCs) and the Zero Crossing Rate[14] (ZCR) sound features. These features have been selected for their capacity at obtaining pairwise distinct clusters for each device for each context. Contextual input observations are characterized by the operating status of the TV and the Amazon Echo smart speaker.

[13] https://en.wikipedia.org/wiki/Mel-frequency_cepstrum
[14] https://en.wikipedia.org/wiki/Zero-crossing_rate

The dataset depicted in Figure 28 represents the expected behaviour of both devices operating together in the environment. It is fed to the learning algorithm depicted in Figure 20 and the resulting initial IOHMM model is depicted in Figure 29 where distributions are fitted to Gaussian Mixture Models (GMMs), characterized by the mean and the standard deviation of each feature (i.e., `<feature>->[<mean>,<stdev>]`). Colours of states and state transitions correspond their associated clusters as depicted in Figure 28 (observations depicted in red are outliers).

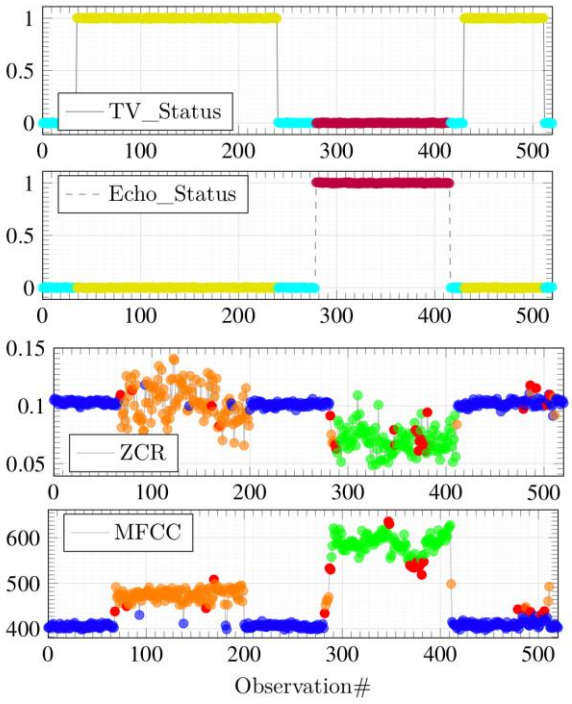


Figure 28: Real dataset gathered from a smart-home. Observations correspond to the expected behavior and are coloured based of their associated cluster.

On the basis of the output observations (ZCR and MFCC), three states are identified as depicted in Figure 29. The blue one corresponds to the silence, the orange characterizes the sound emitted by the TV and the green one characterizes the sound emitted by the Amazon Echo smart player (here, playing music). Input observations (TV_Status and Echo_Status) lead three clusters to be identified, corresponding to different configurations of the devices. The one in cyan corresponds to the configuration where none of the devices is in operation, the one in yellow to the configuration where only the TV is in operation and the one in purple where only the Amazon Echo is in operation, playing music. As expected, the devices do not operate simultaneously. The model thus learned corresponds to the expected behaviour (Figure 29).

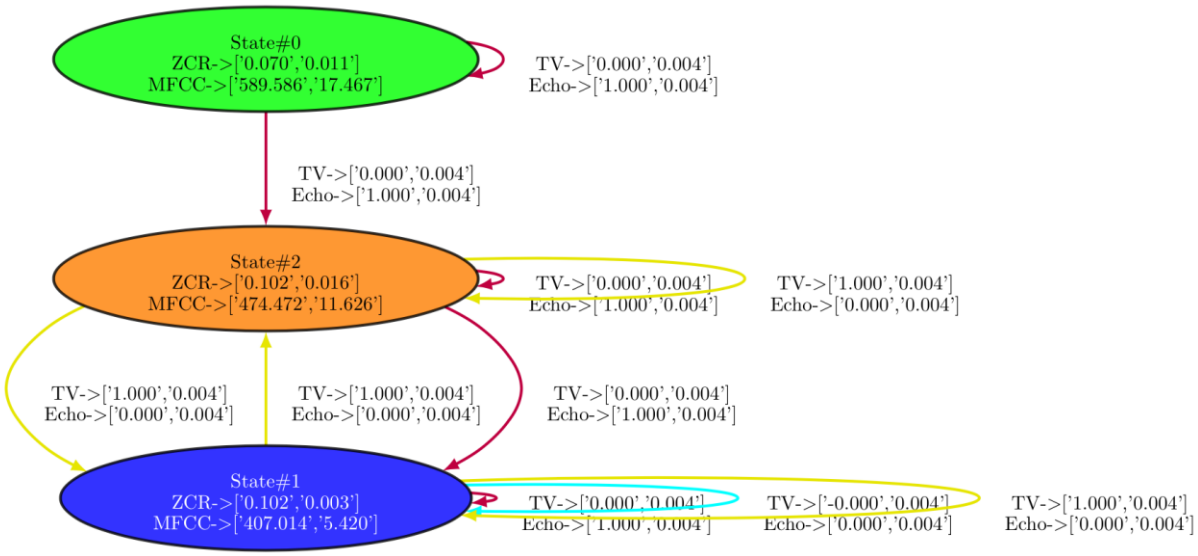


Figure 29: IOHMM model learned from the input/output observations depicted in Figure 28, applied to algorithm depicted in Figure 20.

The data is further complemented with longer run observations gathered from the same physical environment (Figure 30). This dataset highlights two unanticipated behaviours (denoted in magenta) that will have to be identified by the dissimilarity graph algorithm:

1. The first unforeseen behaviour concerns a configuration on the TV and the Amazon Echo smart speaker, both operating simultaneously (Figure 30, input features from observation#80 to #230). This behaviour is anomalous, both devices must not operate simultaneously.
2. The second unforeseen behaviour relates to a new device that has appeared in the environment. This device is an autonomous smart vacuum cleaner which the inhabitants have recently acquired (Figure 30, output features from observation #280 to #430, then from observation #520 to #610 and, finally, from observation #800 to #1000). This unforeseen behaviour is not anomalous, the model of the expected behaviour has to be updated according to this new device.

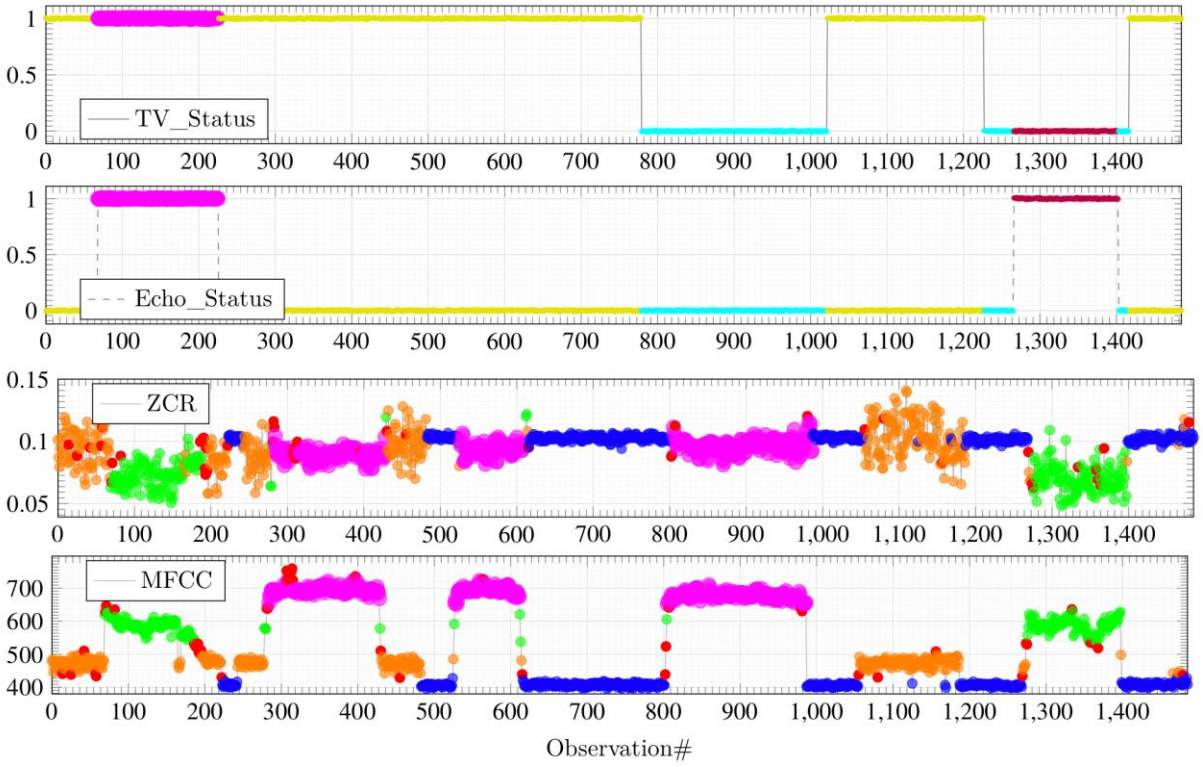


Figure 30: Observations from the field, coloured by the clusters identified by the FISHDBC algorithm. Observations coloured in magenta denote behaviours not foreseen in the model of the expected behaviour previously learned.

The dataset is applied to the dissimilarity graph algorithm depicted in Figure 23 and Figure 25. The resulting dissimilarity graph is depicted in Figure 31.

Observations from the field whose behaviour correspond to the expected behaviour lead blue states and state transitions to appear while those whose behaviour do not correspond to the expected behaviour lead red states and state transitions to appear.

On the basis of this graph, designers can better understand the symptoms of what went wrong with the system and/or the model of the expected behaviour; the red state characterizes the behaviour of an autonomous vacuum cleaner in operation, denoting an issue with the model as this device was not initially foreseen. Additional state transitions occur, denoting unforeseen legitimate and anomalous behaviours (e.g., having both the TV and the Amazon Echo smart speaker operating simultaneously).

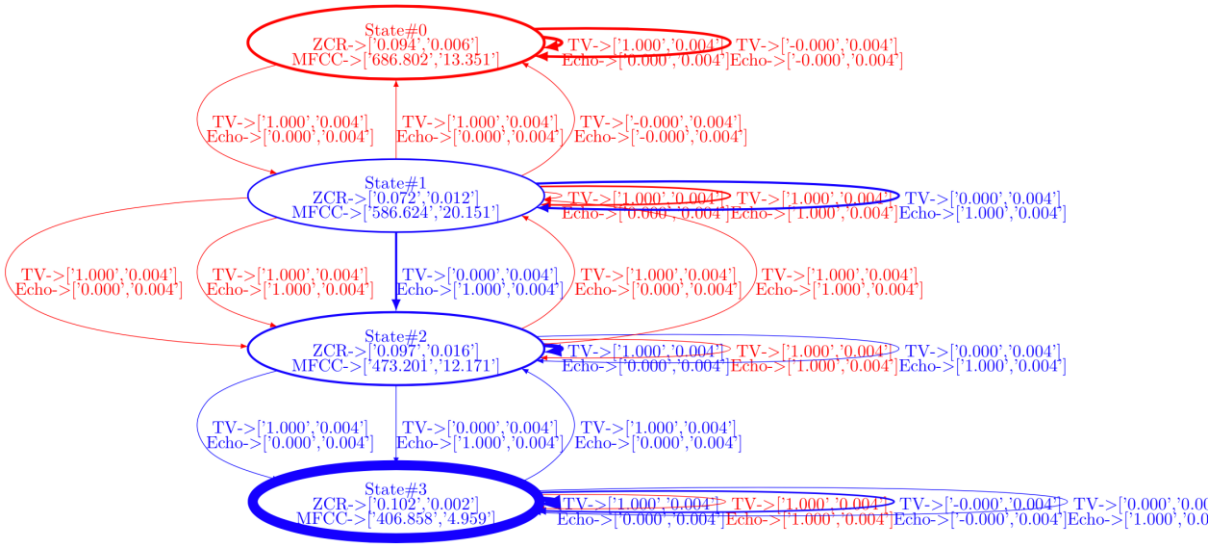


Figure 31: Dissimilarity graph between the expected behaviour whose model is depicted in Figure 29 and the behaviour observed from the field, depicted in Figure 30.

As depicted throughout these examples, the BDA enabler makes clear the differences between the expected behaviour and the one observed in the field. It helps designers to investigate the symptoms of behavioural drifts and correlate various events to, for instance, determine their root causes, making it an ideal companion for the RCA enabler described in Section 4.

3.4 Beyond ENACT

The BDA enabler, beyond the ENACT project, might be used to address concerns such as systems health monitoring, fault detection and analysis, fault diagnosis, etc. At the heart of the enabler, the newly developed clustering-based Input/Output Hidden Markov Model (IOHMM) learning algorithm might be leveraged, beyond the BDA enabler, for learning any behavioural models whose dynamics depends on some contextual inputs. The algorithm is generic and can accommodate any underlying clustering algorithm making it flexible and open to future evolutions. Among these evolutions, it is planned to extend the algorithm to handle temporal constraints where state transitions also depend on the time spent in each state (Input/Output Hidden Semi-Markov Model).

4 Root-Cause Analysis for Smart IoT Systems

4.1 Overview and Main Achievements

Root Cause Analysis (RCA) is a systematic process for identifying "root causes" of problems or events and an approach for responding to them. RCA is based on the idea that the effective management requires more than merely "putting out fires" when problems are detected, but also finding ways to prevent them. RCA is an important part of Risk Management which consists of vulnerability scanning, monitoring & anomaly detection, RCA, and reaction/ remediation. Especially in complex systems (e.g., Big IoT systems), Risk Management becomes a very difficult and time-consuming task. RCA helps facilitating this task of the system's administrators and the DevOps engineers, so that they can have more hints to respond faster to the incidents.

In the context of ENACT, the RCA Enabler relies on machine learning algorithms to identify the most probable cause(s) of detected anomalies based on the knowledge of similar observed ones. This is performed by assess the similarity between the symptoms of new problems with the known one. Figure 32 demonstrates the high-level architecture of the implemented enabler.

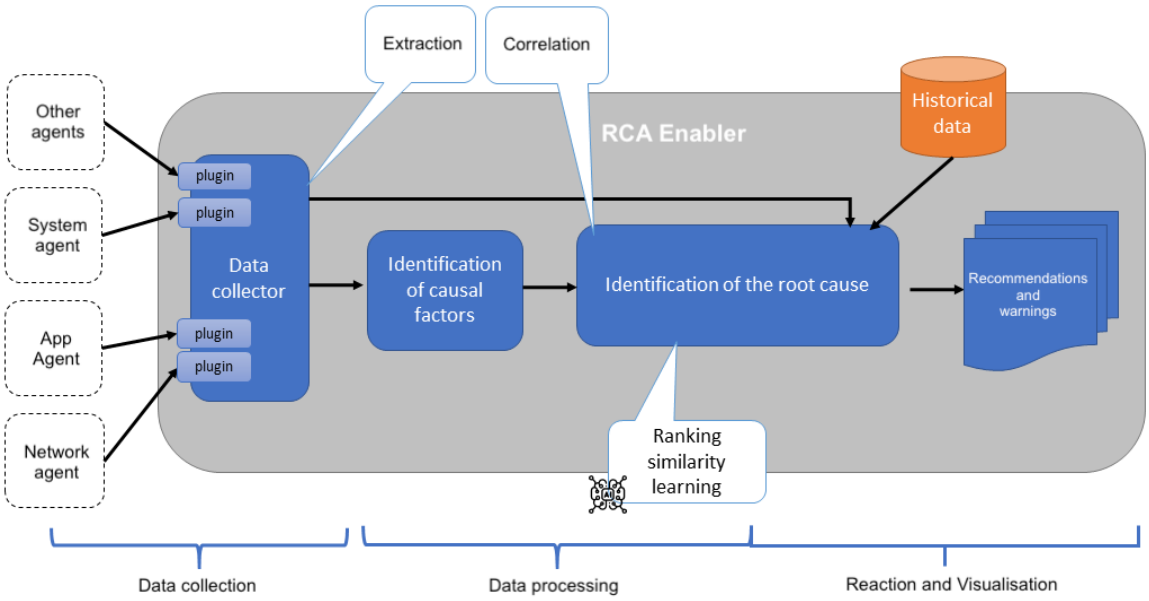


Figure 32: RCA Enabler high level architecture

The **data collector** allows gathering information from different sources (e.g., network, application, system, hardware) by relying on dedicated monitoring agents. It has a plugin architecture that enables its extension to new formats of data. The parsing of such data allows extracting numerous attribute values that can be relevant for the identification of the origin of detected incidents. The selection of the most relevant attributes is performed based on several machine learning algorithms (further discussed in Section 4.2.2.1) in order to increase the accuracy of the analysis and reduce the data dimensions as well as the computations needed.
The **historical data** is a set of information used for learning purposes. They are labelled events collected overtime to describe the origin cause of several incidents as well as the relative attributes values. This data is constructed by two means:

- *Active learning:* By actively performing different tests including the injection of known failures and attacks, the collected data can be easily labelled since we deal with a controlled system.
- *Passive learning:* Once an incident is detected without knowing its origin, with the aid of the system experts, classical RCA is performed by debugging different logs and correlating various events to determine the corresponding root causes. The result of this task can be stored in the database with its relevant attributes values. The detections of the incidents could be performed

by the BDA (further discussed in the Section 6.2) and the Security and Privacy Monitoring Enabler (WP4).

Based on these two inputs (collected new data and historical data), the idea is to determine when the system reaches a known undesirable state with a known cause. This involves using the concept of Similarity Learning and, more specifically, "Ranking Similarity Learning" [28]. That means the tool calculates the "similarity" of a new state with the known ones and then presents a list of the most similar states with the relative order of similarity. The final goal is to recognise the root origin of an incident and based on the set of known mitigation strategies based on the experience reflected in the historical data. In this way, the tool can recommend to perform the relevant countermeasures.

The final version of the Root Cause Analysis (RCA) enabler comes with the following main contributions:

- Conceptual approaches to collect different system traces and to extract potentially relevant data attributes to build the labelled training dataset.
- The techniques to evaluate the importance of the attributes and to select the most relevant ones based on different Machine Learning techniques.
- A knowledge-based approach to identify the root cause of detected incidents by analyzing the symptoms and calculating the similarity of new events with the learned ones in the historical data.

Compared to the initial release of the enabler in D3.2, the following achievements have been made since then:

A prototypical implementation of the enabler has been developed and tested in generic cases as well as with a real IoT testbed. The current version of the enabler consists of the following new features:

- Data collecting: The data collector is now able to deal with multiple formats of different data sources provided by the agents, namely the MQTT messages, the logs in json/csv, the IoT-6LoWPAN traffic). This will be discussed in Section 4.2.1.
- Data processing: Different data processing techniques have been performed including normalization, attribute selection, similarity calculation to eliminate the noises, reduce the computation time, and enhance the efficiency as well as the accuracy of the analysis. This will be discussed in Section 4.2.2.
- Reaction and Visualization: The analysis results are published back to the system's administrators/ DevOps engineers via the MQTT communication and on GUI dashboards). This will be discussed in Section 4.2.3.

Furthermore, the RCA Enabler is currently being evaluated in the ITS use case and the results will be reported in the deliverable D1.4 and D1.5.

Regarding the status of all the features to be delivered at the end of the project as described in the "Extra Document"[15]. Table 13 summarizes their status.

Feature	Description	Status at M22	Status at M35
DevOps features			
Continuously detect causes of both security and operational incidents.	See section 4.3.2. Security attacks (e.g., DoS/DDoS) and operational incidents (e.g., node failure) were detected.	Ongoing	Completed
Continuously learn about the detected issues in the past to enhance future analysis.	See section 4.3.2	Ongoing	Completed

[15] the "Extra Document": Plans for development and evaluation for 2020 and until the end of ENACT. Submitted to the Project Officer in January 2020 as an additional document to the planned deliverables.

Provide recommendation to react.	See section 4.3.2	Ongoing	Completed
Trustworthiness features			
Allow security expert to automatically identify the initiator of both security and functional flaws.	See section 4.3.2	Ongoing	Completed
Allow the security expert to assess the impact of both the detected flaws and the respective reactions.	See section 4.3.2	Ongoing	Completed

Table 13: Status of all features of RCA

4.2 Technical Description of Enabler

In general, the RCA Enabler works following two phases: knowledge acquisition phase (Figure 33) and monitoring phase (Figure 34). The former is for building a historical database of known problems / incidents; and the latter consists of monitoring the system in real-time, analysing the newly coming incident by querying the historical data, and suggesting possible root causes. The details of each module will be further described in the following subsections.

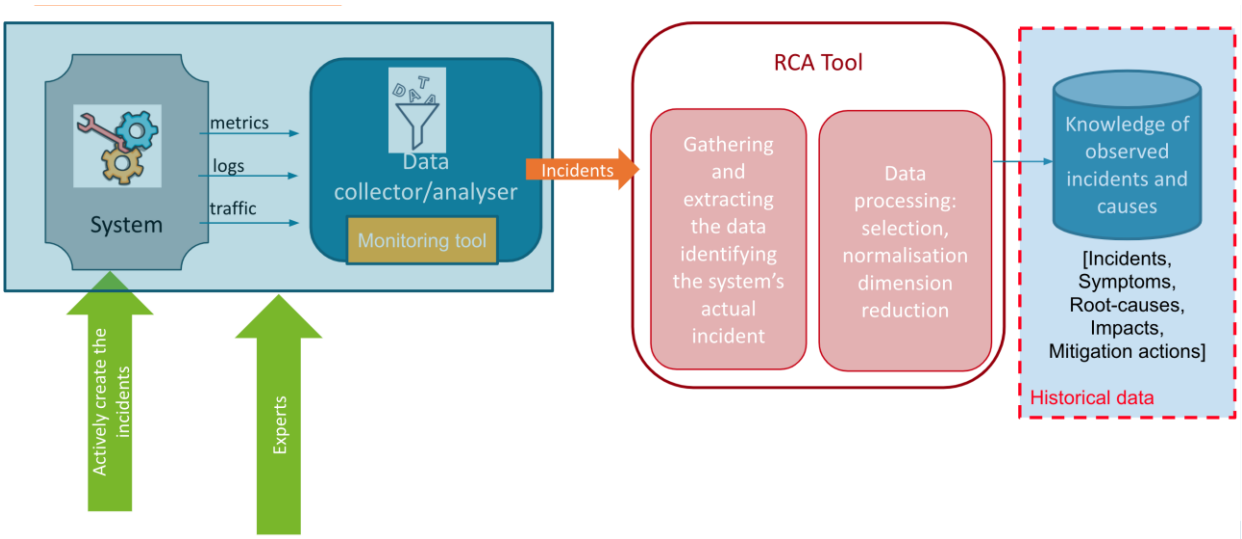


Figure 33: RCA- Knowledge acquisition phase

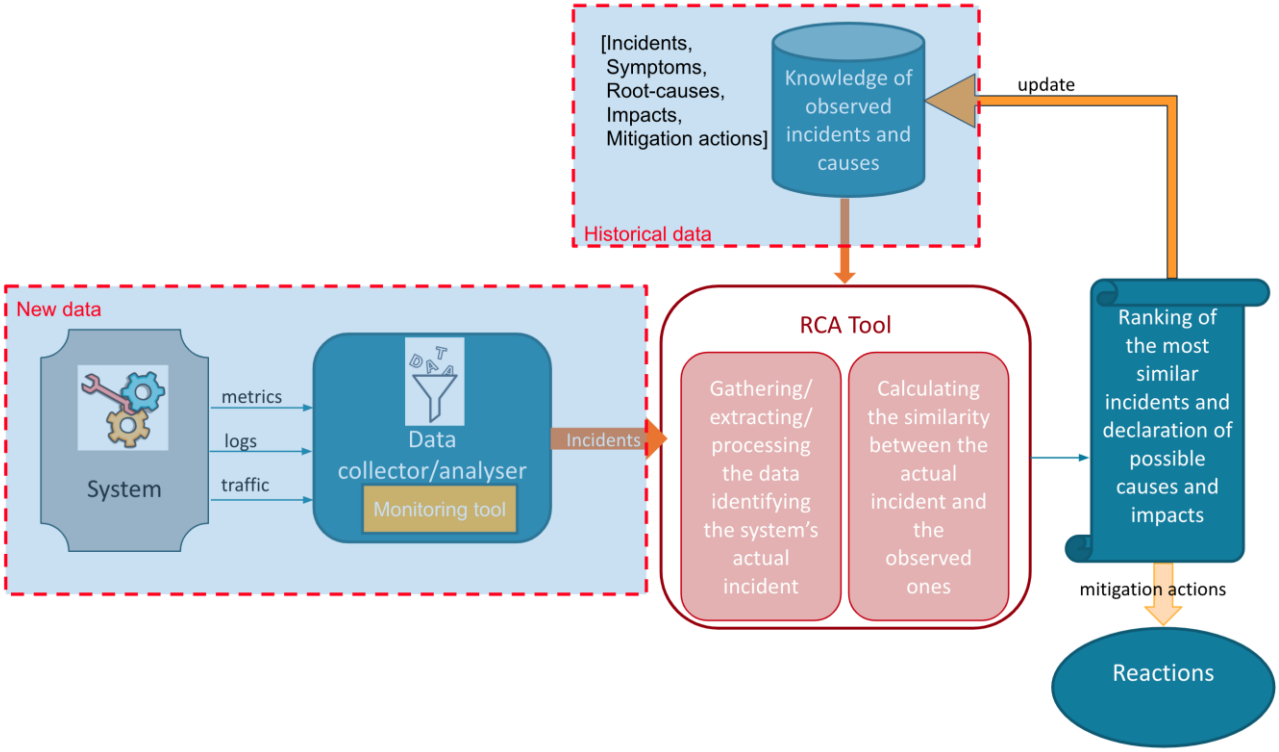


Figure 34: RCA- Monitoring phase

4.2.1 Data collection

In order to analyse the system, different statistics and data are required, including the logs, metrics, traffic and any data that could identify the functioning state of the system. A data collector is necessary and can be provided by the system itself (i.e., in the ITS use case, the metrics are collected and sent to RCA Enabler by the MQTT broker) or an enabler can be deployed for collecting different types of data, namely:

- Capturing network traffic: For example, MMT-Probe (TCP/IP networks), MMT-IoT (IoT-6LoWPAN) are able to sniff and record the traffic.
- Reading and extracting logs: The current version of RCA Enabler supports by default reading the data input of json and csv files. Other formats can be easily added thanks to the extensibility of MMT-Probe (e.g., creating new plugins).

In the knowledge acquisition phase, the data can be collected in two manners:

- Actively injecting or reproducing known failures and attacks, then collecting labelled corresponding data. The list of failures and attacks can be the results taken from a vulnerability scanning process, which is the first step of Risk Management as aforementioned in section 4.1.
- Passively monitoring the system, debugging different logs and traces, correlating various events and especially consulting the system experts to determine the corresponding root causes as well as its relevant data. This is closely related to a monitoring and anomaly detection process, which is the second step of Risk Management as aforementioned in Section 4.1.

In the monitoring phase, the data is collected and transmitted to the enabler in real-time. In theory, there is no restriction in the type of data to be gathered. On the contrary, a maximum amount of available data for identifying the system functionalities is desirable. Even though some data could be redundant, data processing steps will be taken to extract the most appropriate data. This will be discussed in the next section.

4.2.2 Data processing

After being collected, the data need to be cleaned to avoid the noises and to be normalised to fit in our defined standard before being used to calculate the similarity scores. These steps will be demonstrated in the following subsections.

4.2.2.1 Attribute selection

Attribute selection (also known as feature selection) [29] is one of the core concepts in machine learning which tremendously impacts the performance of the model. For complex systems, it is common that the data collected is too complicated or redundant. In the other words, there might be some irrelevant or less important attributes (i.e. noises) contributing less to the target variable. Removing these noises helps us not only to improve the accuracy but also to reduce the training time. This is the first and most important step which should be performed automatically based on the feature selection techniques, or manually with the aid of system experts.

The current version of the RCA Enabler has been integrated with the following feature selection techniques:

- Univariate feature selection: The selection of the best features is based on univariate statistical tests. Each feature is compared to the target variable while the other features are temporarily ignored. The goal is to determine whether there is any statistically significant relationship between them. Each feature has its test score. The bigger the score is, the more likely the feature is important. The features with top scores should be selected. Currently the test score is the average of the scores calculated based on the most well-known models including **chi-square test**, the **f test,** and the **mutual information classification test** [29]**.**
- Recursive feature elimination (RFE): Select features by recursively considering smaller and smaller sets of features. The idea is to use an external estimator (**logistic regression** model and **random forest** model [30]) that assigns weights to features (e.g., the coefficients of a linear model). The least important features are step by step pruned from the current set of features. This procedure is recursively repeated on the pruned set until the desired number of features left is eventually reached. Compared to univariate feature selection, RFE considers all features at once, thus can capture interactions.

Whenever the enabler receives a learning dataset, it evaluates the importance of all given attributes in using the five aforementioned selection models. The synthetic results help us to finally decide which attributes would be taken. A concrete example will be illustrated in the Section 4.3.2.

4.2.2.2 Data normalisation

Normalisation is a concept informally used in statistics and the term "normalized data" may have different meanings. In principle, normalizing data means eliminating the units of measurement of heterogeneous data, making the attributes comparable despite their different ranges of values.
In our perspective, data normalisation consists of two steps:

- Standardizing data to have a **mean** of zero and a **standard deviation (s)** of 1 (Figure 35):

$$x_{standardized} = \frac{x - mean(x)}{s}$$

- Rescaling the data to have values between 0 and 1:

$$x_{normalized} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

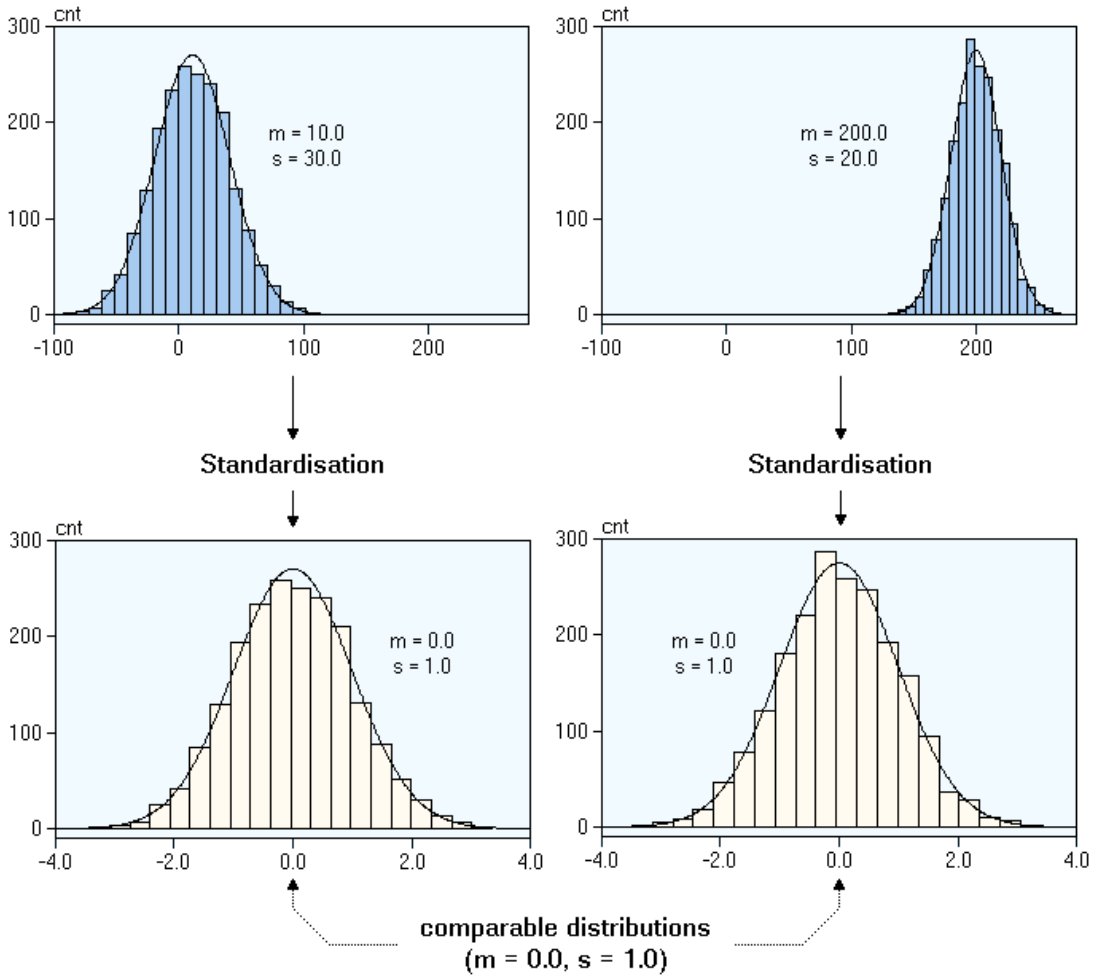


Figure 35: Data standardization examples

4.2.2.3 Similarity calculation

Suppose the temporal state of the system can be reflected by ***n*** metrics (i.e., ***n*** attributes). This set of ***n*** attributes can be represented by a vector in a multi-dimensional space of ***n*** dimensions. Calculating the similarity/ dissimilarity of two states becomes the problem of measuring the distance of orientation (the angle) and of magnitude (the length) of their two representing vectors. Figure 36 depicts an example in a 3-dimensional space.

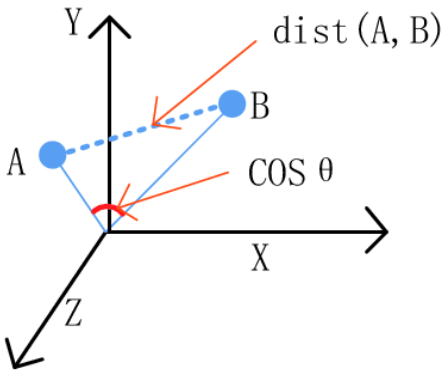


Figure 36: Similarity calculation in 3-dimensional space

The current version of the RCA Enabler has been integrated with the following classic similarity/ distance measures:

- Cosine similarity [31]
- Adjusted cosine similarity [31]
- Jaccard similarity [31]
- Euclidean distance [31]
- Manhattan distance [31]
- Minkowski distance [31]

These measures would be used to calculate the similarity score whose value is between 0 and 1. The bigger the similarity score is, the more similar two compared states are (e.g., if the similarity score equals to 0.95, there is a probability of 95% that two compared states can be considered the same). The similarity score can be computed based on one or multiple similarity/ distance measures. In the training phase, we determine the measures to take into consideration so that when we compare a known state and its repetition, we have the similarity score as high as possible. In addition, to avoid the false positives, the similarity between a common proper state and each known malicious state should be as low as possible.

4.2.3 Reaction and visualization

First, the RCA Enabler analyses in real-time the live data coming from the system under monitoring and reports back to the DevOps engineer the most similar known incident, the corresponding root causes, potential impacts and recommended remediations stored in the historical database in two manners:

- Continuously report the similarity score of the current state and its most similar known incident. If the similarity score climbs over a parameterized threshold, an alert should be raised.
- When a new incident is alerted by another anomaly detector (e.g., BDA Enabler or S&P Monitoring Enabler) or when RCA Enabler witnesses itself several identical symtoms reflected from the system, it raise an alarm.

This helps the system administrators foresee the evolution of the system from a normal functioning state to a known fault or failure, and determine the root causes to be able to perform the most appropriate mitigation actions. For example, in the context of the ITS use case, RCA communicates with the ITS through an MQTT connection. Each time the RCA receives messages containing all the relevant attributes, it performs the analysis to determine at what level the system state is similar to a known incident and publishes the result to the MQTT exchange.

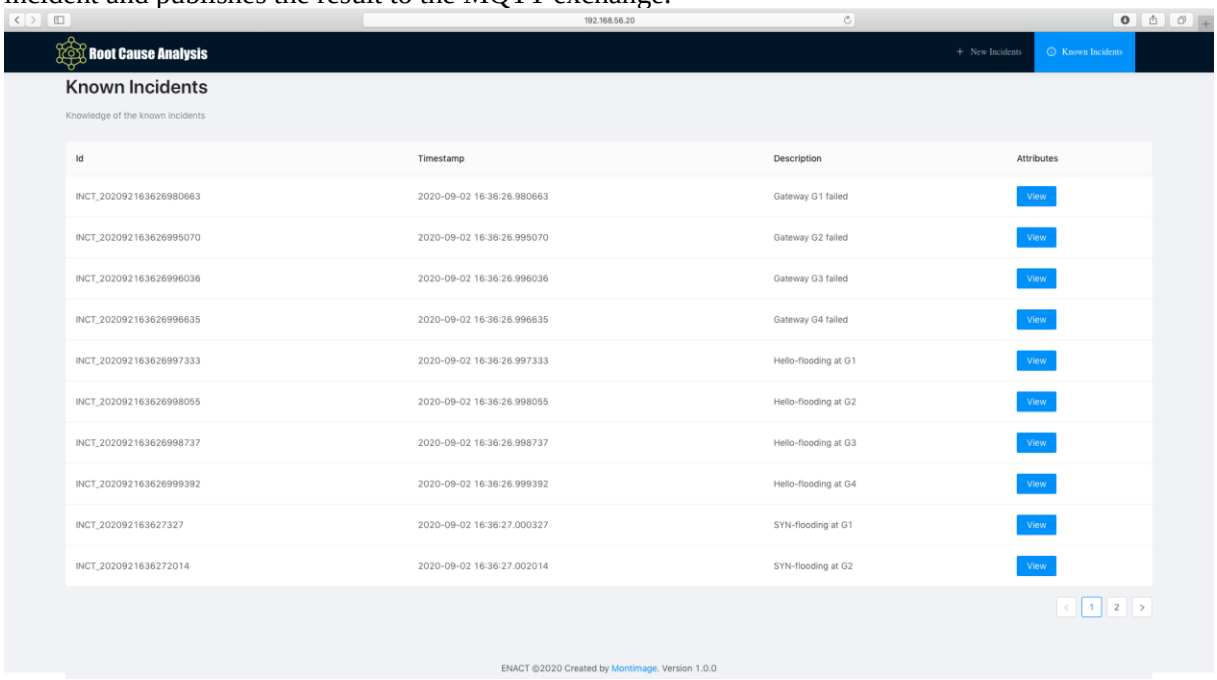


Figure 37: Historical database of known incidents

Regarding the visualization, the results of the RCA Enabler can be viewed intuitively on a GUI that follows strictly the design template of ENACT (ANT design). Figure 37, Figure 38 and Figure 39 present some screenshots of RCA GUI. From the GUI, ones can view the historical database of known/ learned incidents with their corresponding symtoms (represented by the relevant data attributes), potential impacts, and possible mitigation actions based on the experiences. It displays also the newly detected incidents and at what level they are similar to the known ones. If an incident is alerted but there is no significant similar one found, there is an option so that the experts, based on their confidence and knowledge, can intercept and add this current incident to the historical data. The pattern related to it will be automatically taken and saved to the database. The experts are in charge of enter the root-causes based on their manual analysis, the impacts and the mitigation actions.

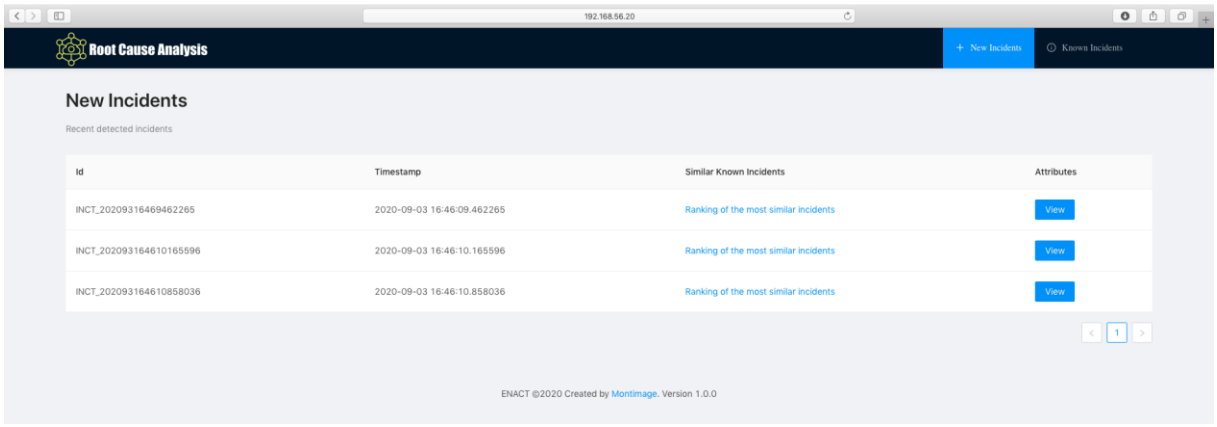


Figure 38: Newly detected incidents

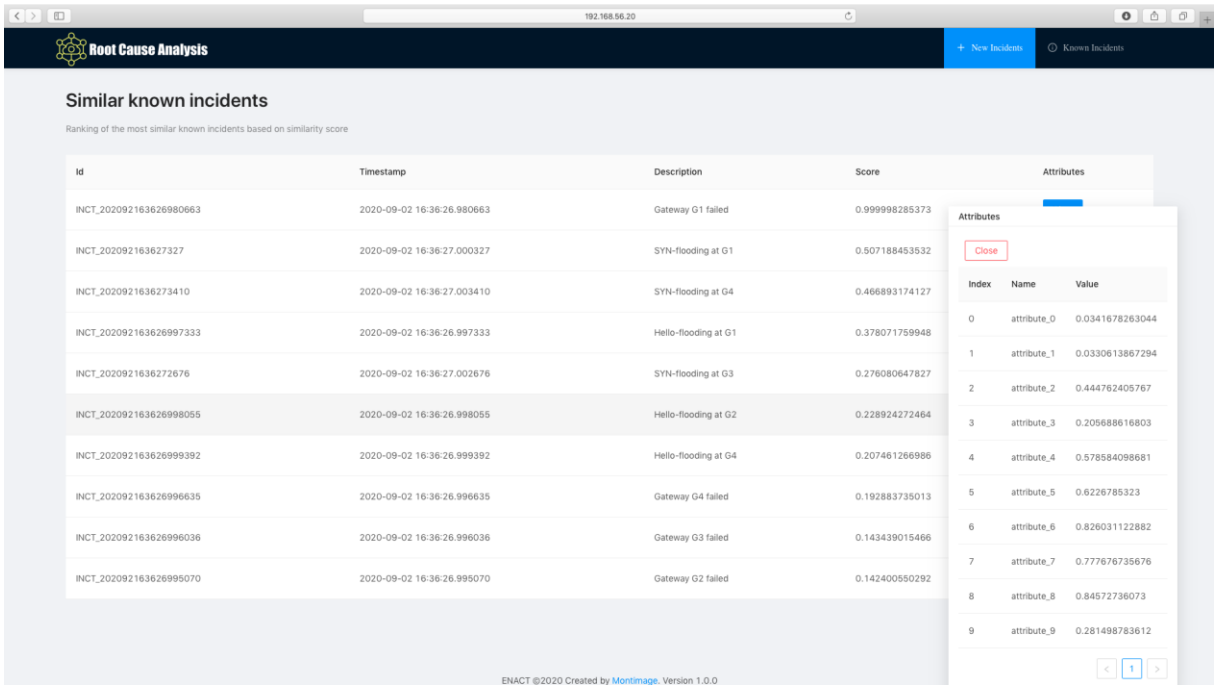


Figure 39: A newly detected incident and its similarity scores in comparison with known ones

In conclusion, a fully functional implementation of the enabler has been developed to collect, process, and analyse the monitoring data to fulfil the task of a root-cause analysis tool. This will be further evaluated in the following section.

4.3 Evaluation

4.3.1 Performance evaluation with generated testing data

To evaluate the RCA Enabler, we first generated a learning data set in CSV format including a number of known records. Each record describes an "incident" with different "attributes" values. These learned "incidents" are stored together with the potential origin causes. The "attributes" refer to the values of metrics that can be gathered. Afterwards, we generated new sets of "attributes" and checked whether they were recognised by the system as a known incident among the ones that were already inserted in the historical data. The RCA Enabler measures the similarity of each new record and each learned incident and ranks them by determining the most likely similar ones. We assess the performance by calculating the response time of the enabler in function of the number of known states and the attributes identifying a state. In this evaluation, the similarity score is calculated as the average of all similarity measures mentioned in Section 4.2.2.3.

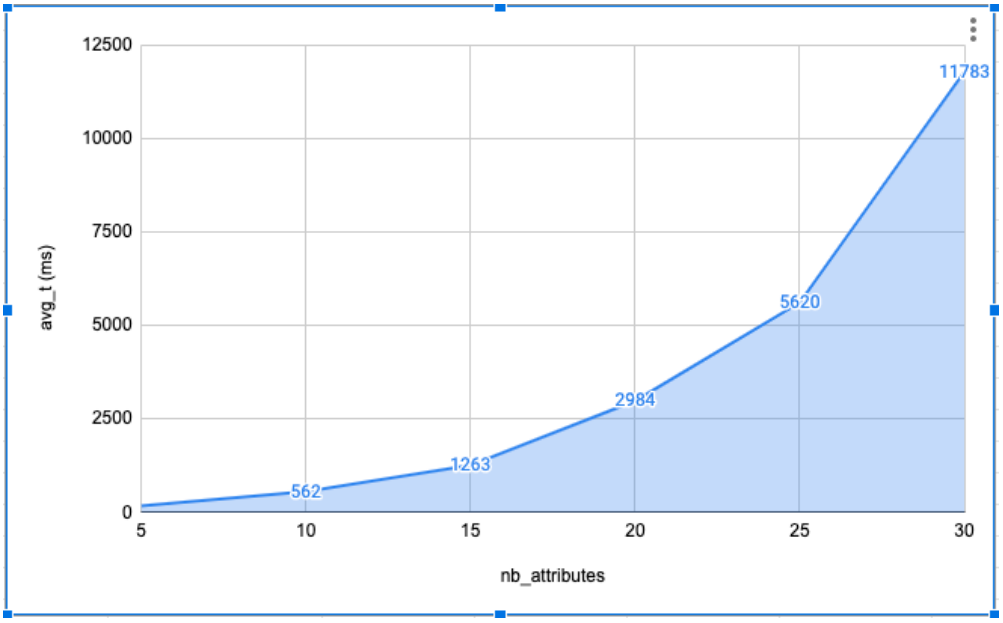


Figure 40: RCA Enabler's response time towards the number of attributes

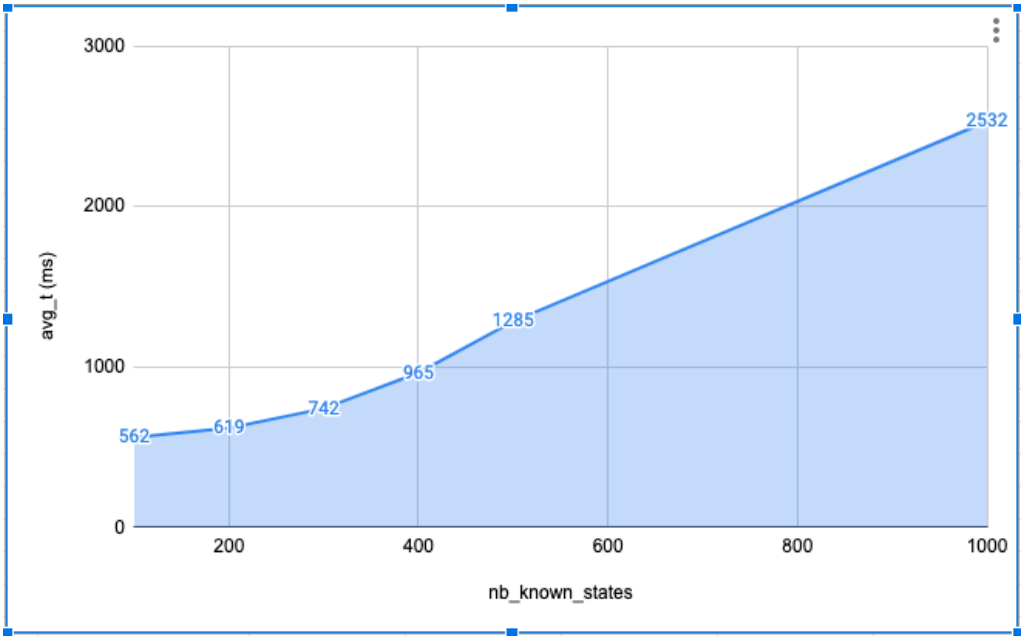


Figure 41: RCA Enabler's response time towards the number of known incidents

Figure 40 and Figure 41 show the results of our experiments. As can be expected, more time is needed when the data volume to be handled increases. However, we observed that the number of known states had less effect on the response time than the number of the attributes did. The processing time needed increases more drastically in the case that a higher number of attributes are under consideration than when there are more known states. This reaffirms the need of integrating "attribute selection" as aforementioned in Section 4.2.2.1. In this test, we did not apply any selection technique because the data was generated in a rather random manner. The attributes are, thus, seemingly equal and make the selection not useful. However, there will probably be a different story when realistic systems are involved (further discussed in Section 4.3.2).

4.3.2 Evaluation on a real IoT Testbed

4.3.2.1 Set-up of the experiment

In order to evaluate the RCA Enabler, we performed a number of experiments on a real IoT testbed called w-iLab.t[16] provided by Imec[17]. Figure 42 demonstrates the general architecture of the experiments. Several IoT devices (Zolertia Re-Mote[18]) formed an IoT network in which the clients report sensed data periodically to the border router, before these reports were forwarded via a USB line to a server installed in a more powerful Linux-based machine. There was an IoT device dedicated to perform sniffing tasks: capturing network traffic, piping via the USB line to the Linux-based machine where MMT-IoT was deployed to analyse the traffic and extract the metrics for the RCA.

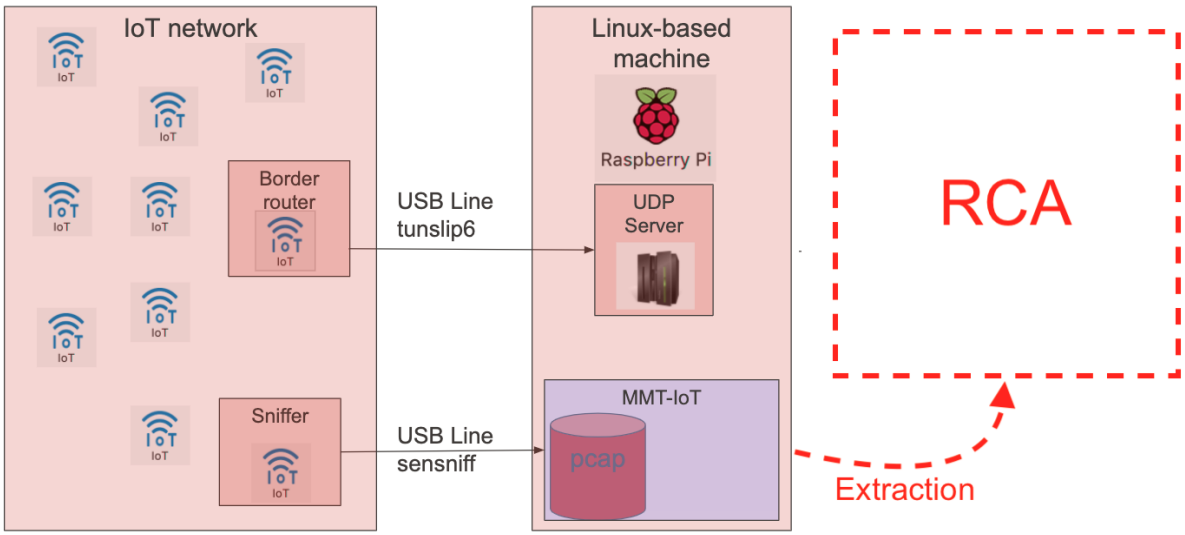


Figure 42: General architecture for the evaluation

In addition, the IoT network consisted of normal clients reporting sensed data every 10 seconds and one (or several) attacker(s) behave interchangeably in three modes:

- Normal mode: reporting data every 10 seconds;
- DoS (Denial of Service) attack mode: reporting data 100 times faster (10 messages/s) and with incorrect Frame Check Sequence (FCS[19]);

[16] https://www.fed4fire.eu/testbeds/w-ilab-t/
[17] https://www.imec-int.com/
[18] https://zolertia.io/zolertia-platforms/
[19] FCS: The FCS field contains a number that is calculated by the source node based on the data in the frame. This number is added to the end of a frame that is sent. When the destination node receives the frame, the FCS number is recalculated and compared with the FCS number included in the frame. If they are different, the frame will be considered malformed (intentionally or not) or modified between the route from the source node to the destination

- Dead mode: not reporting data at all (node failure).

Table 14 summarizes the attributes extracted by MMT-IoT and transferred to RCA for further analysis.

Ref	Attribute	Description
(1)	Network throughput (bps, pps)	The whole network traffic throughput, computed in bits per second (bps) and packets per second (pps)
(2)	Throughput at devices (bps, pps)	Throughput estimated at each device
(3)	Traffic transmitted on links (bytes, packets)	Traffic volume transmitted on each link during a parameterized period (e.g., 10 seconds)
(4)	Number of routing-related packets (packets)	Number of routing-related packets sent and received by each device during a parameterized period (e.g., 10 seconds)
(5)	Transmission delay (ms)	The duration in millisecond since the packet is created by a device (timestamp packaged in the sensed data) until it is captured by the sniffer (captured packet's timestamp)
(6)	CPU usage (%)	CPU usage at each device, packaged in the sensed data
(7)	Memory usage (%)	Memory usage at each device, packaged in the sensed data
(8)	Battery level (%)	Level of battery left at each device, packaged in the sensed data
(9)	Power consumption (W)	Power consumption (DC) at each device in Watts, packaged in the sensed data
(10)	Average packet size (bytes)	Average size of packets transmitted during a parameterized period (e.g., 10 seconds)
(11)	Probe ID	An integer number representing the ID of the MMT-Probe analysing the traffic and performing the extraction
(12)	Protocol ID	An integer number representing the protocol ID

Table 14: Attributes extracted and sent to RCA

4.3.2.2 Attribute Selection and Data Normalisation

As the first step, the RCA Enabler needs to select a number of attributes which are considered "more significant" among 12 ones listed in Table 14 by applying the techniques aforementioned in Section 4.2.2.1.

Table 15 summarises the results when different Feature Selection models were used. There are 6 attributes, namely (1), (2), (3), (5), (6), (7), which are considered important according to all models. Four attributes (8), (10), (11), and (12) are concluded to be not relevant and can be left out. Whilst, the

node. In reality, an incorrect FCS can signify a malformed packet (e.g. due to a misconfiguration or an error in the implementation), a jamming attack (i.e. the attacker abuses the network by generating frames that should be ignored) or a message manipulation attack (i.e. the attacker intercepts and modifies a frame's content).

attributes (4) and (9) are recommended by some models and discommended by the others. The analysis further presented in Section 4.3.2.3 were conducted in taking into account these two attributes and the six attributes recommended by all the models.

Ref	Univariate feature selection			Recursive feature elimination	
	chi-square test	f-test	mutual information classification test	logistic regression model	random forest model
(1)	true	true	true	1	true
(2)	true	true	true	2	true
(3)	true	true	true	2	true
(4)	true	false	true	6	false
(5)	true	true	true	3	true
(6)	true	true	true	2	true
(7)	true	true	true	2	true
(8)	false	false	false	9	false
(9)	false	true	true	6	false
(10)	false	false	false	8	false
(11)	false	false	false	10	false
(12)	false	false	false	10	false

Table 15: Feature Selection results

The knowledge acquisition phase results in the following list of [learned incidents, symtoms, root-causes, impacts, mitigation actions]. To simplify, the symtoms represented by the data attributes will not be displayed.

- [DoS/DDoS related attack, [Symtoms], One or multiple nodes sending data in much higher rate than expected, Sink node might be overloaded and misses the reports from other legitimate nodes, Discard/ Ignore packets coming from the attacker (Blacklist)]
- [Incorrect FCS, [Symtoms], Mis-configuration or Jamming attacks, Sink node wastes resources dealing with these malformed packets and can miss the reports from other normal nodes, Discard/ Ignore packets coming from the attacker (Blacklist)]
- [Node failure, [Symtoms], Node stops reporting for a while, Missing sensed data leads to outdated knowledge about the measured environmment, Verify the dead node]

4.3.2.3 Similarity Calculation and Analysis

Firstly, regarding the DoS attack, one can see clearly in the statistics displayed by MMT-IoT that:

- The traffic volume increased significantly during the attack period (Figure 43).
- The attacker was evidently the most active device (Figure 44) and one end of the most active link (Figure 45).

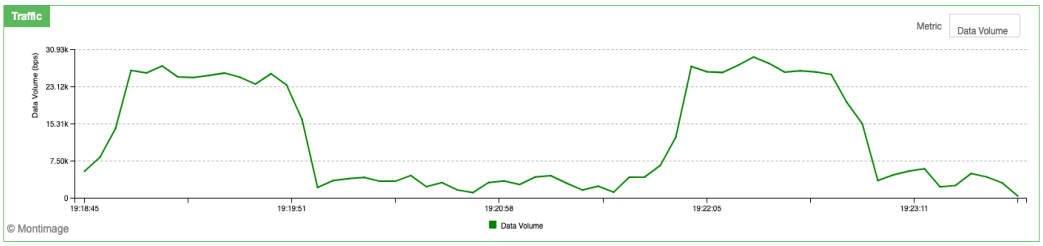


Figure 43: Traffic throughput increased remarkably when the attack took place

From the point of view of RCA, all other selected attributes were more or less affected by the DoS attack. The attack's pattern was learned and when it was repeated, the similarity score observed by the

RCA was always no less than 0.92 (i.e. 92% similar). It is worth noting that in this evaluation, the similarity score is computed based on the "adjusted cosine similarity".
Secondly, in terms of the node failure (dead device), it was reflected also by all the attributes related to this device. The RCA reported the similarity score at the range of between 0.84 and 0.87 when the failure was repeated.

Lastly, when the border router was affected by the jamming attack of incorrect FCS values, its CPU usage jumped remarkably. When this behaviour happened again, the RCA determined that it was up to 94% similar to the learned incident. However, even when all the devices in the network worked normally, the RCA considered that there was up to 78% similarity with the event "Possible jamming attack with incorrect FCS values".

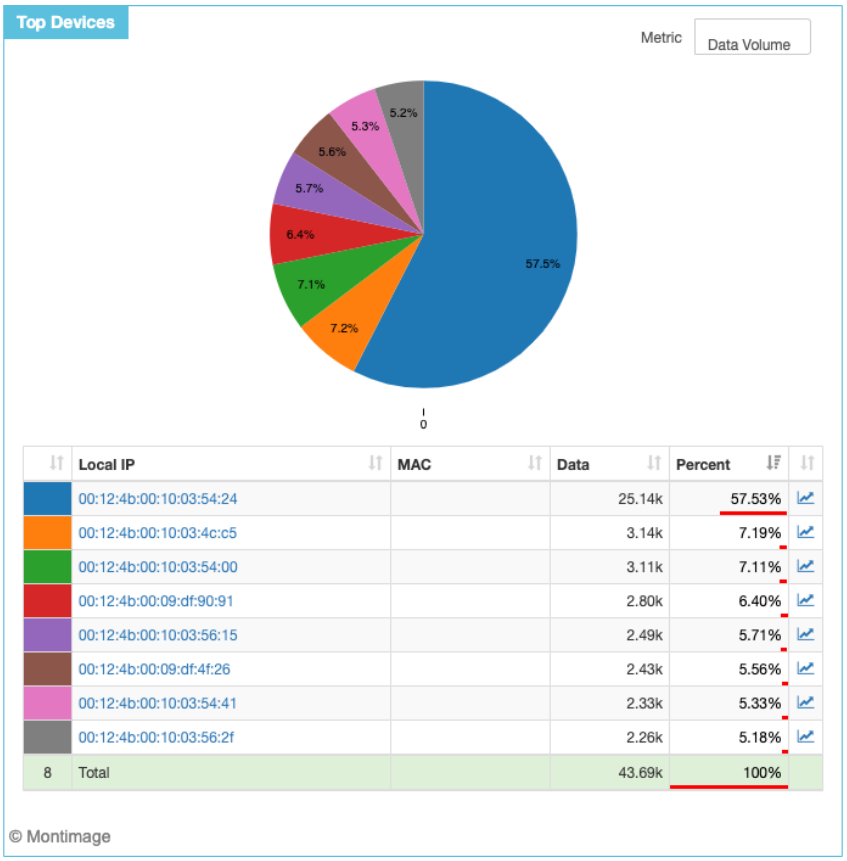


Figure 44: The attacker was the most active device

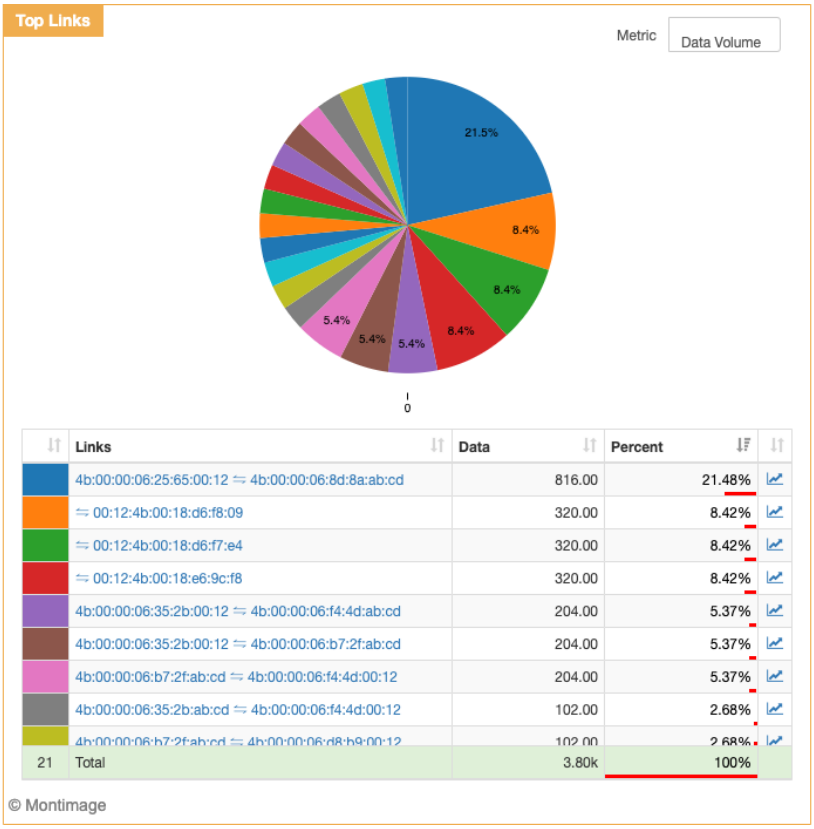


Figure 45: The attacker belonged to the most active link

In conclusion, the aforementioned experiments have proven that the implemented RCA Enabler is able to detect the root-causes of both security and operational incidents as long as they can be produced to be learned into the historical data. This is a continuous learning that can be performed actively and passively. The tool has accomplished the RCA task with a good performance and accuracy that promise a near real-time solution. It will be further evaluated with a more complex smart IoT system which is the ITS use case of ENACT. The results will be described in the D1.4 and D1.5. Although in the context of ENACT the enabler is assessed mostly in IoT systems, it can be applied to many other industrial systems. This is the main topic of the next section.

4.4 Beyond ENACT

The RCA can be applied to various systems other than IoT systems as in the context of ENACT. In general, it can be adapted to any system in which the data identifying its functioning state can be collected. It would be useful if the owner/ administrator has already acquired a significant understanding about the system to facilitate the training phase and the construction of the historical database of known incidents and root-causes. Otherwise, pen-tests can be performed to discover potential vulnerabilities, and so that the attacks/ failures can be injected and then learned.

The RCA Enabler has been developed to be a generic solution with some adaptations to the ITS use case of ENACT. Moreover, RCA enabler is planned to be reused in several other European projects dealing with different contexts (e.g. 5G mobile networks), as well as to become an important tool in the ecosystem of Montimage and to be commercialized together within the MMT-Box[20]. It can also be applicable in industrial SCADA systems.

[20] https://montimage.com/products/MMT_Box.html

5 Adaptation enactment as support for self-adaptation

GeneSIS (aka. the orchestration and deployment enabler) aims at supporting the automatic deployment and adaptation of SIS, including the security and privacy mechanisms, over IoT, Edge and Cloud infrastructure. In this document, we focus on reporting the GeneSIS' support for the adaptation of SIS, including the adaptation of the security and privacy mechanisms. Section 5.1 gives an overview of the adaptation enactment support by GeneSIS with its main achievements. Then, in Section 5.2, we present the technical details of the adaptation enactment support. Next, we demonstrate how GeneSIS can enable the adaptation of SIS together with the security (and privacy) mechanisms in Section 5.3. Finally, we envision in Section 5.4 how GeneSIS can realise more of its potential even beyond the scope of the project. Note that D2.3 provides more details of the orchestration and deployment support by GeneSIS.

5.1 Overview and Main Achievements

DevOps promotes an iterative and incremental approach enabling the continuous evolution of software systems. A key feature to enable such continuous evolution is to support the adaptation of the system with minimal impact on the system already delivered and under operation. This includes ensuring availability and security mechanisms, which must evolve along with the smart IoT systems, continuously fixing security defects and dealing with new security threats.

Within WP3, the focus of GeneSIS is on the adaptation support offered to other tools (from ENACT or not) and to DevOps teams. From this perspective, the main contributions of the enabler are the following:

- The same language can be used by Dev and Ops teams to specify and to adapt a deployment. This language is platform and technology independent supporting the specification of adaptation in deployment over IoT, Edge and Cloud infrastructure. This includes:
    - The specification, deployment, and redeployment of software on devices with no or limited access to Internet.
    - The specification, enactment, and runtime management of availability mechanisms (rolling deployment) in a platform independent way.
    - The specification and (re)deployment of security mechanisms as part of the system in a platform independent way.
- The GeneSIS models@run.time based engine is used to enact a deployment with the following benefits:
    - During a redeployment/adaptation only the minimal necessary part of the system that needs to are adapted, minimizing the impact of a redeployment for better availability.
    - The deployment model is maintained up-to-date, ensuring subsequent DevOps cycles to always work on up-to-date version of the deployment.
    - Deployment models are always checked and validated before a deployment or redeployment is performed.
- A set of APIs for third parties to trigger an adaptation and manipulate a model.

Compared to the initial release of the enabler in D3.2, the main evolutions are:

- Support for blue/green (rolling) deployment and redeployment for maximum reliability and availability. Blue/green deployment can be specified in a platform-independent way.
- Extending support for deployment of security mechanisms toward DevSecOps.
- Refined model checking and comparison mechanism for finer grained adaptation and better reliability.
- Introduction of the concept of Monitoring agent.
- Plug-in mechanism to (i) dynamically load new component types and (ii) extend the GeneSIS deployment engine.

Regarding the status of all the features to be delivered at the end of the project as described in the "Extra Document"[21] all the promised features are delivered as summarized in the following Table 16.

Feature	Description	Status at M22	Status at M35
DevOps features			
Specify deployment model of the overall SIS over Cloud, Edge, IoT resources	See D2.3	Completed	Revised
Check validity of deployment model	See Section 5.2.2	Ongoing	Completed
Specify hardware specificities and requirement to ensure compliance of the deployment with respect to these specificities	See D2.3	Completed	Completed
Provisioning and automatic deployment over Cloud, Edge, IoT resources and the orchestration of the deployed software instances – i.e., create VMs in the cloud, deploy using Docker, SSH, Ansible, update firmware, and deploy on resources with limited access to Internet	See D2.3	Completed	Completed
Automatically adapt a deployment in a declarative way	See section 5.1.2	Completed	Completed
Same language used for design and runtime activities – i.e., runtime information is reflected in the language	See section 5.1.2	Completed	Completed
Trustworthiness features			
Security and Privacy: specify requirements in terms of security and privacy and support the deployment of the selected corresponding mechanisms	See D2.3	Completed	Revised
Security and Privacy: specific GeneSIS components for the deployment of security and privacy mechanisms (at least the ENACT enablers)	See Section 5.2.6. Add support for: (i) injecting ad-hoc and fine-grained security policies in IoT middleware, and (ii) integration and deployment of third-party security mechanisms.	Ongoing	Completed
Security and Privacy: support automatic encrypted communication between deployed components without any change in the component code.	See D2.3	Planned	Completed

[21] "Extra Document": Plans for development and evaluation for 2020 and until the end of ENACT. Submitted to the Project Officer in January 2020 as an additional document to the planned deliverables.

Provide a means to deploy actuation conflict manager and include the concept in the language	See D2.3	Completed	Completed
Support the automatic blue/green deployment of Cloud and Edge resources	See Section 5.2.5	Planned	Completed

Table 16: The promised features and corresponding status of the orchestration and deployment enabler

5.2 Technical Description of Enabler

As discussed in our different literature reviews [32] [33] [34], whilst many solutions exist for the continuous deployment of cloud-based systems, very little effort has been spent on providing solutions tailored to the delivery and deployment of applications across the whole IoT, Edge, and Cloud space. GeneSIS enables continuous orchestration and deployment of Smart IoT Systems throughout the IoT – Cloud continuum. A detailed description of GeneSIS (a.k.a. the orchestration and deployment enabler [35] [36] [37]) can be found in deliverable D2.3. In this section we focus on presenting how GeneSIS supports dynamic adaptation in the deployment of a Smart IoT System (SIS).
From a deployment model specified using the GeneSIS Modelling language (see D2.2 for details about the GeneSIS modelling language), the GeneSIS deployment execution engine is responsible for: *(i)* deploying the deployable artefacts (i.e., installing and configuring the actual software binaries, code, scripts, etc. to be deployed on the target), *(ii)* ensuring communication between them (i.e., deployed components are configured to ensure communications between them), *(iii)* provisioning cloud resources (i.e., cloud resources need to be created before a software can be deployed on it. For instance, a virtual machine has to be created and started before a software component can be deployed on it), *(iv)* monitoring the status of the deployment (i.e., monitoring if hosts are still reachable and if software components are still running), and *(v)* adapting a deployment (i.e., modifying how the SIS is deployed).

5.2.1 Overall architecture

The GeneSIS deployment engine implements the Models@run.time pattern to support the dynamic adaptation of a deployment with minimal impact on the running system. Models@run.time [38] is an architectural pattern for dynamic adaptive systems that leverage models as executable artefacts that can be applied to support the execution of the system. Models@run.time provides abstract representations of the underlying running system, which facilitates reasoning, analysis, simulation, and adaptation. A change in the running system is automatically reflected in the model of the current system. Similarly, a modification to this model is enacted on the running system on demand. This causal connection enables the continuous evolution of the system with no strict boundaries between design- and run-time activities.

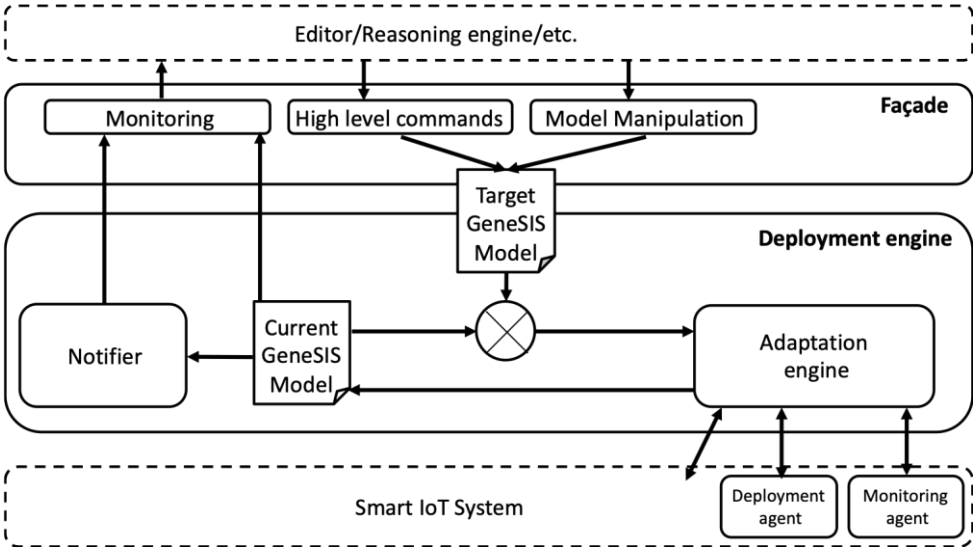


Figure 46. Architecture of the GeneSIS execution engine.

Our engine is a typical implementation of the Models@run.time pattern. When a target model is fed to the deployment engine, it is compared (see crossed circle ⊗ in Figure 46) with the GeneSIS model representing the running system. Finally, the adaptation engine enacts the adaptation (i.e., the deployment) by translating the difference between the current and the target models into one or several of the following deployment steps. After the deployment, the engine synchronizes the current GeneSIS model with the actual deployment result.

The GeneSIS deployment engine is non-invasive, meaning it does not require any GeneSIS bootstrap or agent running on a target device to deploy software on it. However, when decided by the DevOps engineer, the GeneSIS deployment engine can deploy on a target device a monitoring agent. This agent is an instance of netdata[22] and provides information about the performance and health status of the Infrastructure-Component, including data about the Software-Components it hosts (Figure 47). The engine can also deploy on a device a so-called deployment agent. The GeneSIS deployment engine can thus delegate part of the deployment procedure to this agent. This is particularly relevant when a device is not directly reachable by the deployment engine but via another device (e.g., typically the case in local networks). More details about the deployment agent can be found in D2.3.

Figure 47. GUI of the netdata monitoring agent.

[22] https://github.com/netdata/netdata

5.2.2 The GeneSIS Comparison Engine

The comparison engine (denoted as ⊗ in Figure 46) is at the core of the GeneSIS support for dynamic adaptation. The inputs to the *Comparison engine* (also called *Diff*) are the current and target deployment models. The output is a set of lists of changes (that can directly be mapped to actions) representing the required evolutions to transform the current model into the target model. The different lists are presented in Table 17. The evaluation of this list by the GeneSIS execution engine results in: *(i)* modification of the deployment and provisioning topology and *(ii)* deployment of the evolved SIS.

List	Content	Used for
list_removed_ infrastructure_components	List of Infrastructure components removed in the target model compared to the current one	Terminate or stop a cloud service
list_added_ infrastructure_components	List of Infrastructure components added in the target model compared to the current one	Provision and configure a cloud service and check availability of the other type of infrastructure components (i.e., the ones that cannot be provisioned)
List_of_removed_ software_components	List of software components removed in the target model compared to the current one	Remove the internal component instance from its current host
list_of_added_ software components	List of software components added in the target model compared to the current one	Deploy the internal components and configure the external ones
list_of_added_ hosted_components	List of software components that are hosted on another component added in the target model compared to the current one	Deploy the internal component instance from its current host
list_of_removed_ communications	List of communications removed in the target model compared to the current one	Disconnect the endpoints of the communication
list_of_added_ communications	List of communications added in the target model compared to the current one	Configure the endpoints of the communication
list_of_removed_ containments	List of containments removed in the target model compared to the current one	Check the validity of the deployment
list_of_added_ containments	List of containments added in the target model compared to the current one	Check the validity of the deployment
list_of_added__ deployer	List of communications with the property ‘isDeployer’ set to true (i.e., meaning that a deployment agent will be required) added in the target model compared to the current one	Check the list of components that need to be deployed by a deployment agent and deploy them

Table 17: Types of output generated by the Comparison engine

The comparison engine processes the entities composing the deployment models in the following order: infrastructure components, software components, communications, containments, on the basis of the logical dependencies between these concepts. In this way, all the components required by another component are deployed first.

For each of these concepts, we compare the two sets of instances from the current and target models. This comparison is achieved on the matching of the various properties of the instances, of their types, and on their logical dependencies with other components.

Compared to D3.2, the comparison of components and link instances has evolved and does not focus anymore on the following few properties: *Name, IP, Resource*, and *Ports*. Instead, the comparison engine now performs a deep comparison of the instances where all their attributes are considered and compared. This deep comparison is recursive, meaning if an attribute is itself an object, all its own attributes will be evaluated, and so recursively. It is also possible to exclude some attributes from the comparison when needed. The list of attributes to be excluded from the comparison can be provided in a configuration file (see Listing 1). Typical example of attributes excluded are runtime information attached to instances in the current deployment model but are never in the target model. As a result, the comparison mechanism of the GeneSIS deployment engine can be dynamically tuned based on the deployment context, resulting in greater flexibility. To the best of our knowledge, this approach is very seldomly offered by models@run-time execution engines.

Listing 1. Configuration of the comparison engine.

```
const excluded_properties = ["_runtime", "container_id", "_configure"];
```

In addition, the following changes in logical dependencies can lead to mismatch between two instances of components:

- The component is not anymore on the same host (can be because the host itself has been updated).
- The other endpoint of a communication with the mandatory property set to true has changed.

For each unmatched instance from the current model, the instance is added to the corresponding list of removed instances. Similarly, for each unmatched instance from the target model the instance is added to the corresponding list of added instances.

It is also worth noting that before each comparison, the target model is checked and validated, ensuring its consistency. In particular, the following properties are checked:

- *Components, ports* and *links* have unique IDs.
- *Links* and *Containments* have no open endpoints.
- All *InternalComponents* are hosted on another component specified in the model.
- Components only have one link with the *isController* property.
- All the required ports with the *isMandatory* property are fulfilled.
- A *port* with a required *Security, Software and Hardware Capability* is linked to a port providing the corresponding *Security, Software and Hardware Capability*.

As said before, from the list of changes identified by the comparison engine, the GeneSIS execution environment derives an adaptation plan (typically a one-to-one mapping), which, in turn, is used to enact the adaptation of the system (i.e., of the deployment of the system). This process can be triggered by providing the GeneSIS execution environment with a new deployment model via its Façade. In the following we detail the different means to interact with the GeneSIS execution environment.

5.2.3 Interacting with the GeneSIS execution environment

The facade provides a common way to programmatically interact with the GeneSIS execution engine via a set of three APIs.

- The monitoring API offers mechanisms for remote third parties (*e.g.*, reasoning engines) to observe the status of a system. Third parties can either consume the whole GeneSIS model of the running system enriched with runtime information or subscribe to a notification mechanism.

- The high-level commands API exposes a pre-defined set of high-level commands that avoid direct manipulation of the models (*i.e.*, the model is automatically updated when the command is triggered).
- Finally, the model manipulation API provides support for the atomic MOF-level[3] modifications of the deployment model.

In the following we detail each of these interfaces.

5.2.3.1 The Monitoring API

For its monitoring interface to send notification to third parties, GeneSIS embeds a MQTT message broker. MQTT has been selected because it follows the Publish-Subscribe pattern, it is scalable and offers great space, time, and synchronization decoupling [16]. At the time of writing, GeneSIS exploits two topics in the message queue to distributes its messages:

- **Status:** this topic is used to distribute messages about the status of the different components that form the SIS. Possible statuses are: error, running, and under_configuration. Messages are written in JSON as follows:

```
{
    node: <name of the component>,
    status: 'error'
}
```

- **Notification:** this topic is used to distribute messages describing the status of the deployment process. Possible statuses are: success, error, ongoing Messages are written in JSON as follows:

```
{
    action: <name of the deployment action>,
    status: 'success'
    message: 'Component A has been successfully deployed'
}
```

5.2.3.2 The High-level commands API

A set of high-level commands are exposed by GeneSIS in the form of a REST API. As already detailed in D3.2, this includes commands to retrieve and deploy a deployment model depicted in Table 18.

Method	Resource	Content-type	Description
GET	/genesis/types	Response: application/json	To retrieve all the component types registered in the execution engine
POST	/genesis/deploy	Response: application/json Parameter: application/json	To provide the engine with a new deployment model and trigger a re-deployment
GET	/genesis/logs	Response: text/plain	To retrieve all the logs from the execution engine
GET	/genesis/model	Response: application/json	To retrieve the current deployment model
GET	/genesis/model_ui	Response: application/json	To retrieve the current deployment model including its graphical representation

Table 18. High-level commands already defined in D3.2

The deploy command is composed of two operations: *(i)* pushing a model to the GeneSIS execution as a new target model and *(ii)* triggering the deployment with the objective to reach the topology described in this target model. We have extended the API, providing third parties with the ability to manually control these two operations independently via separate commands. This provides an indirection

between these two operations, which is particularly interesting when a set of operations such as model checking or an optimization of the deployment needs to be performed on the target model before enacting the deployment. This, for instance, applies when the Actuation Conflict Management (ACM) enabler and GeneSIS are integrated within a DevOps pipeline. Once a deployment model is defined (or updated) for a smart IoT system, the GeneSIS execution engine can check it against actuation conflicts before triggering the deployment. These commands are detailed in Table 19.

Method	Resource	Content-type	Description
POST	/genesis/push_model	Response: application/json	To push a new target deployment model into the GeneSIS deployment engine without triggering the deployment
POST	/genesis/deploy_model	Response: application/json Parameter: application/json	If a target deployment model is loaded in the execution engine and has not been deployed yet, deploys it
GET	/genesis/get_target_model	Response: text/plain	To retrieve the target deployment model

Table 19. New deployment commands added since D3.2

Finally, a new command has been added for generating the documentation of the APIs offered by the GeneSIS execution engine (Table 20). The description of the API is made using swagger. The content generated is a full webpage, which can be browsed using any regular browser. Another option is to point the users of the GeneSIS API to the online documentation attached to the GeneSIS code repository. However, by providing each GeneSIS execution engine with the means to generate its own documentation, users of a GeneSIS execution engine are always provided with a documentation aligned with the running version of the engine.

Method	Resource	Content-type	Description
POST	/api-docs	Response: Application/xml	Generates and delivers the API documentation

Table 20. Accessing the GeneSIS documentation

5.2.3.3 The Model Manipulation API

As presented in D3.2, one of the objectives for the second period was to extend the GeneSIS execution engine façade with an API enabling the MOF-reflection atomic modifications of a deployment model. This includes introspecting and modifying components and links as well as adding/removing components and links into a deployment model. Details about the API are presented in Table 21.

It is worth noting these atomic modifications do not result in a redeployment, which must be triggered manually. The changes are applied on the target deployment model in memory of the GeneSIS execution engine. In case there is no such model, it is created at the first modification.

Method	Resource	Content-type	Description
PUT	/genesis/component	Response: application/json Parameter: application/json	To update an attribute of a component or add a component. Consumes as parameter, a JSON including, the name of the component, the name of the attribute and the new value or the JSON serialization of a whole component in order to add it to the deployment model
PUT	/genesis/link	Response: application/json Parameter: application/json	To update an attribute of a link or add a link. Consumes as parameter, a JSON including, the name of the link, the name of the attribute and the new value or the JSON serialization of a

			whole link in order to add it to the deployment model
GET	/genesis/component	Response: application/json	To retrieve information about a specific component. Return a JSON description of the component, including runtime information (i.e., status)
GET	/genesis/link	Response: application/json	To retrieve information about a specific link. Returns a JSON description of the link

Table 21. Model manipulation API.

5.2.4 Dynamically extending the GeneSIS execution engine

GeneSIS comes with a set of predefined component types that can be seamlessly instantiated in deployment models. In addition, GeneSIS embeds a plugin mechanism that enables the dynamic loading of new component types. This plugin mechanism leverages the module facilities offered by Nodejs. Each component type has to be designed as a Nodejs module in a single file to be placed in a specific folder (i.e., the folder named repository at the root of the GeneSIS installation) of the GeneSIS execution engine. This repository is scanned by the GeneSIS execution engine before each deployment ensuring all available types are loaded before a deployment model is analyzed and deployed. The plugins are loaded in memory as regular Nodejs modules from the repository as depicted in Listing 2.

Listing 2. Dynamic loading of component types.

```
if (extname === '.js' && (opts.filter === undefined || opts.filter(current))) {
        var module = require(current); // module is loaded
        var tmp = {};
        tmp.id = stat.name;
        tmp.module = module;
        modules.push(tmp); // and placed in a repository with metadata
}
```

When loading and analyzing a deployment model, for each component, the GeneSIS execution engine checks first if the component is an instance of a primitive type, if not, it searches for its type specification in the repository. Listing 3 illustrates how to specify a component type. Some of the requirements when specifying a new type are:

- The module needs to import the GeneSIS metamodel (cf. First line of Listing 3).
- The module needs to be exported (cf. Last line of Listing 3).
- The component type needs to inherit from another component type (primitive or not). In Listing 3, we are creating a ThingML component type, that inherits from Software component.
- The type constructor must consume as argument a JSON object specifying values of its attributes. All the attributes not specified are instantiated with default values.

Listing 3. Example of a component type as a plugin.

```
var mm = require('../metamodel/allinone.js'); // loading the GeneSIS metamodel
var uuidv1 = require('uuid/v1');

/*****************************/
/*      ThingML component     */
/*****************************/
var thingml = function (spec) {
    // The inheritance - attributes are inherited and set with default values
    var that = mm.software_node(spec);
    // Attributes can be defined for this specific type
    that._type += "/thingml";
    that.name = spec.name || "MainCfg";
```

```
    that.nr_description = spec.nr_description || "";
    that.file = spec.file || "./LightSensorApp.thingml";
    that.src = spec.src || "";
    that.libraries = spec.libraries || ["Adafruit GFX Library","Adafruit ST7735 Library"];
    that.target_language = spec.target_language || "java";
    that.config_name = spec.config_name || that.name;

    // This function is called before the resources are used
    that._configure = function(){
        // Some code
    }

    return that;
};

module.exports = thingml;
```

The deployment of a software component is typically specified in the *Resources* attached to it (see D2.3 for details about the concept of resources). As a recall, *Resources* provide a means to specify how a software component can be installed and deployed on a target host. Nevertheless, sometimes it is desirable or even needed to extend this mechanism as components may require some specific processing on the GeneSIS execution engine side (and not on the target host) before the *Resources* can be used to deploy them. For instance, ThingML components need to be compiled using the ThingML compiler and then build using tools such as Maven before the deployable artefact can be deployed on the target host. It is particularly beneficial delegating this task to GeneSIS as it enables the seamless migration of a component from one host to another without manually rebuilding it in case the two hosts are not offering the same execution environment (e.g., if moving from a host running Java to a host only offering Nodejs as execution environment). Similarly, any component type loaded as a plugin may require a similar processing from GeneSIS. However, all processing for all possible component types cannot be hard coded into the GeneSIS execution engine. To overcome this issue, plugins are allowed to embed specific behavior to be executed by the GeneSIS execution engine. More precisely, component types may come with a method called “configure” (see Listing 3), which is executed by GeneSIS before it considers the *Resources* attached to the component providing a means to customize the deployment process.
The GeneSIS execution has also been extended with mechanisms to support the blue/green deployment of specific software components of a SIS. Details about these mechanisms can be found in D2.3. In the following we report how the deployment of software components deployed using the blue/green deployment strategy can be maintained and adapted at runtime.

5.2.5 Availability & Fault-tolerance Mechanisms

Availability refers to the ability of the system to mask or repair faults such as the cumulative service outage period does not exceed a required value over a specified time interval [39], etc. To improve availability and fault-tolerance, which is the ability for the system to hide faults from its users, is of primary importance. Container technologies support high-availability by providing fault-tolerance mechanisms and zero downtime upgrades. Docker Swarm, for instance, manages multiple image replicas (i.e., multiple copies of the same software component) and upgrades them gradually with minimum disruption of the users, by implementing various well documented strategies such as, replication, blue/green deployment, or rolling upgrades [36]. These tactics are however dependent on the underlying infrastructure, and changing platform requires changes in the deployment model. In other words, a deployment specified for Docker swarm can only be executed via docker swarm introducing some sort of platform dependence.

To better decouple these deployment tactics from the underlying execution platform, GeneSIS provides technology agnostic availability mechanisms. The concrete tactics, namely blue/green deployment and

replication are abstracted away by the number of replicas that must exist at runtime (this number directly drives availability). During deployment, GeneSIS takes into account the underlying platform, exploiting pre-existing mechanisms if any, or deploying built-in components if no support is provided. If GeneSIS detects that the replicated component is deployed on Docker, it directly reuses features from Docker Swarm. If, conversely, it detects that the component is deployed on a simple Linux OS; GeneSIS will deploy separate watchdogs to detect faults, and a proxy to switch from this replica to an operational one if fault is detected. In addition, when the user updates the deployment model, GeneSIS performs a blue/green deployment: It first creates additional replicas (using the new version), then reconfigure the proxy only when these new replicas are operational, and finally, decommissions the older replicas (using the previous version). Although the current version of GeneSIS only supports Docker and Linux operating systems, the principle applies to other technologies, and GeneSIS can be extended accordingly.

When the target platform does not provide fault-tolerance mechanisms, GeneSIS deploys several instances of the user component (so called replicas), together with additional software components to detect and mitigate faults in these replicas. These additional components are:

- A proxy, which routes incoming traffic towards the right replica;
- A watchdog, which periodically monitors the health of replicas, and provisions new one should any fail. The watchdog basically maintains a given number of replicas.

Figure 48 illustrates this setup. To improve availability, GeneSIS starts by deploying a proxy, inactive at first. Then, GeneSIS deploys the watchdog, and sets the requested number of replicas. The watchdog then checks the status of each replica and registers to the proxy those that have successful started up. The proxy is then able to forward incoming requests from the user to one of the replicas. We refer the reader to D2.3 for more details about the implementation of these features.

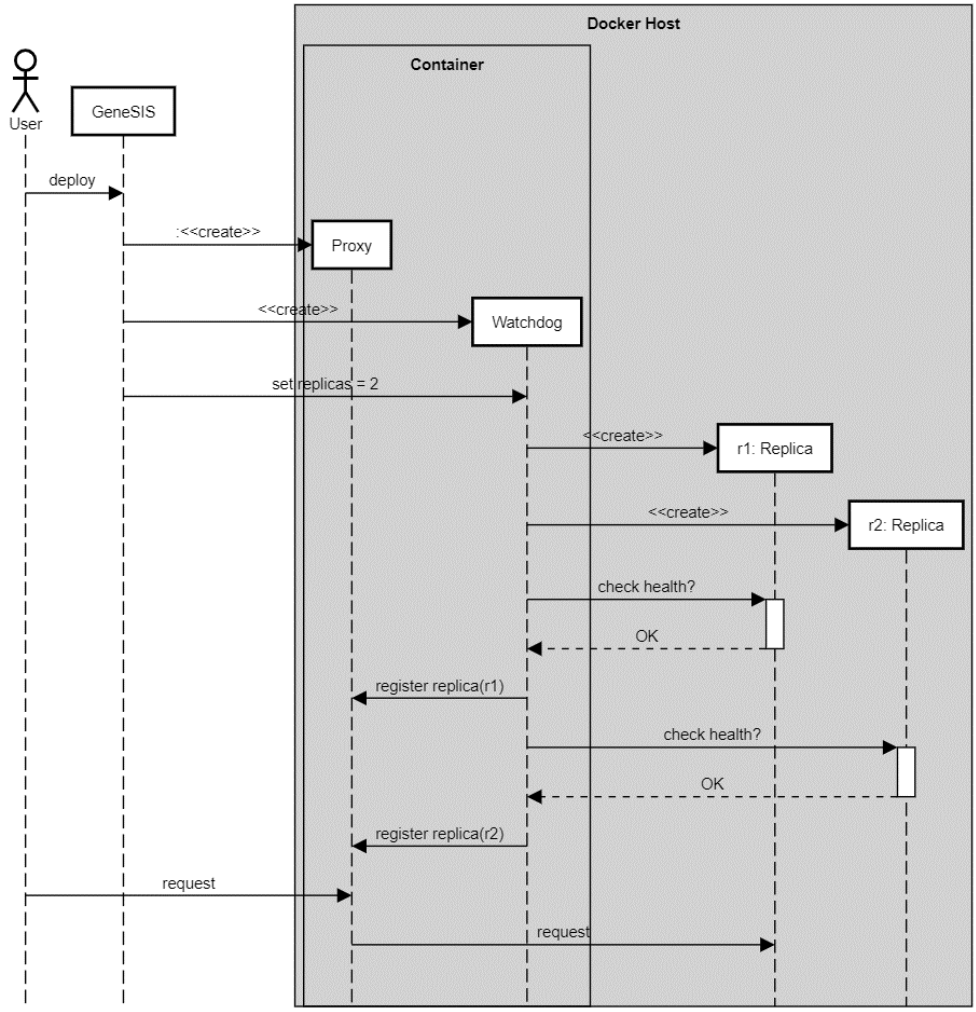


Figure 48. Setting up fault-tolerance mechanisms with GeneSIS.

Figure 49 illustrates how the watchdog masks replicas faults. The watchdog periodically queries replicas (Step 1), expecting a response within a given timeframe (1 s. by default). If the watchdog receives an error response (or no response), it notifies the proxy (3), which therefore selects another replica to process the user's requests (4), and then destroys the faulty replica (5). Then the watchdog creates another replica to take over the failed one (6) and registers it to the proxy once the new replica is up and running (7 – 9).

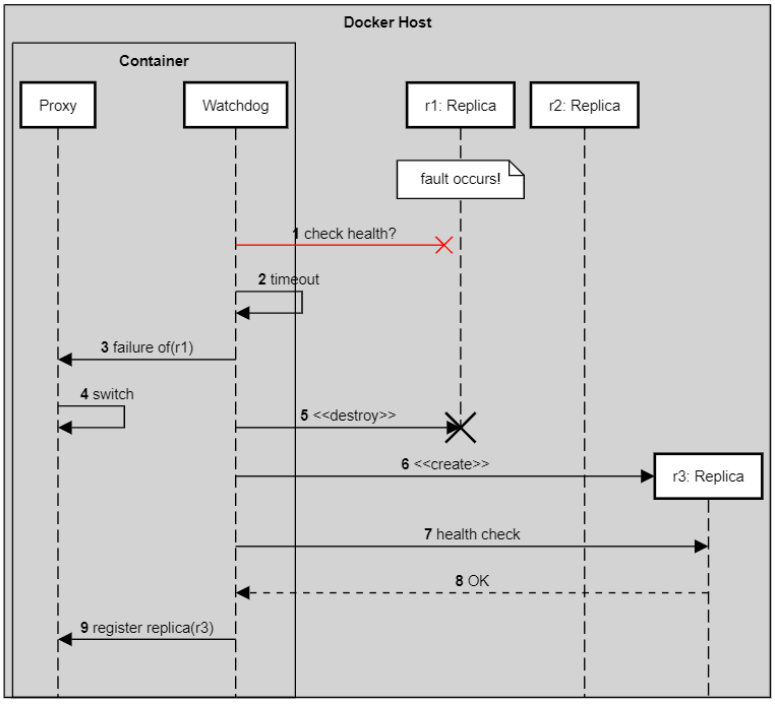


Figure 49. Masking failures of replicas.

Since a health check is an application-dependent behavior, the user must provide it as a script to be executed periodically by the watchdog. GeneSIS defines the interface of the health check script as follows. The health check must accept the endpoint of the replica to query as its sole input parameter and must output the replica status is return through its exit code, zero if the replica is healthy and any other value otherwise. This gives the user the ability to integrate any application-specific health check logic. The listing below shows one such health check script based on the HTTP status code, returned by a service.

```
#!/bin/bash
ENDPOINT="${1}"
response=$(curl --write-out '%{http_code}' --silent --output /dev/null
"${ENDPOINT}")
if [ "${response}" != 200 ]
then
    exit 1
fi
```

At the time of writing, the proxy that GeneSIS deploys is NGinx and we implemented the watchdog as two Bash scripts, deployed within the NGinx container. The first script is the watchdog and simply registers the given health check script as a CRON job that runs periodically. The second script maintains the list of existing replicas (and their endpoints) and react to events coming from the watchdogs, removing faulty replica from the list, and reconfiguring the Nginx server, by rewriting and then reloading its configuration file.

From one deployment to the next, the user may decide to modify the availability of its application, by updating the number of replicas. If, for instance, the user demands a single instance (i.e., 1 replica), fault-tolerance decreases to the fault-tolerance of the user component taken in isolation. Availability remains higher however, because GeneSIS continues to perform blue/green deployments during upgrades: It deploys the single new version aside of the previous version, before it reconfigures the proxy and to finally decommission the previous version. Upgrades are therefore transparent for the user, improving availability. The user may as well transition to a "regular" deployment without any availability mechanisms, in which case, GeneSIS destroys the proxy, the watchdog, and any replicas,

and directly expose the last replica to the incoming traffic. Alternatively, the user may decide to change the implementation, and to use Docker Swarm instead of the GeneSIS built-in mechanisms. In that case, GeneSIS stops all replicas, destroys the proxy and the watchdogs, and recreate replicas through Docker Swarm.

As explained this new feature provides GeneSIS with mechanisms to improve the reliability, resilience and availability of SIS including during their evolution (e.g., update, adaptation). In the following we detail how GeneSIS has also been extended with novel mechanisms to support the DevSecOps of a SIS thus improving its trustworthiness.

5.2.6 Dynamically adapting Smart IoT Systems for the continuous enhancement of security controls

SIS typically exposes a broad attack surface and their security must not be an afterthought. The ability to continuously evolve and adapt these systems to their new environment is decisive to ensure and increase their trustworthiness, quality, and user experience. This includes security mechanisms, which must evolve along with the SIS, continuously fixing security defects and dealing with new security threats. Following the DevSecOps principles [40], there is an urgent need for supporting the continuous deployment of SIS, including security mechanisms, over IoT, Edge, and Cloud infrastructures [41]. A key feature to enable such continuous evolution is to support the adaptation of the system having enhanced security mechanisms or updated security policies with minimal impact on the SIS already delivered and under operation. In this section, we mainly present how GeneSIS supports adapting SIS for the continuous enhancement of security controls. More details about the security controls and how GeneSIS supports the full DevSecOps cycle can be found in D2.3.

As a continuous deployment tool, GeneSIS supports for adapting SIS with updated security mechanisms in two ways: 1) according to a new development cycle; and 2) a run-time adaptation for the SIS to operate securely. This DevSecOps approach of adaptation leverages the GeneSIS' necessary mechanisms, interfaces, and abstractions to dynamically adapt the deployment and configuration of a SIS as presented earlier or in more details in D2.3. We elaborate more on the two kinds of DevSecOps adaptation support in the following paragraphs.

First, GeneSIS supports for adapting SIS with updated security mechanisms according to a new development cycle. In this line of adaptation, the SIS in operation is evolving with new business logic components or even new physical devices being added resulting in the need for enhancing security mechanisms accordingly. Let us consider a running example of a SIS that is built on the SMOOL [42] platform for smart building (Figure 50). SIS are typically built on top of IoT platforms such as SMOOL or FIWARE[23], which often act as an intermediary for the communications between the things within the IoT environment. They provide a proper ground for building mechanisms, in different applications and scenarios, to control and monitor these communications.
In the smart building system, there are IoT applications that get access to sensors' data from the smart building to make decisions and send commands to control the actuators, e.g., window blinds. The applications interact with the smart building via the SMOOL platform (in the middle). For the DevSecOps adaptation support, GeneSIS not only provides the modelling language embedded in a web UI for specifying the components of such IoT platforms, but also the reconfiguration and rebuild of these components (for integrating new security mechanisms within the IoT platform) before deployment (for adaptation or for a new development cycle). For example, in the SMOOL platform, each SMOOL producer or consumer is associated with a security checker for checking the security key of sensor data or actuation commands. GeneSIS allows updating the configuration of the security checker (e.g., by injecting new configuration to overwrite the default one), and automatically rebuilding the SMOOL producer or consumer including the reconfigured security checker. More details about this process can be found in D2.3. By doing so, GeneSIS enables the DevOps team to make reconfiguration or redevelopment and redeployment easily for the evolution of SMOOL producers or consumers including

[23] https://www.fiware.org

security checkers. This approach is what we call the DevSecOps adaptation support for the co-evolution of business logic components and the security mechanisms. This means that when new business logic components require security mechanisms to evolve, GeneSIS can support the adaptation, even including the integration of the IoT platform with other (third-party) security mechanisms. We present such an integration below for the SMOOL platform. But our approach for DevSecOps adaptation support in GeneSIS is generic and applicable for other IoT platforms.

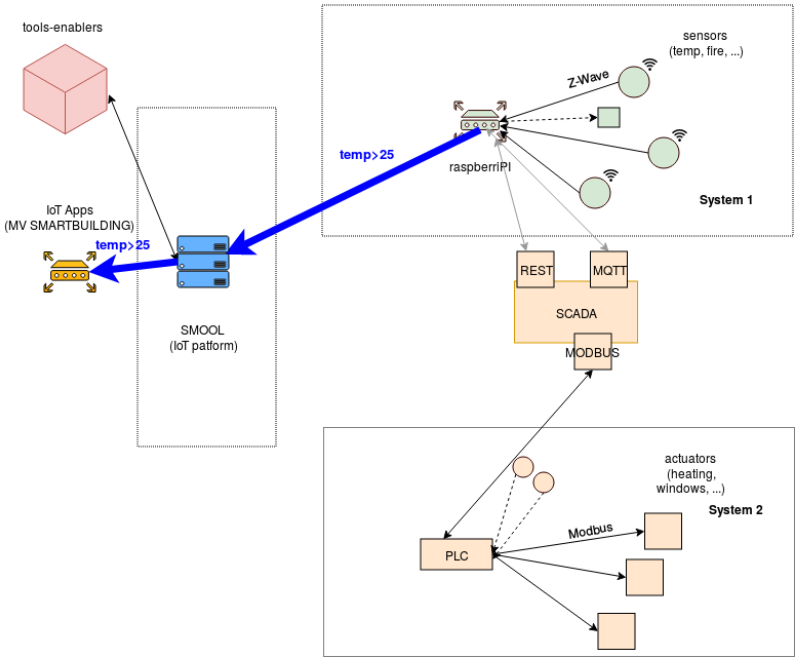


Figure 50. Tecnalia's Kubik smart building & SMOOL IoT platform

In IoT platforms like SMOOL, there are often default security enforcements. For example, the actuation orders being sent to the physical actuators in System 2 of Figure 51 must be checked before they are actually sent to the actuators. This check makes sure only genuine actuation commands can be sent to the actuators. In other words, the SMOOL platform allows to check for actuation commands with valid security tokens. All the IoT apps must send actuation commands with valid security tokens.
However, during the evolution of the smart building system, new applications can be added, and new physical devices can also be added. New security requirements come up because the smart building system must control which apps can access which actuators. This means that more fine-grained security control must be introduced, which may not be available in the IoT platform.

In this new development cycle, the DevOps team needs to introduce a new security mechanism that can enhance the fine-grained control of how the different applications can access to the sensors and actuators of the smart building. GeneSIS has a generic support for seamlessly integrating and deploying any advanced access control mechanism together with the IoT platform in use, e.g., the SMOOL platform. The DevOps team can choose to develop the required access control mechanism based on an open-source framework like Casbin[24], or the Context-based Access Control mechanism presented in D4.3, or an in-house solution.

[24] https://casbin.org/

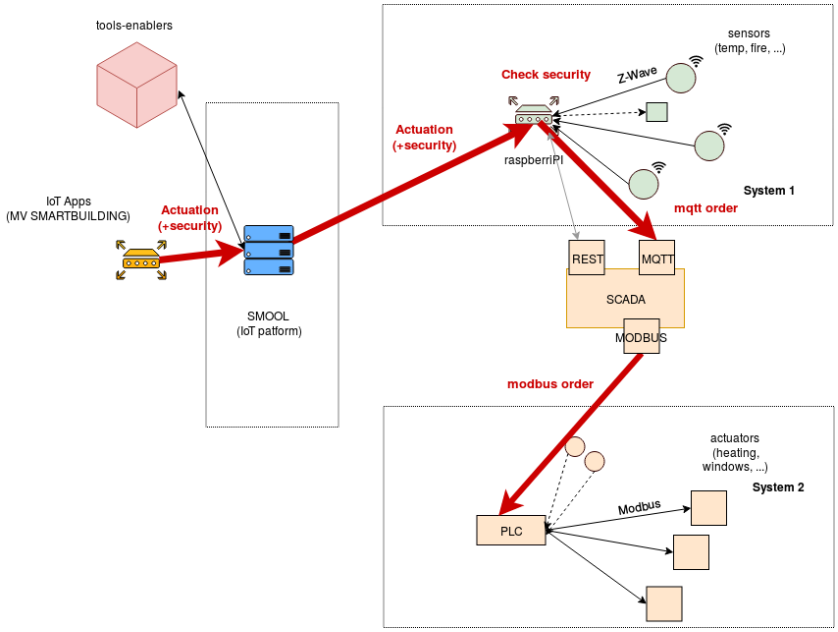


Figure 51. All actuation orders must be genuine!

Using GeneSIS, the DevOps team can specify the access control service to be deployed, e.g., Casbin. The integration of Casbin into the existing SMOOL platform is easy thanks to GeneSIS' support for dynamically extending the security checker of the SMOOL platform to become a security enforcement point of the Casbin service. Thanks to ThingML support within GeneSIS, the extended SecurityChecker is compiled and deployed so that it works as a security enforcement point of the jCasbin service. This seamless integration means that not only that the actuation commands passing through the checker must be genuine, but also, they must conform to the access control policy defined in the jCasbin service. In other words, Casbin provides a fine-grained security control, for example, that the application A can change the blind's position in the living room. We use Casbin as an example here but this way of enhancing security controls is applicable to other access control mechanisms like the Context-based Access Control mechanism in WP4.

We have shown that with the support from GeneSIS, the DevOps team can develop a new version of the smart building system together with enhanced security mechanisms according to its evolution. The deployment of this new development cycle can be triggered manually from GeneSIS's web UI. After the successful deployment, GeneSIS also allows dynamic adaptations that can be triggered at any point, manually or automatically for adapting security enforcement to improve how the IoT system operates securely. In this line of adaptation, new security policies or configurations can be updated dynamically for the security mechanisms that are in operation. For example, the role-based access control policy can be easily updated according to new requirements. The trigger of such adaptation can be manual, but also can be automatic from a risk assessment process or after a reasoning process of actuation conflict management. In both ways presented so far, GeneSIS allows DevOps teams to reconfigure and update security mechanisms by design, in line with the evolution of IoT applications and the development of security and privacy risks.

5.3 Evaluation

We evaluated our approach of adapting Smart IoT Systems, including for the continuous enhancement of security controls using the SMOOL platform, in a smart building scenario. This scenario was inspired from ENACT Smart Building use case and has served as a baseline for the evaluation of GeneSIS by the use case provider. The core change compared to the Smart Building use case is that instead of interacting directly with sensors and actuators in Kubik, these were simulated with the help of HomeIO. A main benefit for using HomeIO in these different evaluations was that it provided us with the ability to tune security policy without impacting the security of the Kubik infrastructure. Moreover, using the HomeIO simulation also allowed us to highlight the instant impact of the DevSecOps support by

GeneSIS to the evolution of the smart building system together with its security mechanisms. In our evaluation, the smart building system will be evolving from a version with the deployment model as shown in Figure 52. In this initial version, there is an *EnergyEfficiency* application, which gets access to sensors' data to make decisions for energy efficiency and send commands to control the actuators, e.g., window blinds and LED-lights. In particular, it maximizes the exploitation of daylight and regulates the in-door temperature whilst minimizing the energy consumption. If the room is bright because of daylight, it will switch off the LED-lights, and vice versa. On the other hand, if the room temperature is high, the application may need to close the window blinds to prevent sunlight heating the room. The *EnergyEfficiency* application interacts with the smart building via the SMOOL platform (in the middle). The HomeIO simulation sends sensor data from the simulation and receives actuation commands via MQTT protocol. More details on the deployment demo using HomeIO simulation can be found in this video[25].

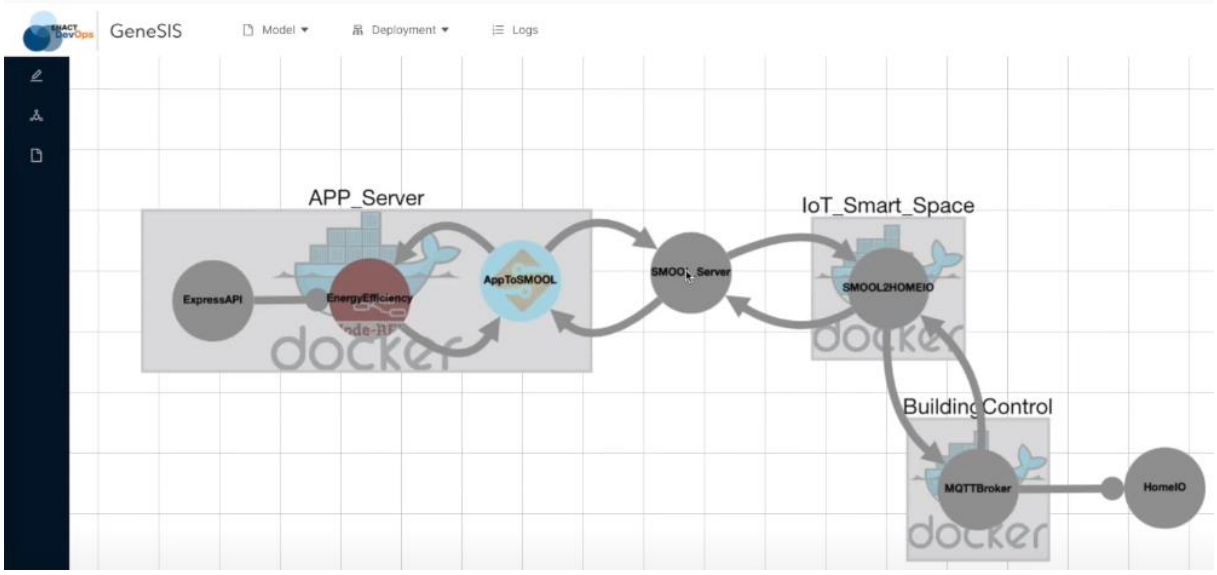


Figure 52. An initial version of a smart building system (deployment view)

There are two notable security mechanisms associated in this version of the smart building. The first one is a secure API gateway (Express Gateway[26]) that allows secure remote API access to the *EnergyEfficiency* application. The key of using Express Gateway is to define in the configuration file (gateway.config_HomeIO.yml) how the API of the *EnergyEfficiency* application is securely exposed. The second is the security checker in the *SMOOL2HOMEIO* component that makes sure only genuine actuation commands can be sent to HomeIO. In other words, the SMOOL platform allows to check for actuation commands with valid security tokens. Every actuation order must be accompanied by a valid security key before executing the order as specified in the deployment model below.

```
{
   "_type": "/internal/SMOOLSecurity",
   "name": "SecurityEnforcer",
    ...
    "security_Policy": [["BlindPositionActuator", "Authorization"]],
    ...
}
```

In the subsequent development cycle, another application called *UserComfortApp* has been added to the smart building system. Moreover, the smart building system can also have new IoT devices such as *AirQualitySensor* or *SmartDisplay* as shown in Figure 53.

[25] https://youtu.be/yQ9XYWu-EZM
[26] https://www.express-gateway.io/

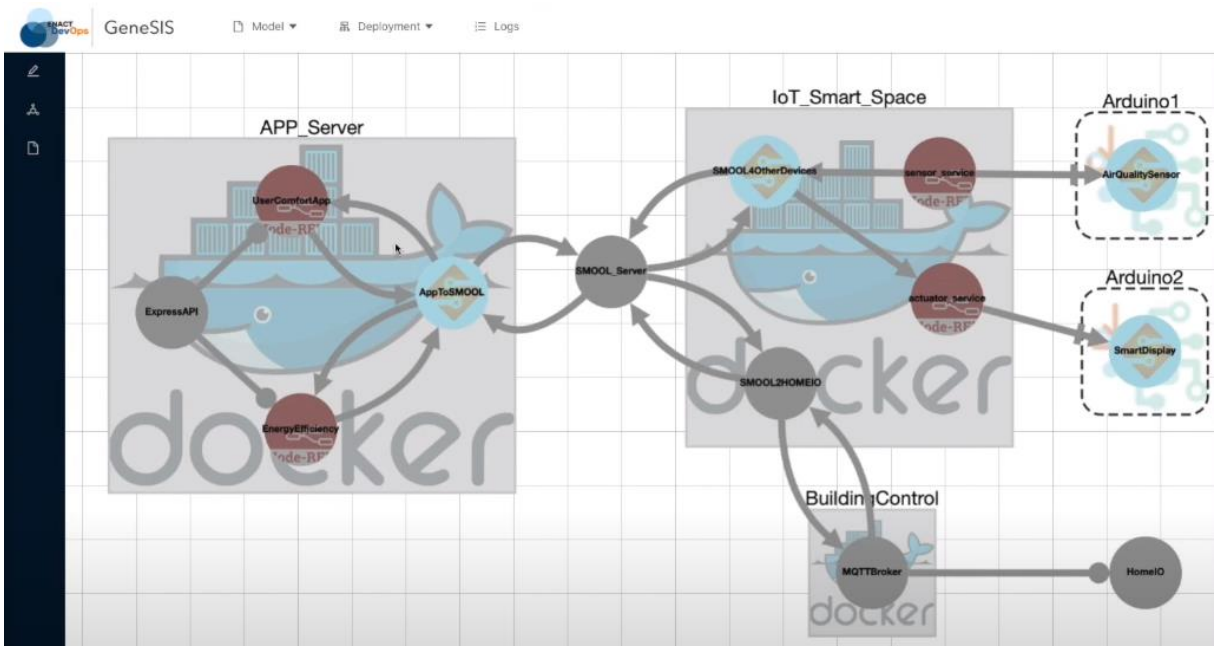


Figure 53. New applications and new IoT devices can be added in a development cycle

In this new development cycle, not only the secure API gateway must be updated with a new configuration file, but also the DevOps team needs to introduce a new security mechanism that can enhance the fine-grained control of how the two applications can access to the sensors and actuators of the smart building. Updating the configuration file (*gateway.config_HomeIO.yml*) for the API gateway is straightforward to enable secure remote API access to both applications *EnergyEfficiency* and *UserComfortApp*. However, to address the requirement of providing more fine-grained control of how the two applications can access to the sensors and actuators of the smart building, the DevOps team need to develop and deploy an advanced access control mechanism.

GeneSIS has a generic support for seamlessly integrating and deploying any advanced access control mechanism together with the IoT platform in use, e.g., the SMOOL platform. The DevOps team can choose to develop the required access control mechanism based on an open-source framework like Casbin[27], or the Context-based Access Control mechanism presented in D4.3, or an in-house solution. Using GeneSIS, the DevOps team can specify the access control service to be deployed, e.g., jCasbin in Figure 54. The integration of jCasbin into the existing SMOOL platform is easy thanks to GeneSIS' support for dynamically extending the security checker in the *SMOOL2HOMEIO* component to become a security enforcement point of the jCasbin service. This seamless integration means that not only that the actuation commands passing through the *SMOOL2HOMEIO* must be genuine, but also, they must conform to the access control policy defined in the jCasbin service. In other words, jCasbin provides a fine-grained security control, for example, that the *EnergyEfficiency* application is allowed to change the blind's position in the living room as can be seen in our video demo[28]. We use jCasbin as an example here but this way of enhancing security controls is applicable to other access control mechanisms like Context-based Access Control mechanism.

[27] https://casbin.org/
[28] https://youtu.be/yQ9XYWu-EZM

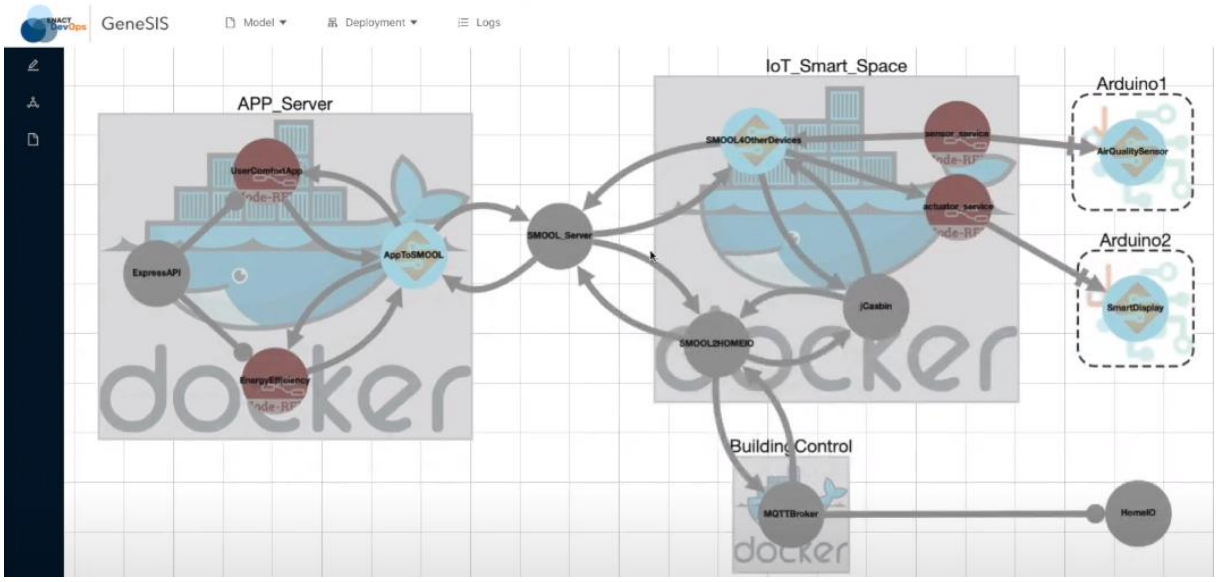


Figure 54. An enhanced security control has been added

Figure 55 shows the GeneSIS's UI for extending the security checker in the *SMOOL2HOMEIO* component to become a security enforcement point of the jCasbin service. Thanks to ThingML support within GeneSIS, the extended *SecurityChecker* is compiled in the *SMOOL2HOMEIO* component for a new version of *SMOOL2HOMEIO* to be deployed that works as a security enforcement point of the jCasbin service.

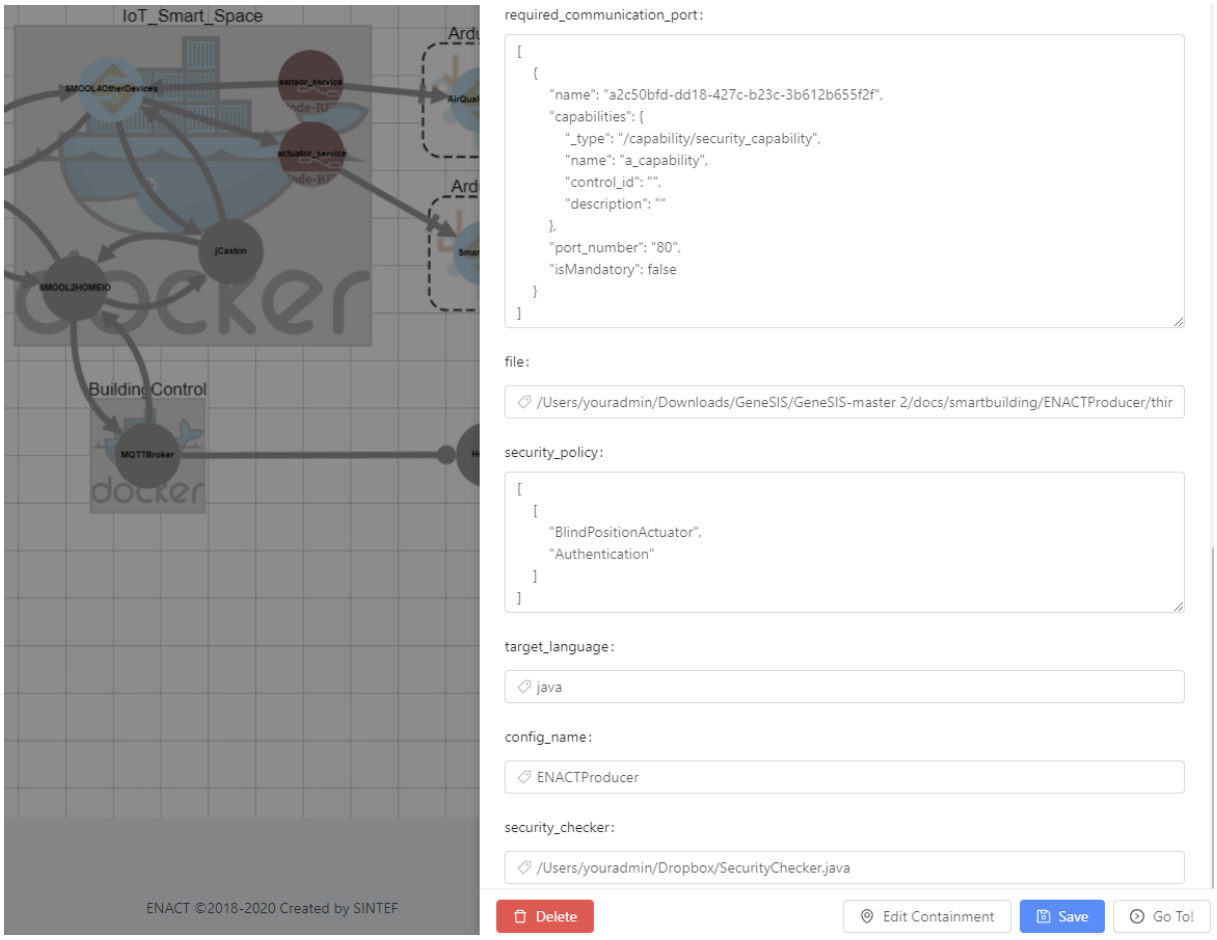


Figure 55. Extend the SecurityChecker.java of the SMOOL2HOMEIO to become an enforcement point for the Casbin service

After the successful deployment (Figure 56), GeneSIS allows dynamic adaptations that can be triggered at any point, manually or automatically for adapting security enforcement to improve how the IoT system operates securely.

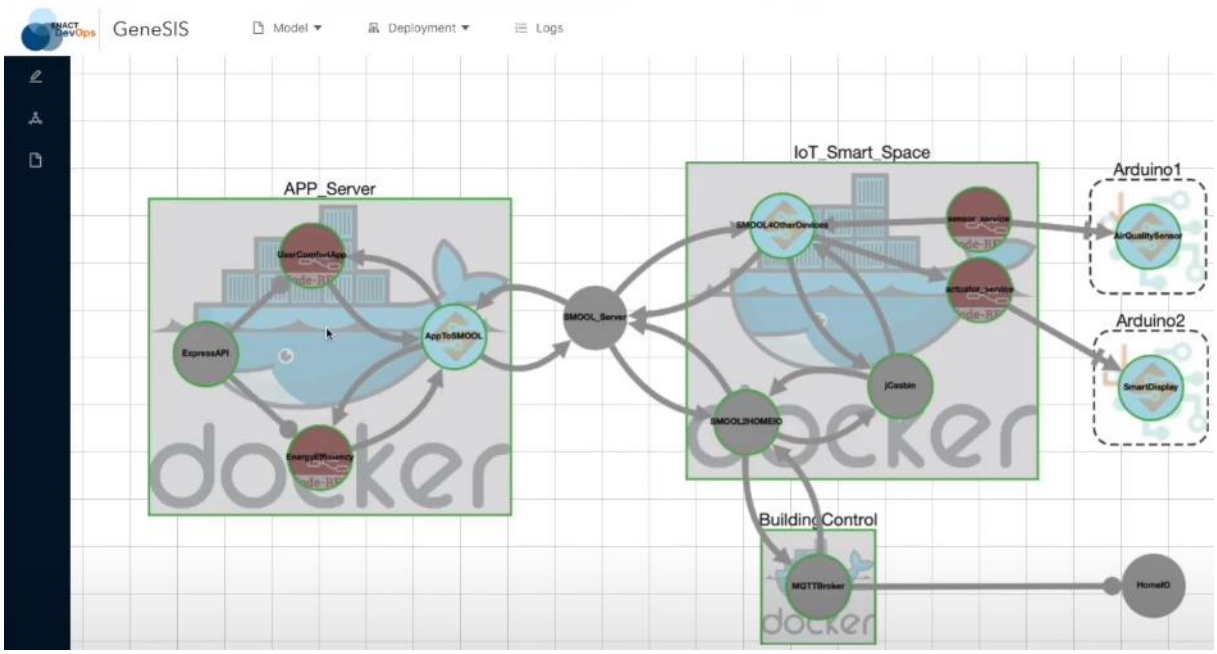


Figure 56. Deployment with success and dynamic adaptations can be triggered at any point

Second, GeneSIS supports for adapting security enforcement to improve how the IoT system operates securely. In this line of adaptation, new security policies or configurations can be updated dynamically for the security mechanisms that are in operation. For example, new gateway configuration can be updated to change how the APIs of the application are accessible as can be seen in the SSH resource declaration below.

```
ssh_resource:
{
   "name": "52a12bfa-0-44d9-9358-f55b9dcc36e5",
   "uploadCommand": [
      "/Users/youradmin/Dropbox/gateway.config.yml",
      "/home/pi/gateway.config.yml"
   ],
```

Similarly, the role-based access control policy can be easily updated according to new requirements. The trigger of such adaptation can be manual, but also can be automatic from a risk assessment process or after a reasoning process of actuation conflict management.

```
ssh_resource:
{
   "name": "52a12bfa-075f-44d9-9358-f55booodcc36e5",
   "uploadCommand": [],
   "startCommand": "cd  jcasbin; mvn spring-boot:run -Dspring-boot.run.arguments='--
casbin.authorization.model=src/main/resources/rbac_model.conf --
casbin.authorization.policy=src/main/resources/rbac_policy.csv --server.port=8011' &> jcasbin.txt",
```

In both ways presented so far, GeneSIS allows DevOps teams to reconfigure and update security mechanisms by design, in line with the evolution of IoT applications and the development of security and privacy risks.

5.4 Beyond ENACT

In the future we will investigate extending GeneSIS and its deployment engine in the following directions. A first objective will be to improve the integration between GeneSIS and the DivENACT executions engine. At the time of writing, the two enablers are integrated as follows. DivENACT is used to manage a fleet of system whilst GeneSIS is exploited to enact deployment on a single system, in particular when the system includes a local network. To do so, DivENACT deploys GeneSIS on a device in the local network yet accessible from outside. Currently the integration is limited in the sense that the runtime information maintained by the GeneSIS deployment engine is not available and thus exploited by DivENACT during the fleet management process. This is particularly important as within a fleet, each system is typically operating in its own context. It is critical to monitor and understand this context in order to adapt and maintain each system within a fleet. GeneSIS is a good candidate for providing such context information at a system level. In the future, and in particular in the context of the FLEET NRC project that started in 2020 and will end in 2024, we will explore how this runtime information can be shared between the two deployment engines and how it can be best exploited by the two tools. This will also include exploring what another runtime information should be considered and gathered. On the contrary, the support for dynamic adaptation offered by GeneSIS may benefit DivENACT, for instance, DivENACT on the basis of fleet level goals may need to trigger system level adaptations.

6 Interplay of Enablers

6.1 Interplay among OLE and BDA

A possible integration has been identified between Online Learning and Behavioural Drift Analysis. Throughout this chapter we are going to explain the conceptual idea underlying this integration and an example used to evaluate the interplay of these two enablers within the Smart Building Use Case.

6.1.1 Conceptual idea

As it is the main goal of the OLE to learn an optimal control strategy for a Markov Decision Process with means of Reinforcement Learning, the actual control strategy can not only be evaluated from the RL perspective in terms of Reward. Furthermore, the control strategy needs to be evaluated from a domain perspective. This domain perspective also plays an important role when engineering the reward function. Therefore, models can be obtained based on the desired behaviour of the optimal control strategy and the underlying reward function. The idea underpinning the integration of OLE and BDA, is using such a model capturing the desired behaviour (or parts of it) the OLE approach should learn on a higher level and use it as an input for the behavioural drift computation. Then using the Behavioural drift signal the learned behaviour could be evaluated based on a single numerical value and it could be easily seen whether a drift in the behaviour of the control strategy has emerged (e.g. based on a change in the context, making it impossible for the learning system to behave according to the obtained model.). Future work could focus on using the behavioural drift signal directly as input in terms of reward for the Reinforcement Learning approach.

6.1.2 Evaluation in the use case

As described in Section 2 the main goal of the OLE within in the Smart Building Use Case was to learn an optimal control strategy for the HVAC system, by optimizing the thermal comfort and minimizing the energy consumption. Based on the reward function used to compute the thermal comfort reward (cf. Section 2.3.2) the following model was derived as input for the BDA:

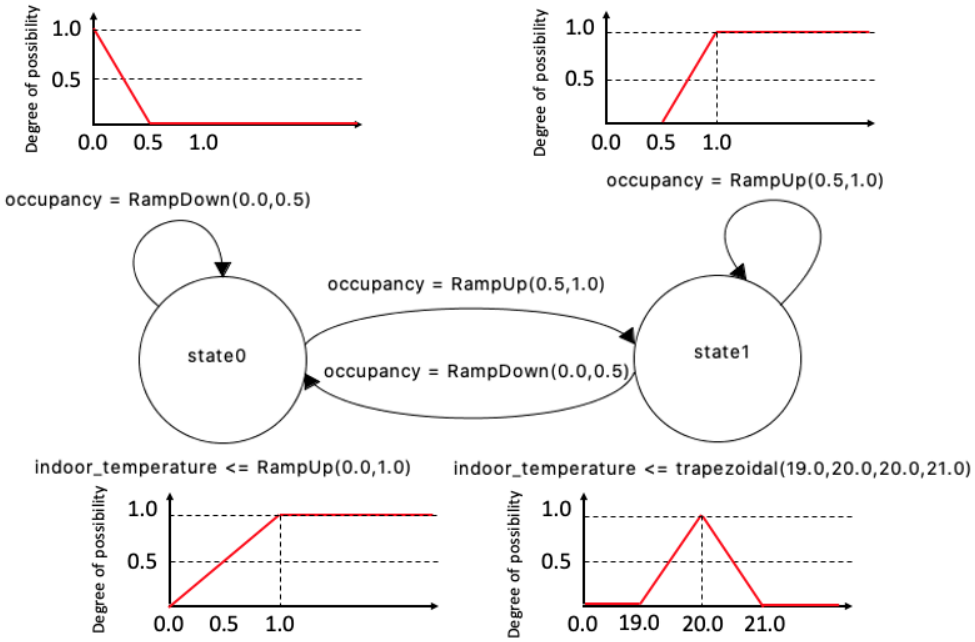


Figure 57: Possibilistic Input/Output Hidden Markov Model describing the expected indoor temperature depending on whether a person is present or no in a room.

The behavioural model depicted in Figure 57 represents the expected indoor temperature which depends on whether or not a person is present in the room and this, independently of the underlying temperature management system. It relies on the possibility theory where distributions are defined as membership functions. The model defines the expected behaviour as follows: when a person is present in the room, the temperature must be equal to 20°C (state#1). When nobody is in the room, the temperature is expected to be greater than 1°C (state#0). In addition to fully accepted temperature values (where degree of possibility is equal to one), the model defines some tolerances. For instance, for the state#1, temperatures below 19.0°C and above 21°C are totally rejected (degree of possibility = 0) while temperatures in between 19.0°C and 21.0°C different from 20°C, while not being perfect, are not totally rejected (0 < degree of possibility < 1).

In conjunction with temperature and occupancy sensor values, this possibilistic Input/Output Hidden Markov Model (IOHMM) is used to compute the behavioural drift as the likelihood (possibility measure) of the observation sequences to have been generated by the model i.e. the likelihood that the temperature is managed in such a way that it remains within the accepted boundaries defined by the model.

For the realization of the integration the simulation data used by the Online learning tool was published via MQTT and used by the BDA-tool to compute the behavioural drift value. Then this value was published again via MQTT. The Online learning tool was subscribed to the BDA-topic and visualizes the evolution of the behavioural drift value in the monitoring pane (cf. Figure 58).

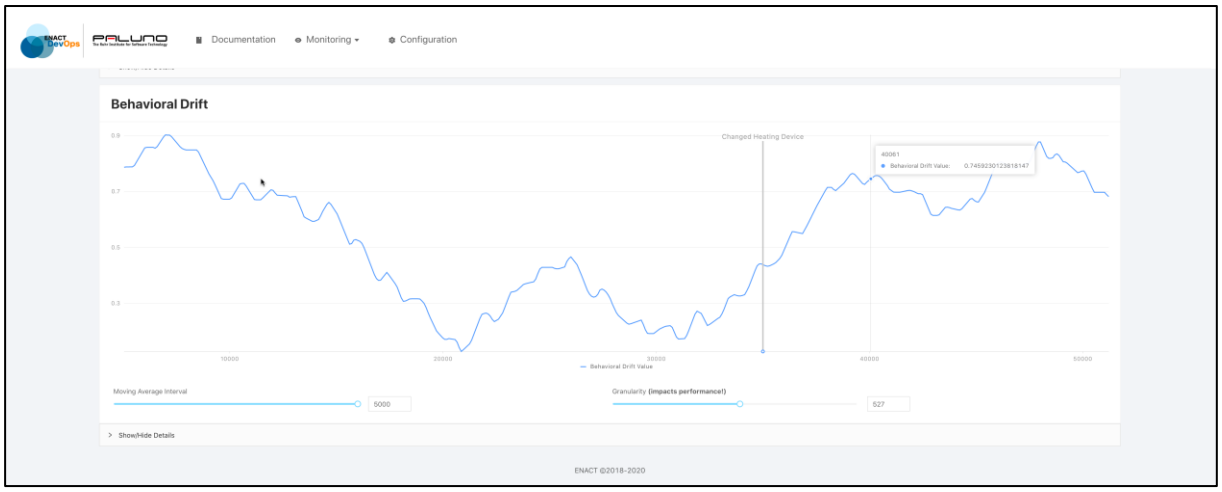


Figure 58: Screenshot of Online learning tool showing behavioural drift value

6.2 Interplay among RCA and BDA

The BDA enabler described in Section 3 is an ideal companion of the RCA enabler described in Section 4. By continuously providing designers with behavioral drifts alerts and their associated symptoms, the BDA enabler may help RCA to improve the historical data. On the basis of the symptoms provided, designers can correlate various events and consult system experts to determine the corresponding root causes as well as its relevant data. In the monitoring phase of the RCA Enabler, as depicted in Figure 59, BDA can signalize its detected misbehaviour so that RCA collects the related metrics/ logs/ traffic (i.e., symptoms), compares with the knowledge in the historical data to determine the corresponding known incident. If no learned incident is found sufficiently similar, the manual analysis with the help of experts needs to be performed to update the historical data with the newly detected incident.
At the time of writing D3.3, the evaluation with the ITS use case has not been fully finished. This interplay will be presented in D1.4 and D1.5 of WP1- Case Studies and Validation.

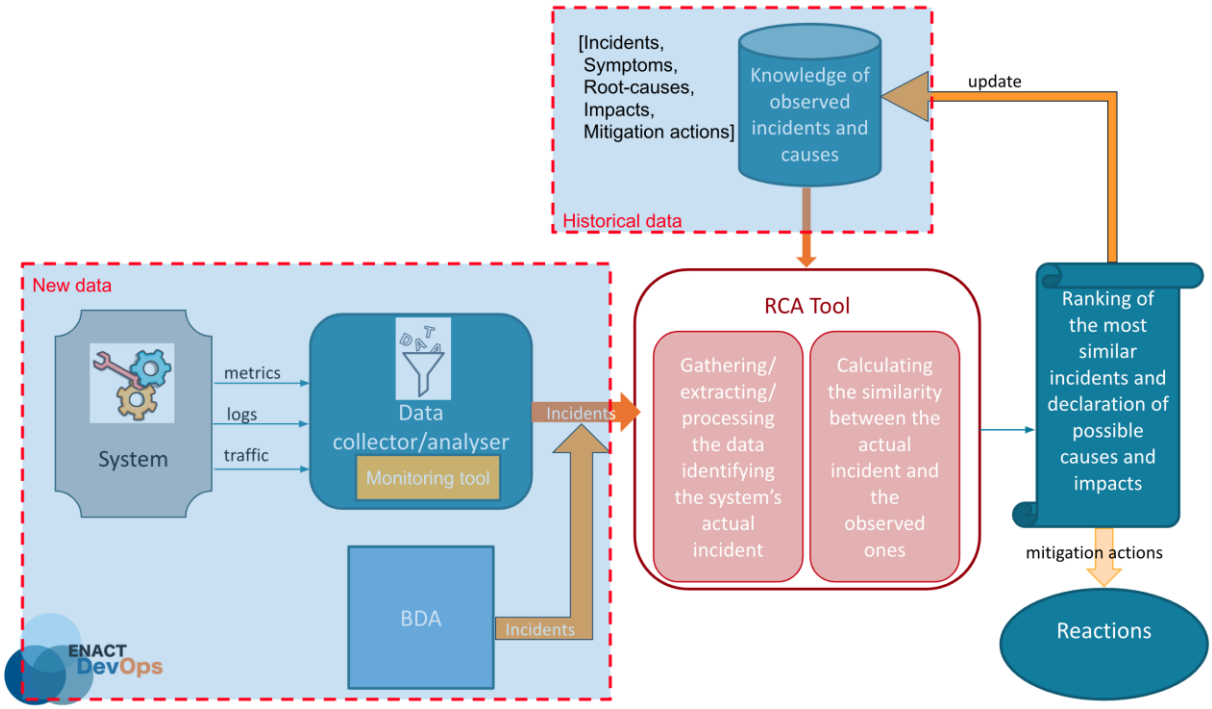


Figure 59: Interplay among RCA and BDA in analysing the new data

7 Conclusion

WP3 aimed at developing enablers for the operational part of the DevOps process. As final outcomes, WP3 provides enablers that provide IoT systems with capabilities to (i) monitor their status, (ii) indicate whether they behave as expected or not, (iii) identify the origin of the problem, and (iv) automatically perform typical operation activities (including self-adaptation of the systems). The main focus of this deliverable was the description of the different enabler solutions. Based on the conceptual designs described in previous deliverables D3.1 & D3.2, the deliverable at hand provided a final set of tools realizing these conceptual approaches. Furthermore, a brief summary concerning the evaluation of the different enablers was given, which is complemented by the use case evaluation described in D1.5. Finally, possible integrations between the different WP3 enablers have been described.

8 References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra und M. Riedmiller, „Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[2] A. Palm, A. Metzger und K. Pohl, „Online Reinforcement Learning for Self-adaptive Information Systems," in *International Conference on Advanced Information Systems Engineering*, 2020.

[3] A. Metzger, T. Kley und A. Palm, „Triggering Proactive Business Process Adaptations via Online Reinforcement Learning," in *International Conference on Business Process Management*, 2020.

[4] A. Metzger, C. Quinton, Z. A. Mann, L. Baresi und K. Pohl, „Feature model-guided online reinforcement learning for self-adaptive services," in *International Conference on Service-Oriented Computing*, 2020.

[5] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto und N. Dormann, *Stable Baselines3*, GitHub, 2019.

[6] E. Barrett und S. Linder, „Autonomous hvac control, a reinforcement learning approach," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2015.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford und O. Klimov, „Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[8] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup und D. Meger, „Deep reinforcement learning that matters," *arXiv preprint arXiv:1709.06560*, 2017.

[9] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor und Y. Wu, *Stable Baselines*, GitHub, 2018.

[10] R. S. Sutton, D. A. McAllester, S. P. Singh und Y. Mansour, „Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000.

[11] D. G. D. L. Iglesia und D. Weyns, „MAPE-K formal templates to rigorously design behaviors for self-adaptive systems," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, Bd. 10, p. 1–31, 2015.

[12] Council of European Union, *Council regulation (EU) no 2016/679*, 2016.

[13] T. G. Dietterich, „Hierarchical reinforcement learning with the MAXQ value function decomposition," *Journal of artificial intelligence research*, Bd. 13, p. 227–303, 2000.

[14] S. J. Russell und A. Zimdars, „Q-decomposition for reinforcement learning agents," in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 2003.

[15] Z. Lin, L. Zhao, D. Yang, T. Qin, T.-Y. Liu und G. Yang, „Distributional Reward Decomposition for Reinforcement Learning," in *Advances in Neural Information Processing Systems*, 2019.

[16] G. Rocher, J.-Y. Tigli, S. Lavirotte und N. L. Thanh, „Overview and Challenges of Ambient Systems, Towards a Constructivist Approach to their Modelling," *arXiv preprint arXiv:2001.09770*, 2020.

[17] T. Ekanayake, D. Dewasurendra, S. Abeyratne, L. Ma und P. Yarlagadda, „Model-based fault diagnosis and prognosis of dynamic systems: a review," *Procedia Manufacturing*, Bd. 30, p. 435–442, 2019.

[18] Y. Bengio und P. Frasconi, „An input output HMM architecture," in *Advances in neural information processing systems*, 1995.

[19] P. Weber und C. Simon, Benefits of Bayesian network models, John Wiley & Sons, 2016.

[20] R. J. G. B. Campello, D. Moulavi und J. Sander, „Density-based clustering based on hierarchical density estimates," in *Pacific-Asia conference on knowledge discovery and data mining*, 2013.

[21] M. Dell'Amico, „FISHDBC: Flexible, Incremental, Scalable, Hierarchical Density-Based Clustering for Arbitrary Data and Distance," *arXiv preprint arXiv:1910.07283*, 2019.

[22] L. R. Rabiner, „A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE,* Bd. 77, p. 257–286, 1989.

[23] A. K. S. Jardine, D. Lin und D. Banjevic, „A review on machinery diagnostics and prognostics implementing condition-based maintenance," *Mechanical systems and signal processing,* Bd. 20, p. 1483–1510, 2006.

[24] A. K. Jain, M. N. Murty und P. J. Flynn, „Data clustering: a review," *ACM computing surveys (CSUR),* Bd. 31, p. 264–323, 1999.

[25] A. Gepperth und B. Hammer, „Incremental learning algorithms and applications," 2016.

[26] D. Azzalini, A. Castellini, M. Luperto, A. Farinelli und F. Amigoni, „HMMs for Anomaly Detection in Autonomous Robots," in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems,* 2020.

[27] Y. Chen, J. Ye und J. Li, „Aggregated Wasserstein Distance and State Registration for Hidden Markov Models," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 2019.

[28] M. Pelillo, Similarity-based pattern analysis and recognition, Springer, 2013.

[29] R. Lowry, „Concepts and applications of inferential statistics," 2014.

[30] M. Kuhn und K. Johnson, Feature engineering and selection: A practical approach for predictive models, CRC Press, 2019.

[31] A. Bellet, A. Habrard und M. Sebban, Metric Learning, Similarity-Based Pattern Analysis and Recognition, Morgan and Claypool, 2015.

[32] P. H. Nguyen, N. Ferry, G. Erdogan, H. Song, S. Lavirotte, J.-Y. Tigli und A. Solberg, „The preliminary results of a mapping study of deployment and orchestration for IoT," *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing,* pp. 2040-2043, 2019.

[33] P. H. Nguyen, N. Ferry, G. Erdogan, H. Song, S. Lavirotte, J.-Y. Tigli und A. Solberg, „A Systematic Mapping Study of Deployment and Orchestration Approaches for IoT," in *Proceedings of the 4th International Conference on Internet of Things, Big Data and Security - Volume 1: IoTBDS*, SciTePress, 2019, pp. 69-82.

[34] P. H. Nguyen, N. Ferry, G. Erdogan, H. Song, S. Lavirotte, J.-Y. Tigli und A. Solberg, „Advances in Deployment and Orchestration Approaches for IoT - A Systematic Review," in *2019 IEEE International Congress on Internet of Things (ICIOT)*, 2019, pp. 53-60.

[35] N. Ferry, P. Nguyen, H. Song, P.-E. Novac, S. Lavirotte, J.-Y. Tigli und A. Solberg, „Genesis: Continuous orchestration and deployment of smart iot systems," in *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, 2019.

[36] D. Farley und J. Humble, Continuous Delivery, Addison-Wesley, 2011.

[37] N. Ferry und P. H. Nguyen, „Towards model-based continuous deployment of secure IoT systems," in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 2019, pp. 613-618.

[38] G. S. Blair, N. Bencomo und R. B. France, „Models@run.time," *IEEE Computer,* Bd. 42, pp. 22-27, 2009.

[39] L. Bass, P. Clements und R. Kazman, Software architecture in practice, Addison-Wesley Professional, 2003.

[40] H. Myrbakken und R. Colomo-Palacios, „DevSecOps: A Multivocal Literature Review," in *Software Process Improvement and Capability Determination*, Cham, 2017.

[41] A. M. et al, „Cyber physical systems: Opportunities and challenges for software, services, cloud and data," 2015.

[42] A. Noguero, A. Rego und S. Schuster, „Towards a Smart Applications Development Framework," *Social Media and Publicity,* Bd. 27, 2014.

Appendix

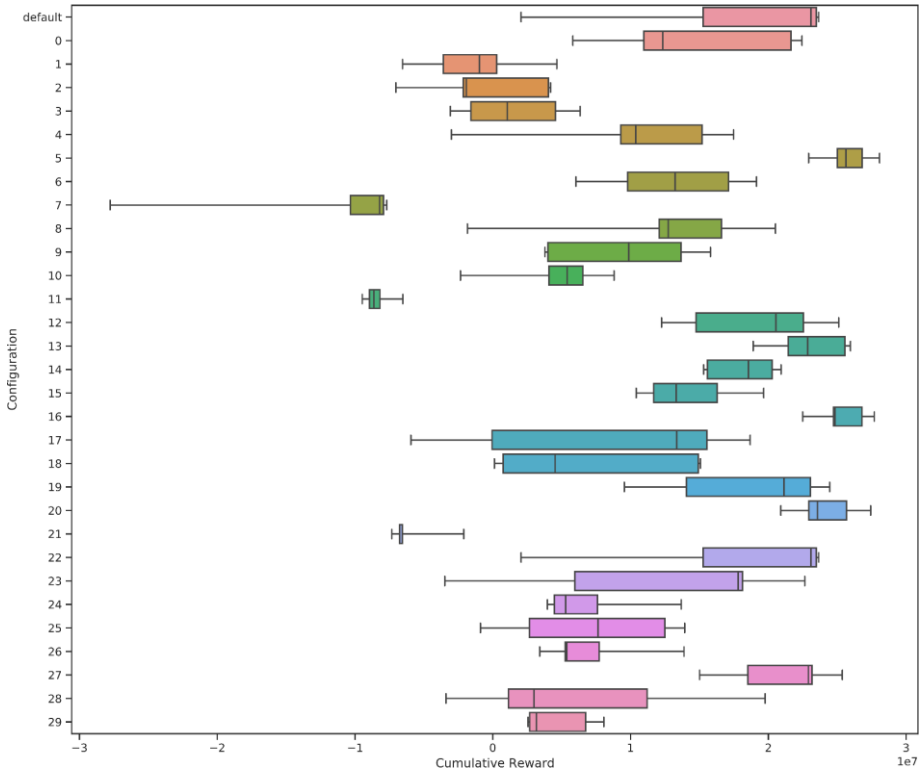


Figure 60: Cumulative Reward of different configurations

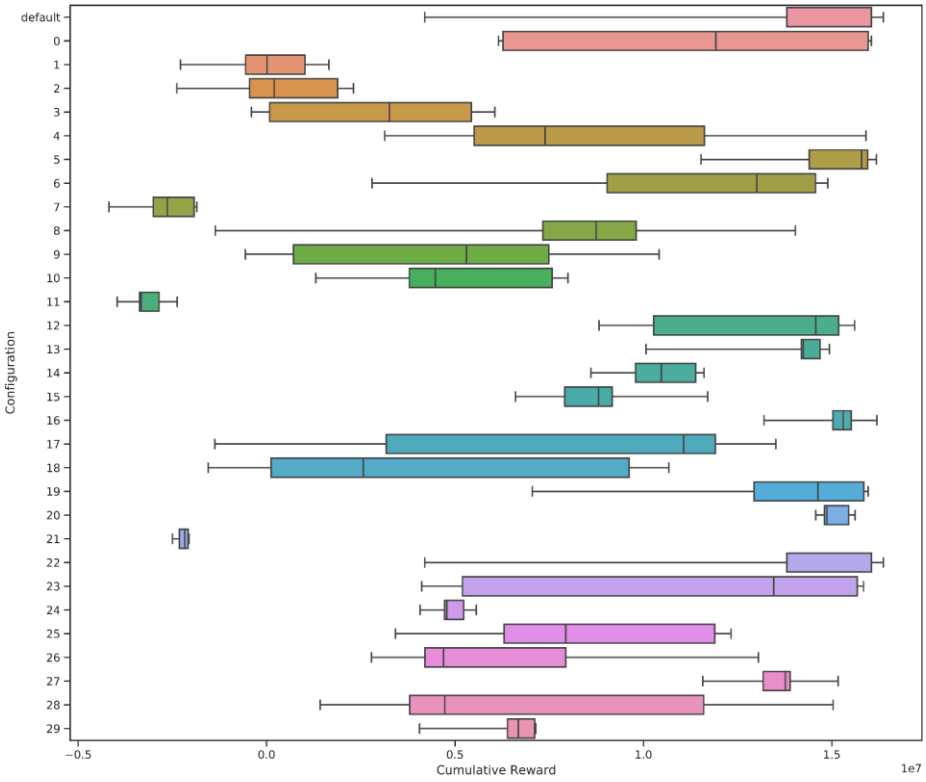


Figure 61: Cumulative Reward of different configurations (after 250.000 timesteps)

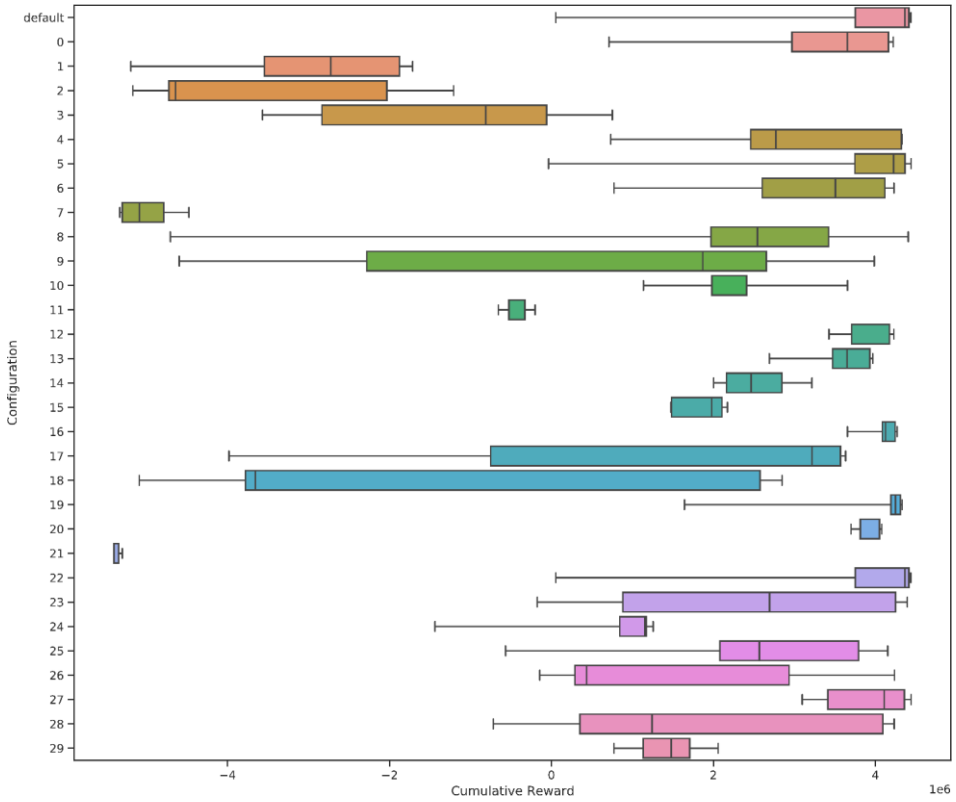


Figure 62: Cumulative Reward of different configurations (after 450.000 timesteps)

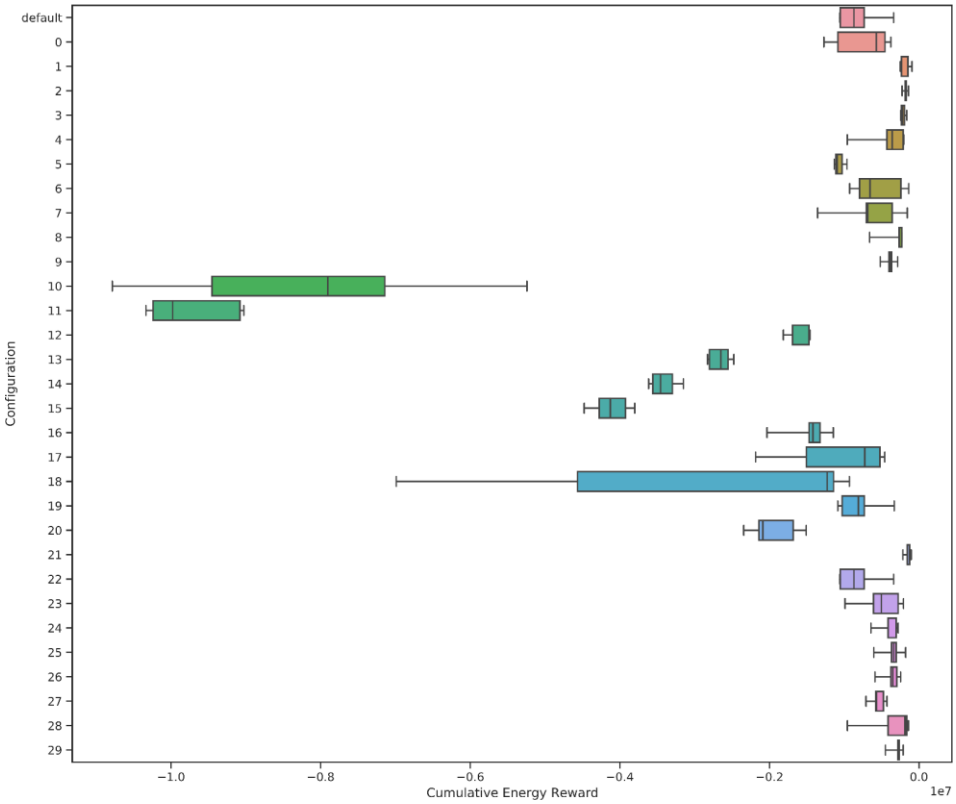


Figure 63. Cumulative Energy Reward of different configurations

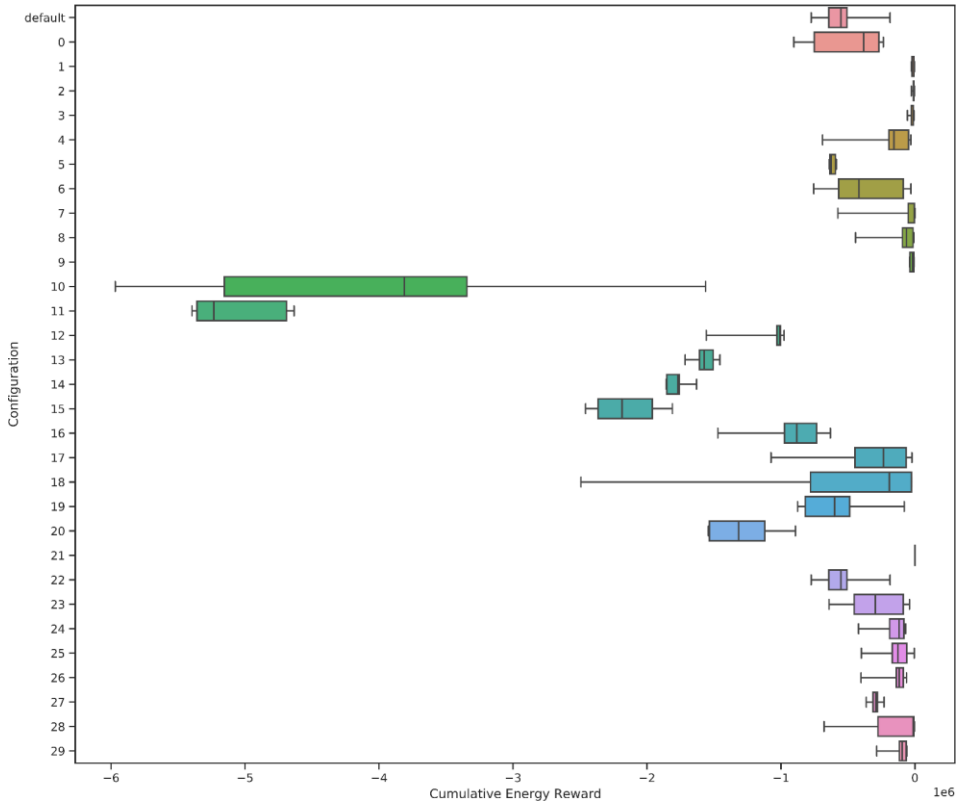


Figure 64: Cumulative Energy Reward of different configurations (after 250.000 timesteps)

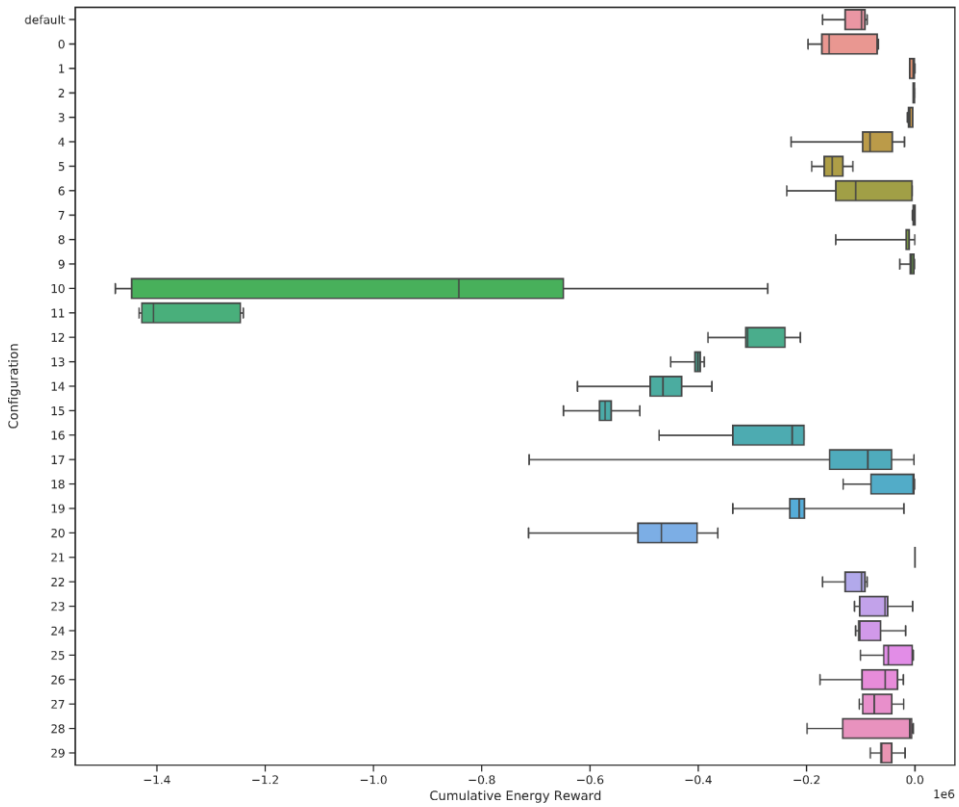


Figure 65: Cumulative Energy Reward of different configurations (after 450.000 timesteps)

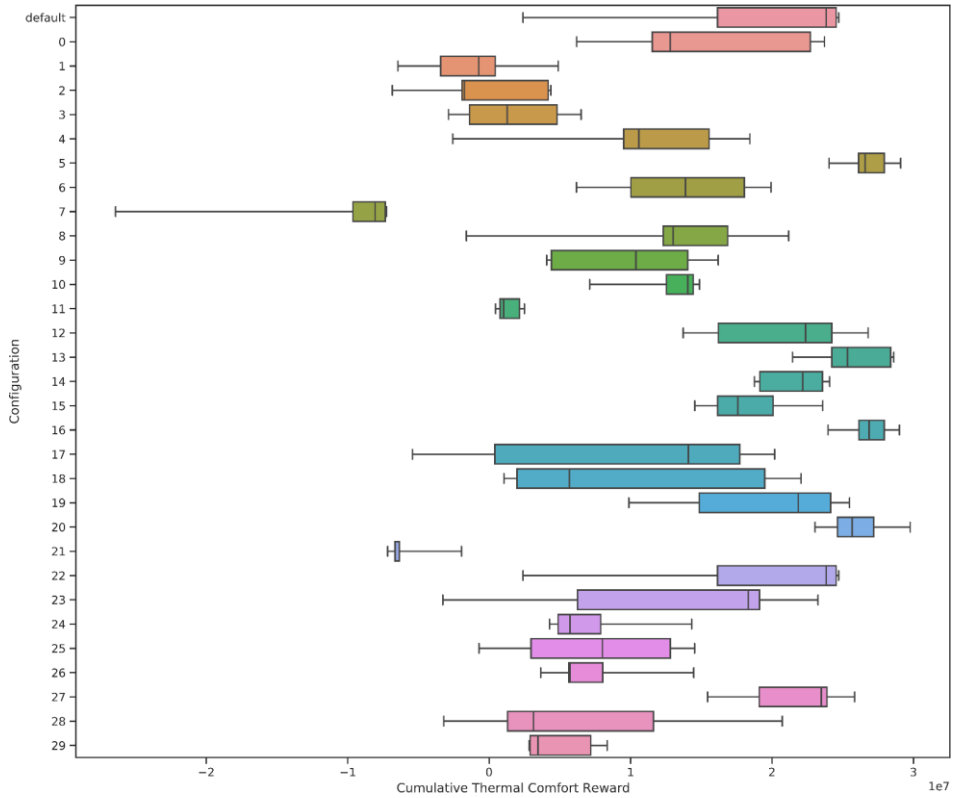


Figure 66: Cumulative Thermal Comfort Reward of different configurations

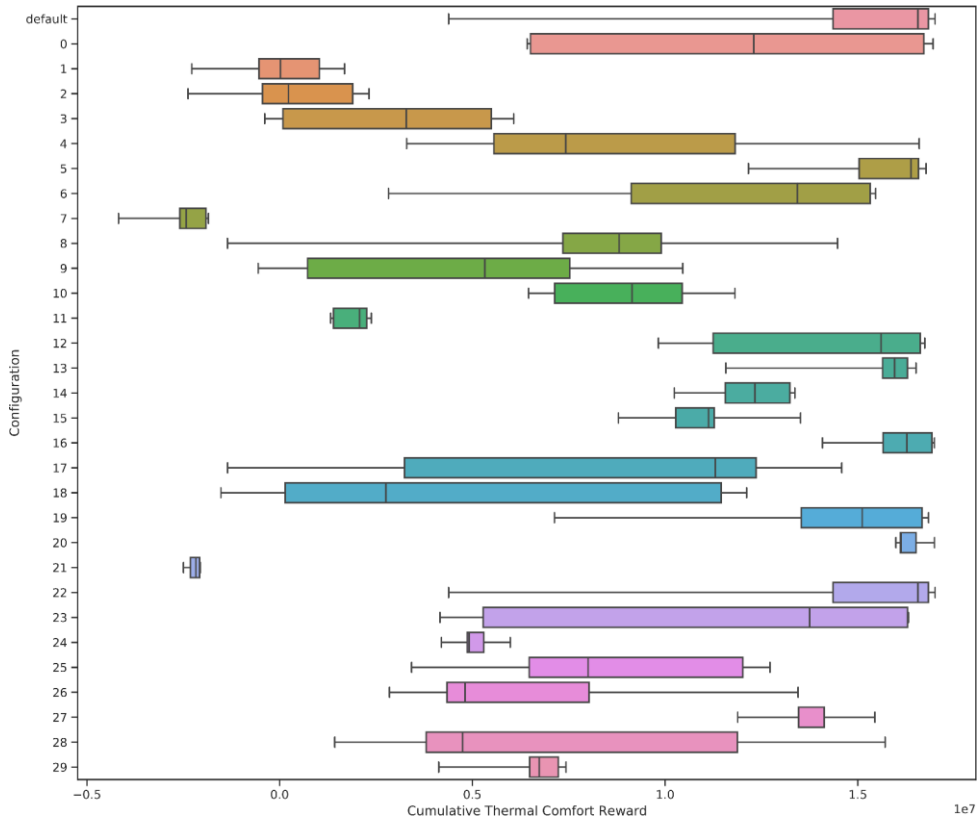


Figure 67: Cumulative Thermal Comfort Reward of different configurations (after 250.000 timesteps)

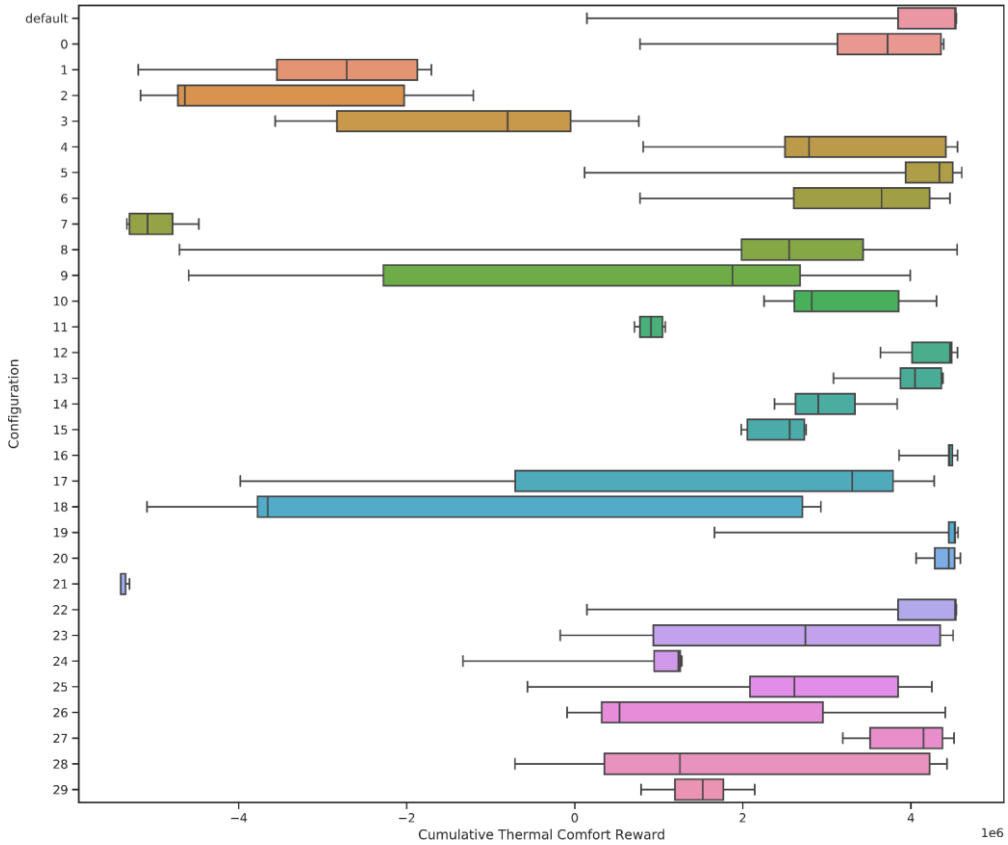


Figure 68: Cumulative Thermal Comfort Reward of different configurations (after 450.000 timesteps)

configuration	**parameter: value (different from default parameter-values)**
0	layer_size: 128
1	layer_number: 3
2	layer_number: 4
3	layer_number: 5
4	shared_layers: 1
5	activation_fn: relu
6	learning_rate: 0.0003
7	learning_rate: 0.003
8	n_steps: 256
9	n_steps: 512
10	n_steps: 1024
11	n_steps: 2048
12	batch_size: 8
13	batch_size: 16
14	batch_size: 32

15	batch_size: 64
16	n_epochs: 10
17	gae_lambda: 0.99
18	gae_lambda: 1
19	clip_range: 0.1
20	clip_range: 0.02
21	ent_coef: 0.0
22	target_kl: 0.01
23	env_seed: 903450155
24	env_seed: 1377776711
25	env_seed: 2522572000
26	env_seed: 3952781742
27	reward_scaling: 0.01
28	reward_scaling: 0.1
29	obs_space_norm: True

Table 22: Overview of all tested parameter configurations

configuration	**layer_size**	**layer_number**	**shared_layer**	**activation_fn**
0	4	1	None	tanh
1	4	1	None	relu
2	4	1	1	tanh
3	4	1	1	relu
4	4	2	None	tanh
5	4	2	None	relu
6	4	2	1	tanh
7	4	2	1	relu
8	4	3	None	tanh
9	4	3	None	relu
10	4	3	1	tanh
11	4	3	1	relu
12	4	4	None	tanh
13	4	4	None	relu
14	4	4	1	tanh
15	4	4	1	relu

16	4	5	None	tanh
17	4	5	None	relu
18	4	5	1	tanh
19	4	5	1	relu
20	8	1	None	tanh
21	8	1	None	relu
22	8	1	1	tanh
23	8	1	1	relu
24	8	2	None	tanh
25	8	2	None	relu
26	8	2	1	tanh
27	8	2	1	relu
28	8	3	None	tanh
29	8	3	None	relu
30	8	3	1	tanh
31	8	3	1	relu
32	8	4	None	tanh
33	8	4	None	relu
34	8	4	1	tanh
35	8	4	1	relu
36	8	5	None	tanh
37	8	5	None	relu
38	8	5	1	tanh
39	8	5	1	relu
40	16	1	None	tanh
41	16	1	None	relu
42	16	1	1	tanh
43	16	1	1	relu
44	16	2	None	tanh
45	16	2	None	relu
46	16	2	1	tanh
47	16	2	1	relu
48	16	3	None	tanh
49	16	3	None	relu
50	16	3	1	tanh
51	16	3	1	relu
52	16	4	None	tanh
53	16	4	None	relu

54	16	4	1	tanh
55	16	4	1	relu
56	16	5	None	tanh
57	16	5	None	relu
58	16	5	1	tanh
59	16	5	1	relu
60	32	1	None	tanh
61	32	1	None	relu
62	32	1	1	tanh
63	32	1	1	relu
64	32	2	None	tanh
65	32	2	None	relu
66	32	2	1	tanh
67	32	2	1	relu
68	32	3	None	tanh
69	32	3	None	relu
70	32	3	1	tanh
71	32	3	1	relu
72	32	4	None	tanh
73	32	4	None	relu
74	32	4	1	tanh
75	32	4	1	relu
76	32	5	None	tanh
77	32	5	None	relu
78	32	5	1	tanh
79	32	5	1	relu

Table 23: Overview of all tested architecture configurations

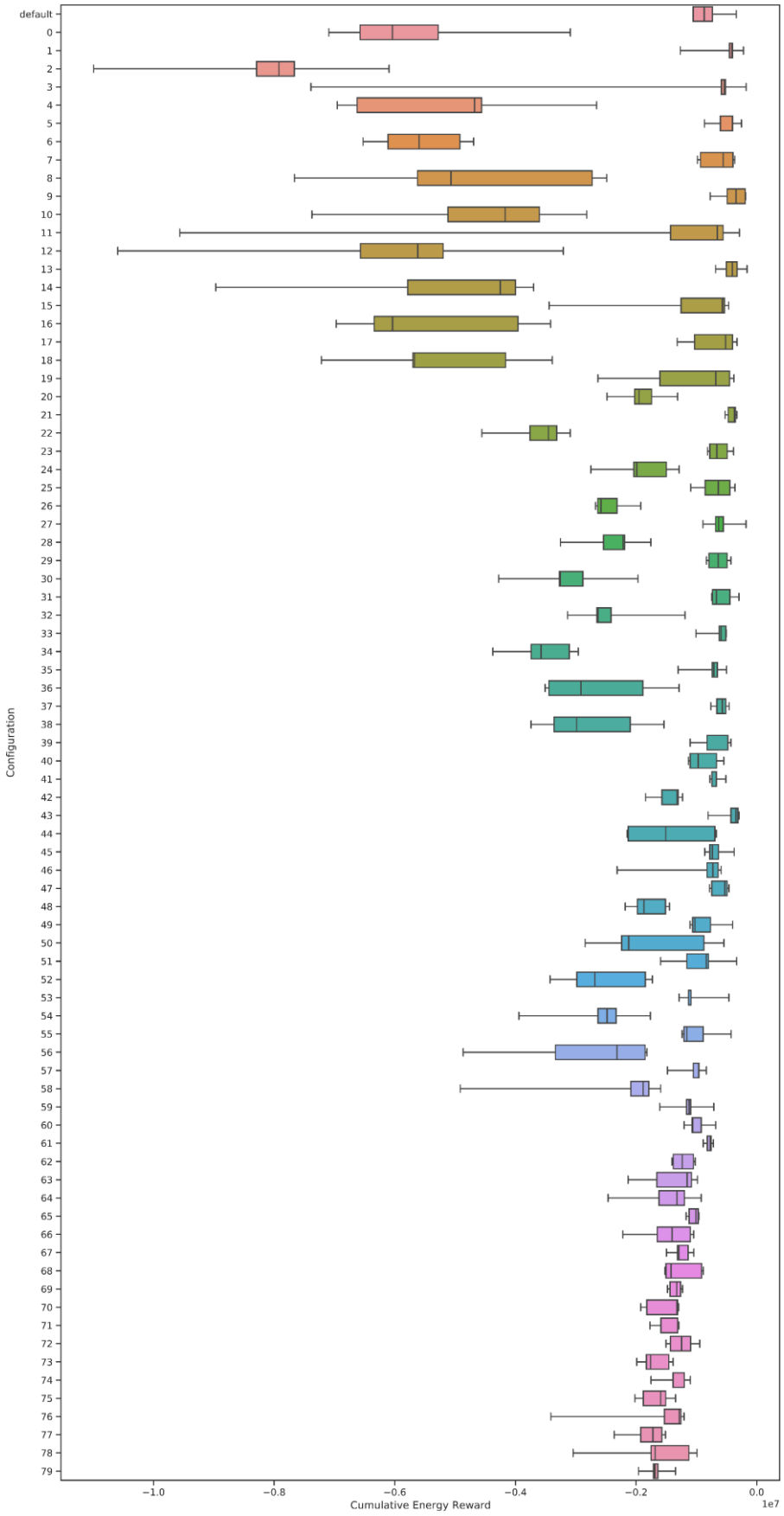


Figure 69: Cumulative Energy Reward of different architecture configurations

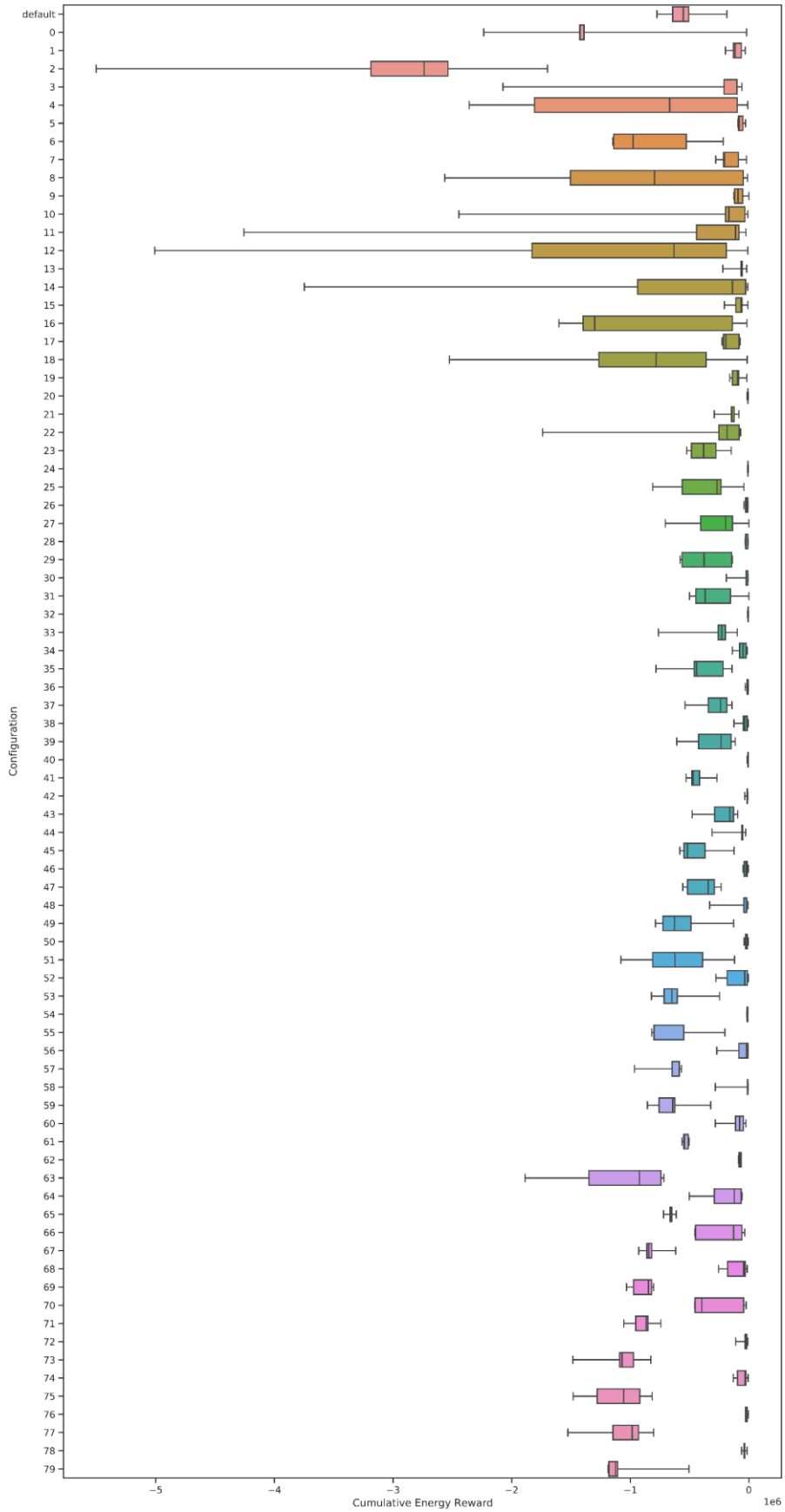


Figure 70: Cumulative Energy Reward of different architecture configurations (after 250.000 timesteps)

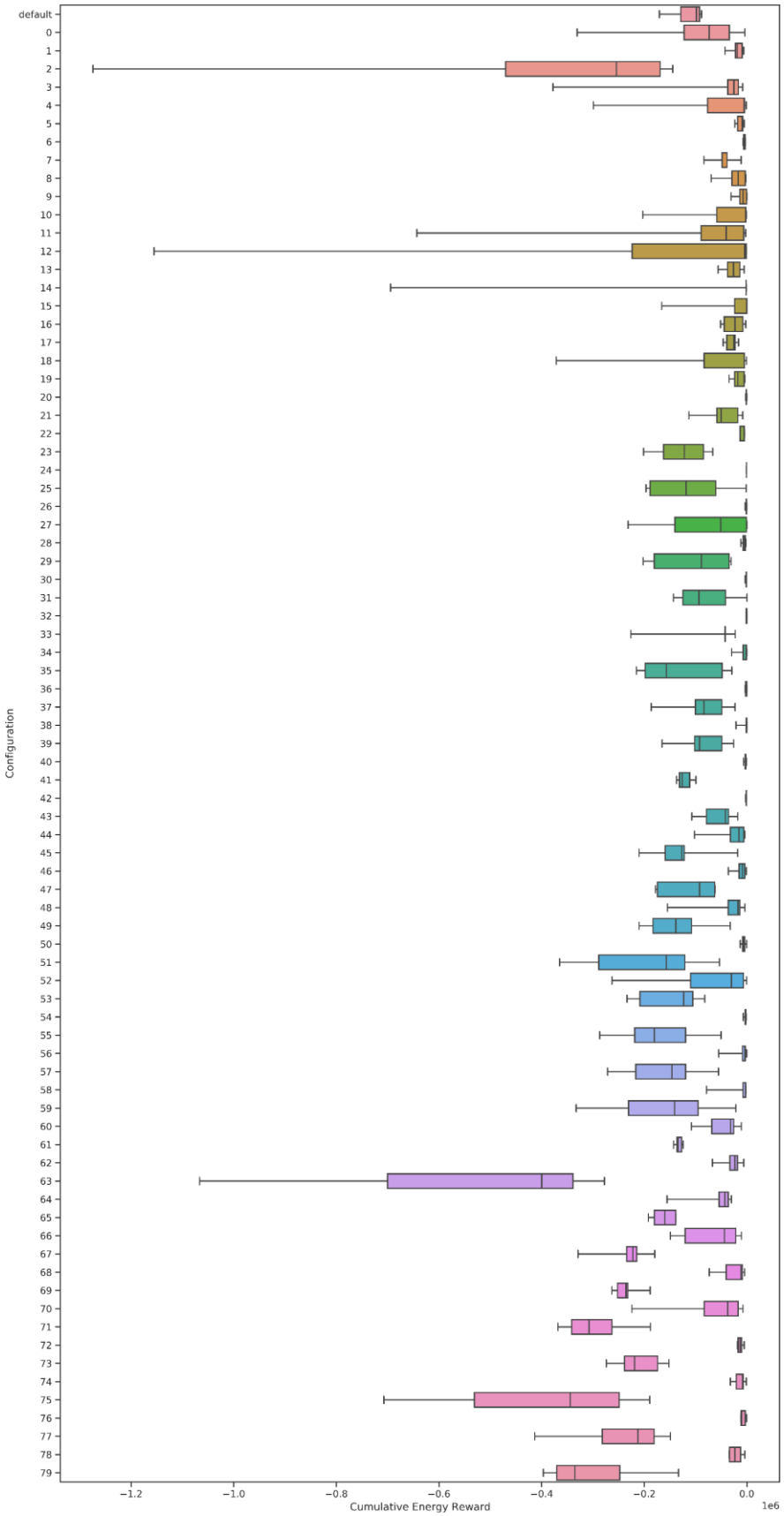


Figure 71: Cumulative Energy Reward of different architecture configurations (after 450.000 timesteps)

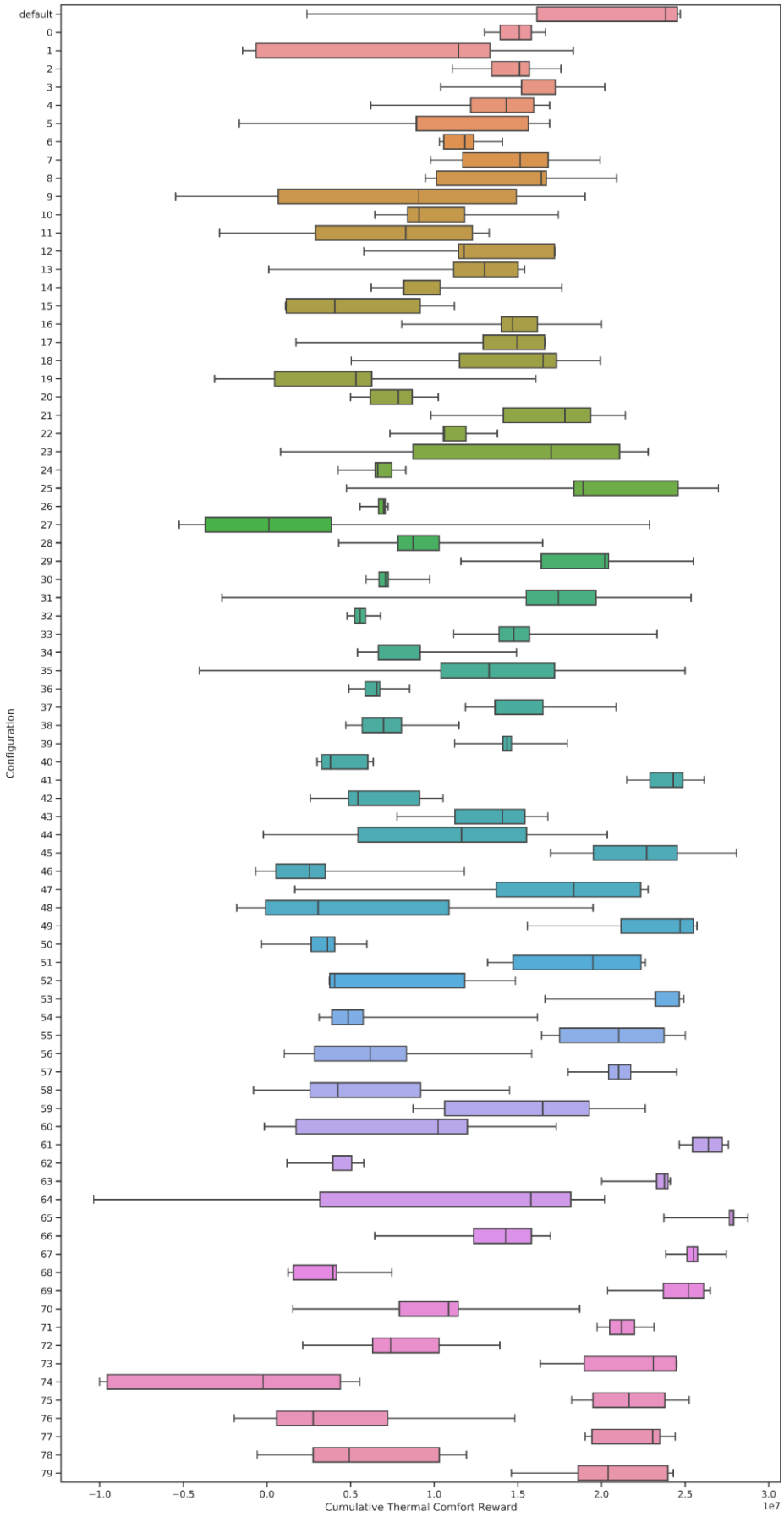


Figure 72: Cumulative Thermal Comfort Reward of different architecture configurations

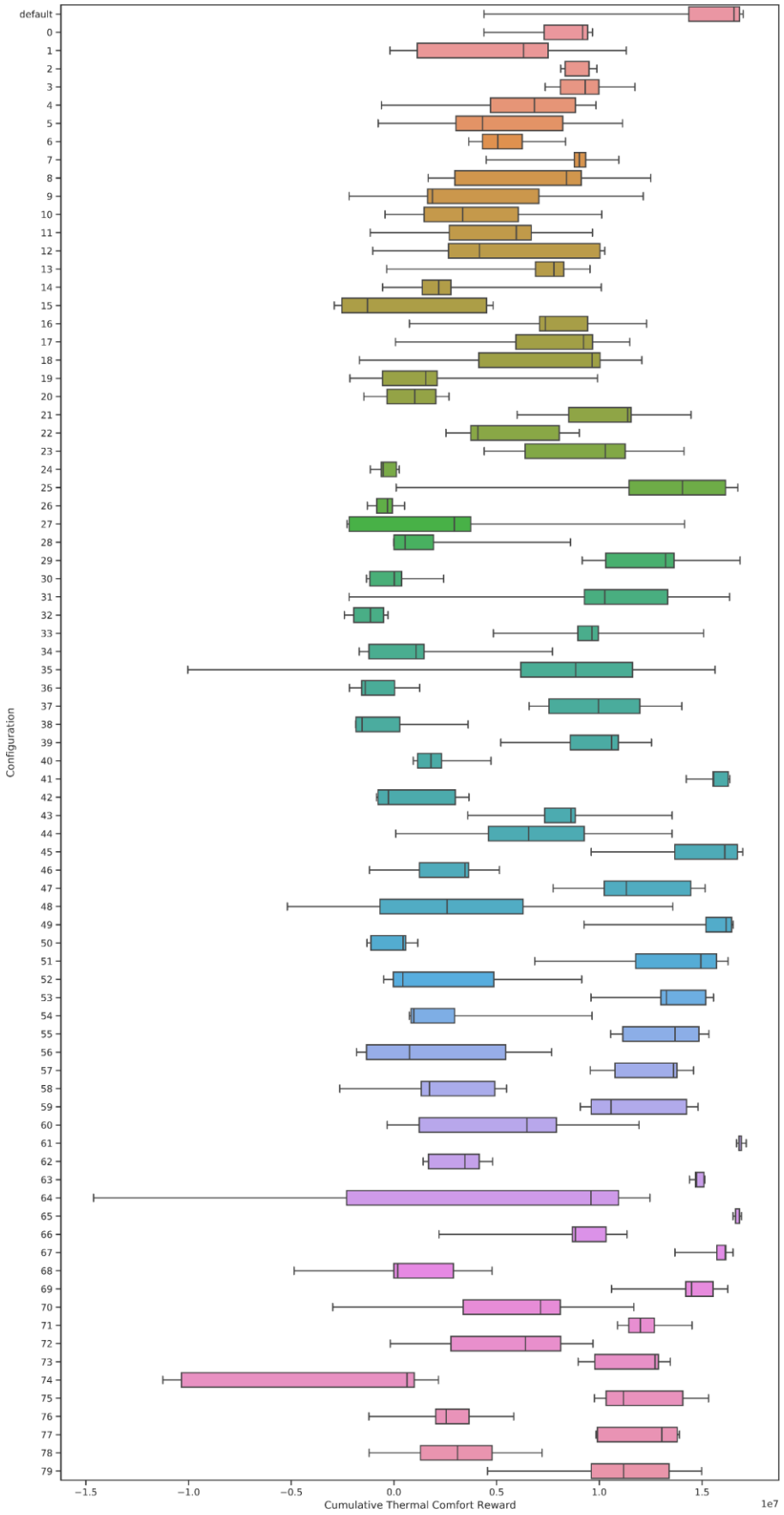


Figure 73: Cumulative Thermal Comfort Reward of different architecture configurations (after 250.000 timesteps)

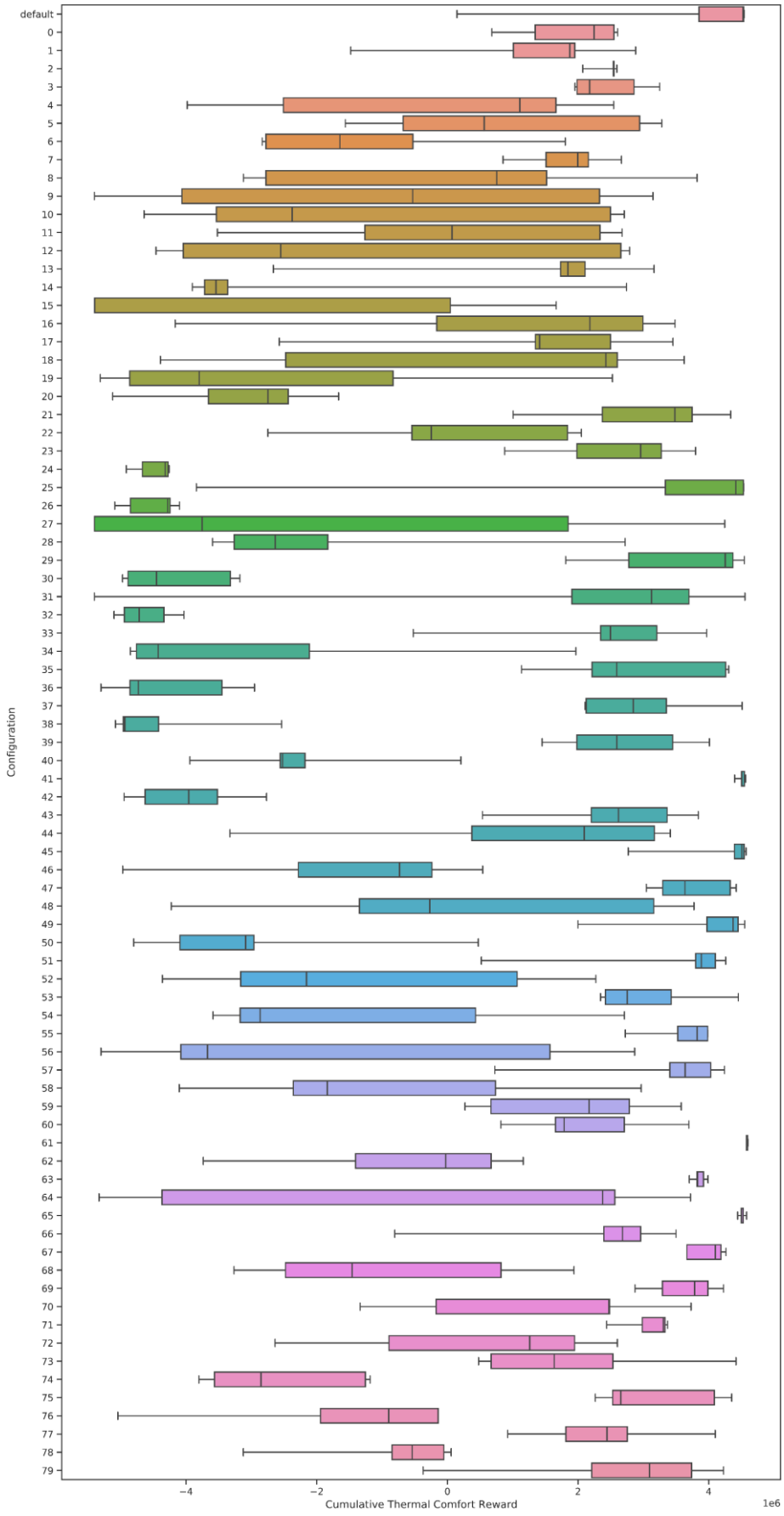


Figure 74: Cumulative Thermal Comfort Reward of different architecture configurations (after 450.000 timesteps)

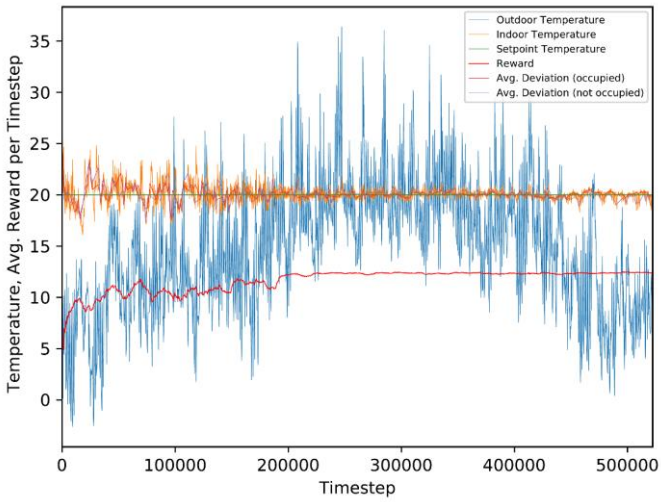


Figure 75: Evolution of temperature & reward (seed: 463276947) (from scratch)

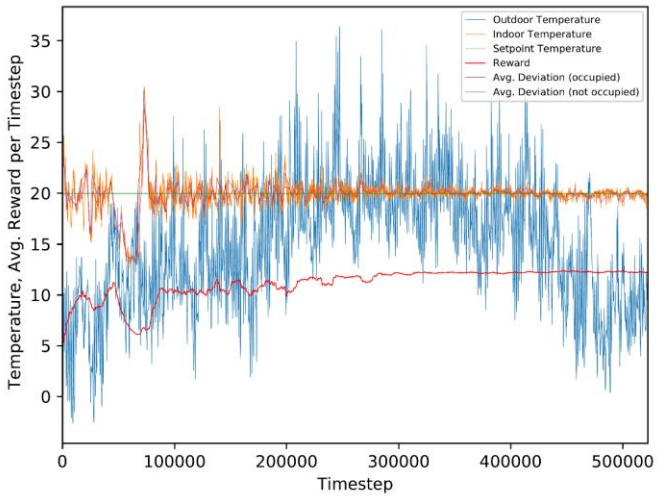


Figure 76: Evolution of temperature & reward (seed: 903450155) (from scratch)

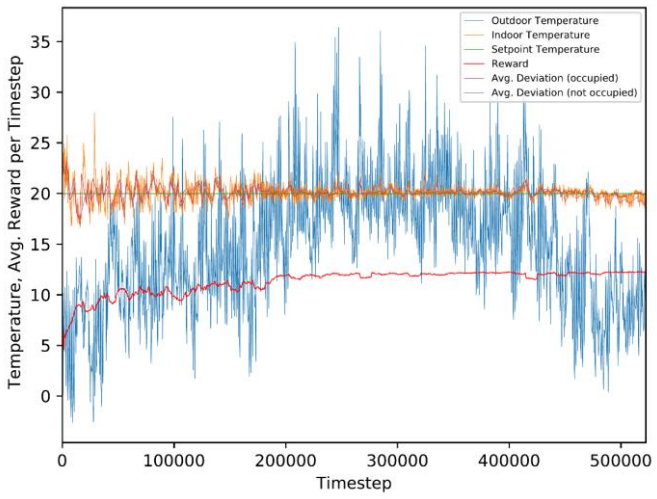


Figure 77: Evolution of temperature & reward (seed: 1377776711) (from scratch)

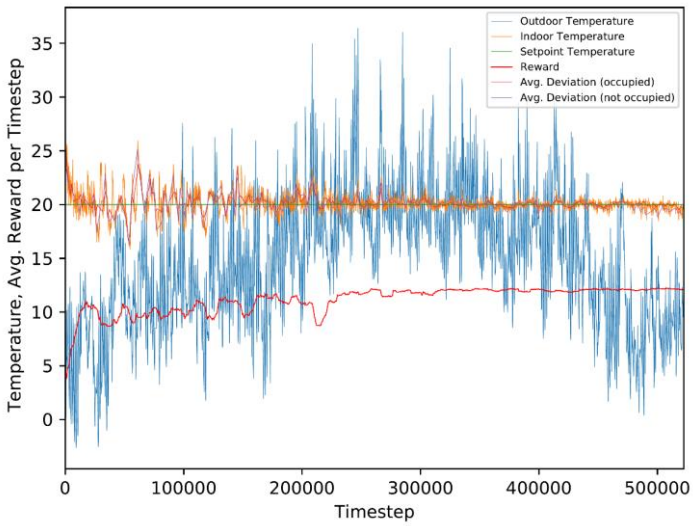


Figure 78: Evolution of temperature & reward (seed: 2522572000) (from scratch)

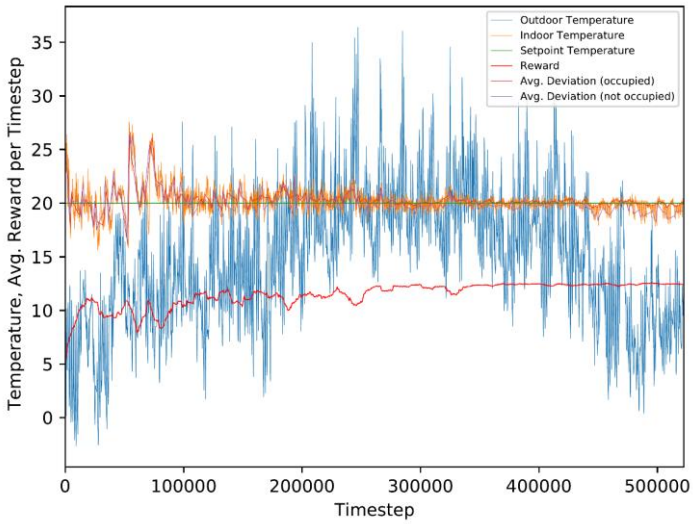
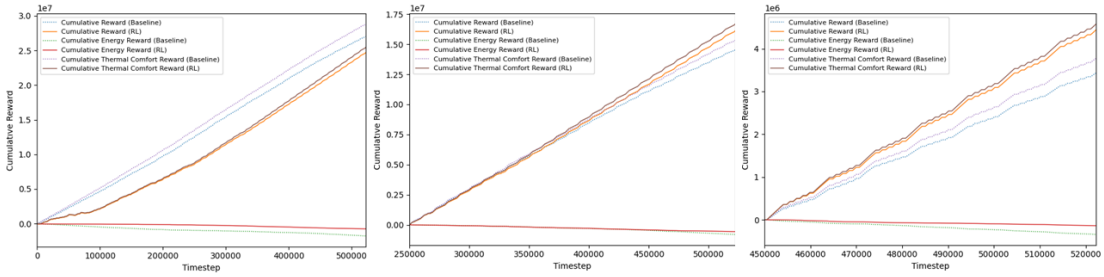
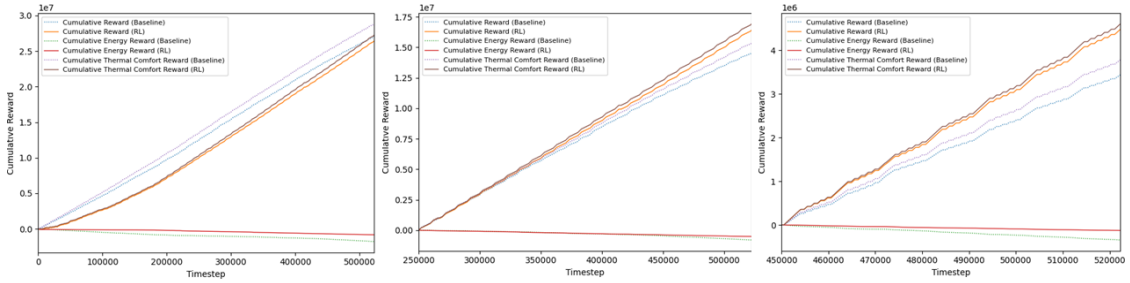


Figure 79: Evolution of temperature & reward (seed: 3952781742) (from scratch)



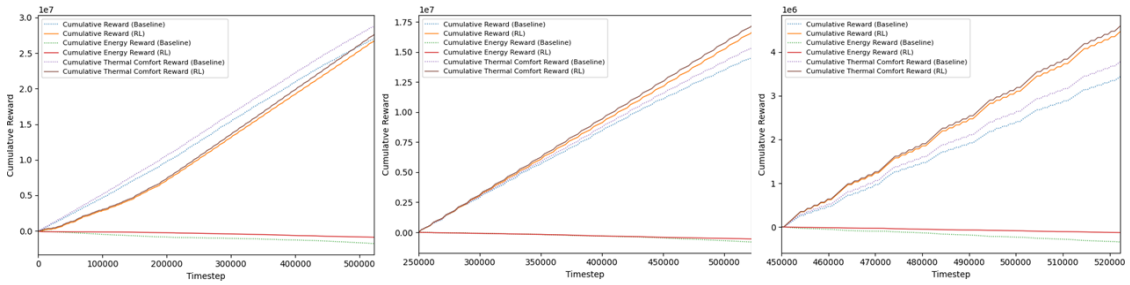
(a) (b) (c)

Figure 80: Evolution of cumulative reward (a) throughout the whole experiment (b) after 250.000 timesteps (c) after 450.000 timesteps (seed: 463276947) (from scratch)



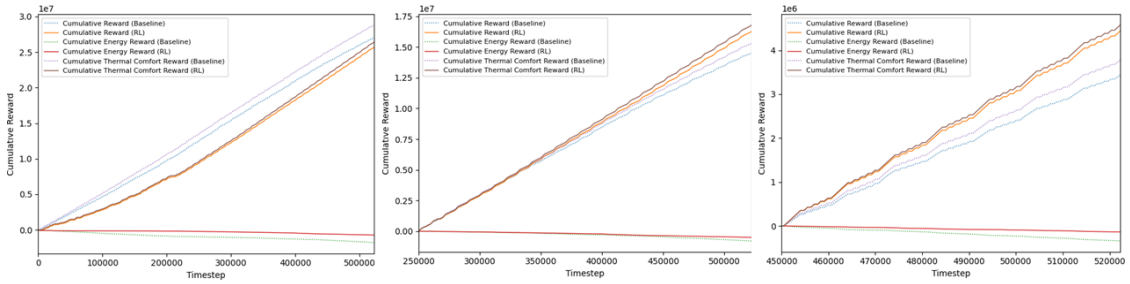
(a) (b) (c)

Figure 81: Evolution of cumulative reward (a) throughout the whole experiment (b) after 250.000 timesteps (c) after 450.000 timesteps (seed: 903450155) (from scratch)



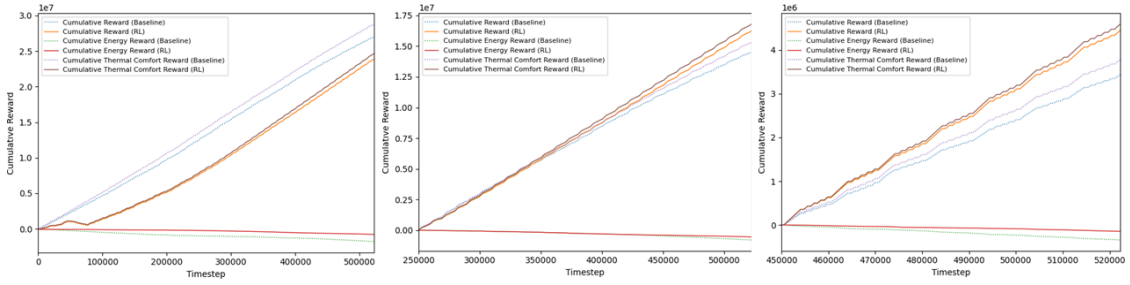
(a) (b) (c)

Figure 82: Evolution of cumulative reward (a) throughout the whole experiment (b) after 250.000 timesteps (c) after 450.000 timesteps (seed: 1377776711) (from scratch)



(a) (b) (c)

Figure 83: Evolution of cumulative reward (a) throughout the whole experiment (b) after 250.000 timesteps (c) after 450.000 timesteps (seed: 2522572000) (from scratch)



(a) (b) (c)

Figure 84: Evolution of cumulative reward (a) throughout the whole experiment (b) after 250.000 timesteps (c) after 450.000 timesteps (seed: 3952781742) (from scratch)

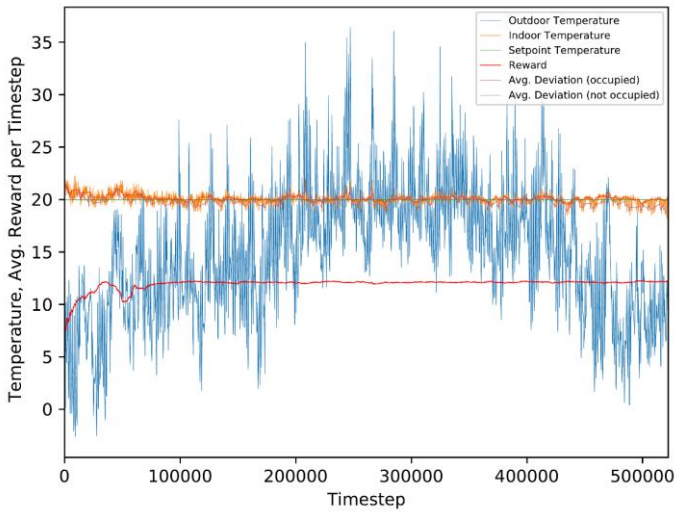


Figure 85: Evolution of temperature & reward (seed: 463276947) (pretrained)

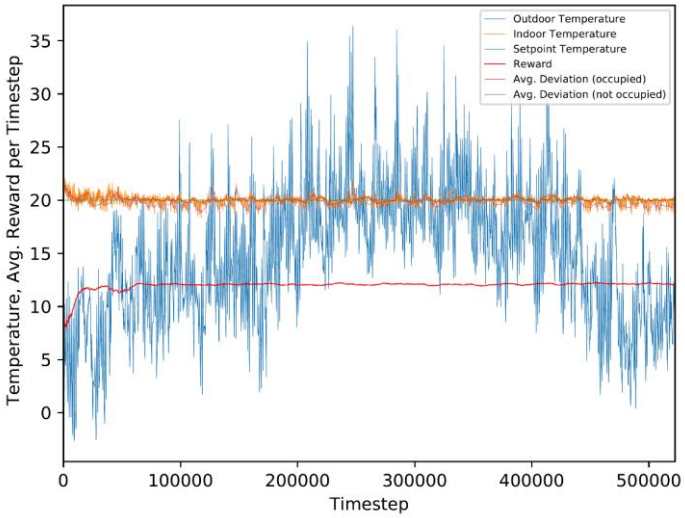


Figure 86: Evolution of temperature & reward (seed: 903450155) (pretrained)

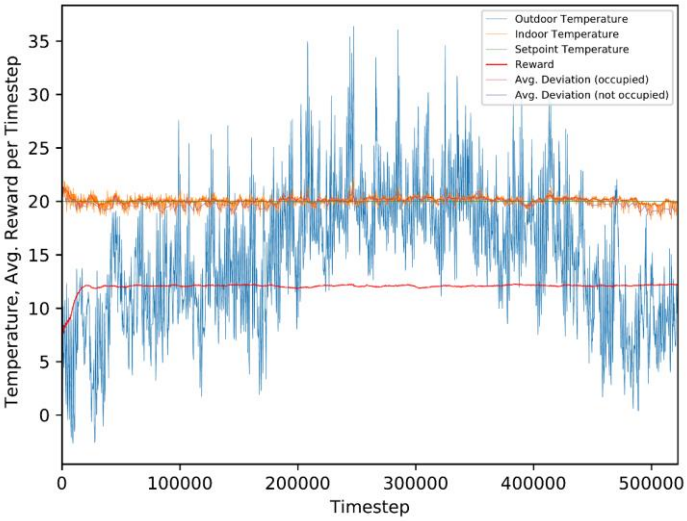


Figure 87: Evolution of temperature & reward (seed: 1377776711) (pretrained)

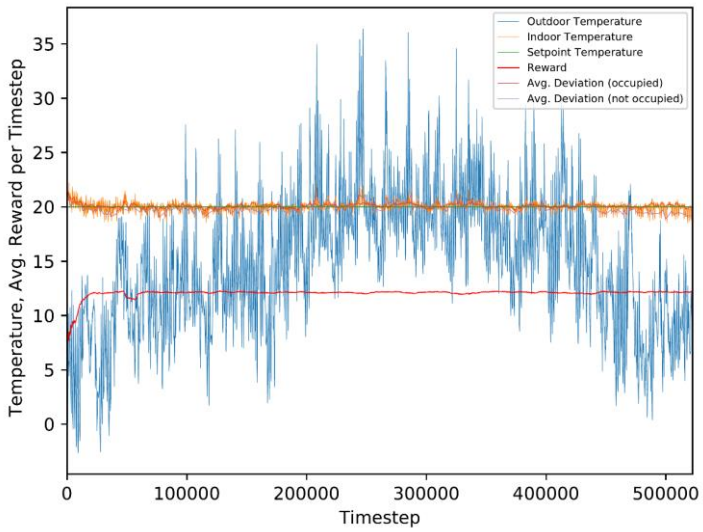


Figure 88: Evolution of temperature & reward (seed: 2522572000) (pretrained)

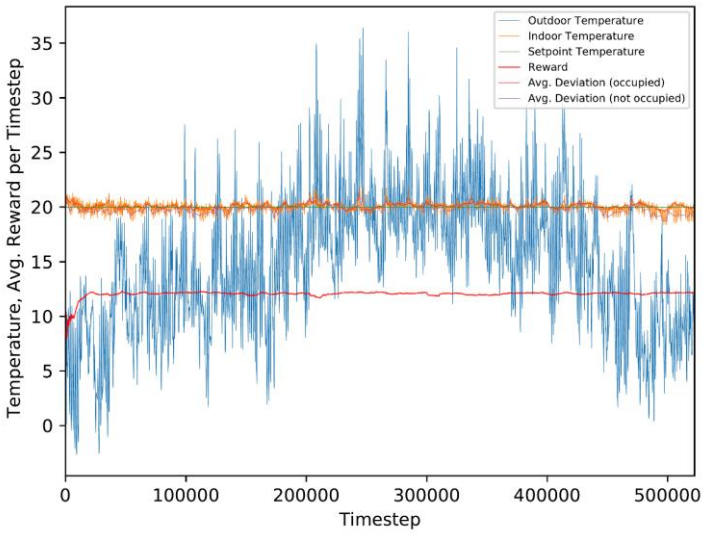


Figure 89: Evolution of temperature & reward (seed: 3952781742) (pretrained)

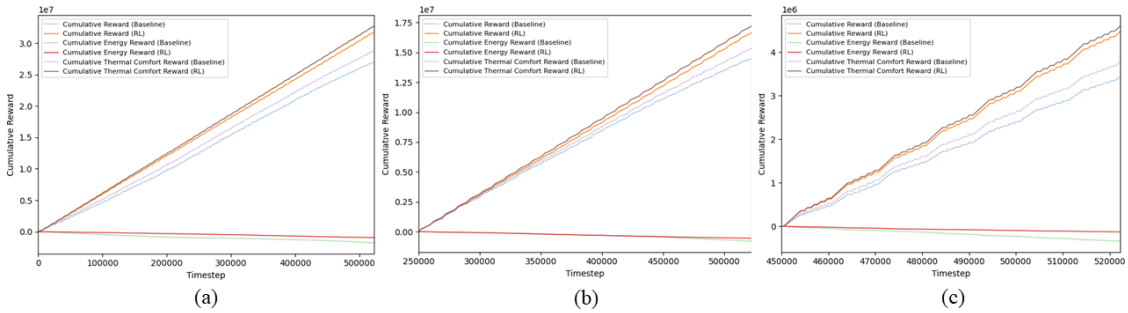
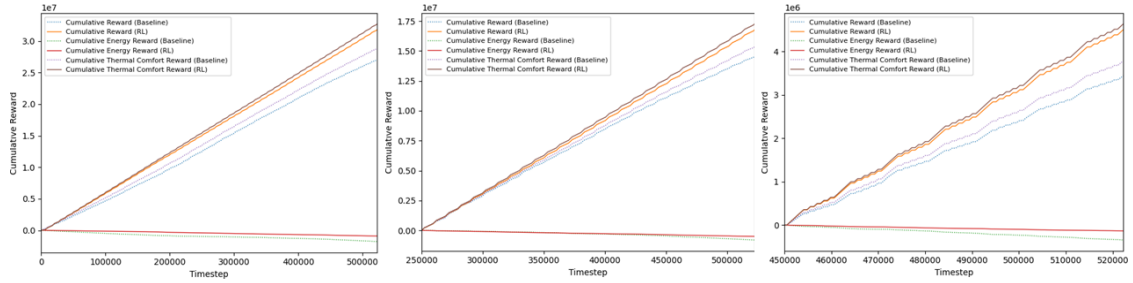
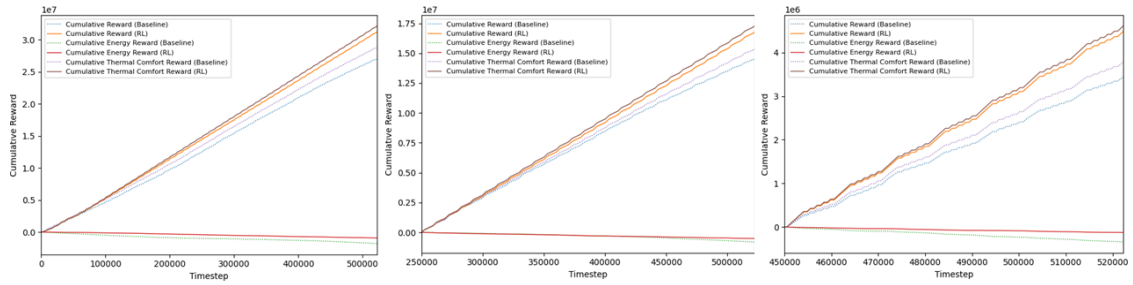


Figure 90: Evolution of cumulative reward (a) throughout the whole experiment (b) after 250.000 timesteps (c) after 450.000 timesteps (seed: 463276947) (pretrained)



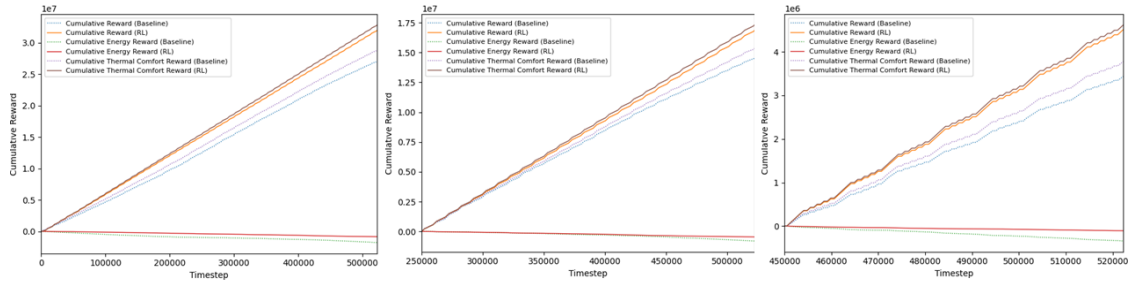
(a) (b) (c)

Figure 91: Evolution of cumulative reward (a) throughout the whole experiment (b) after 250.000 timesteps (c) after 450.000 timesteps (seed: 903450155) (pretrained)



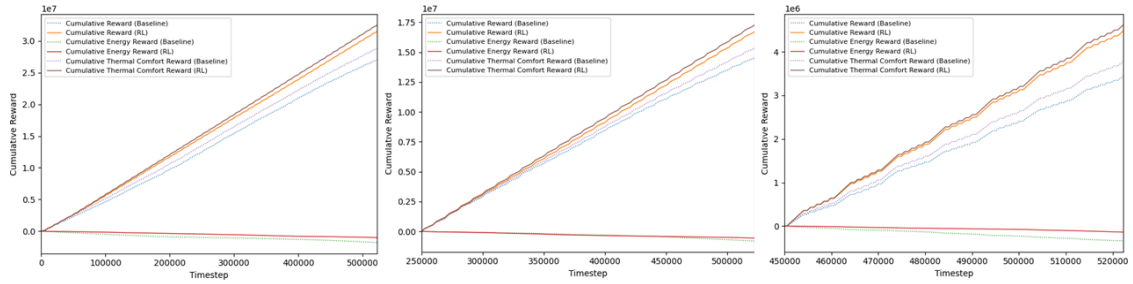
(a) (b) (c)

Figure 92: Evolution of cumulative reward (a) throughout the whole experiment (b) after 250.000 timesteps (c) after 450.000 timesteps (seed: 1377776711) (pretrained)



(a) (b) (c)

Figure 93: Evolution of cumulative reward (a) throughout the whole experiment (b) after 250.000 timesteps (c) after 450.000 timesteps (seed: 2522572000) (pretrained)



(a) (b) (c)

Figure 94: Evolution of cumulative reward (a) throughout the whole experiment (b) after 250.000 timesteps (c) after 450.000 timesteps (seed: 3952781742) (pretrained)