Chapter 8

Validation, Verification and Root-Cause Analysis

By Luong Nguyen, Vinh Hoa La, Wissam Mallouli and Edgardo Montes de Oca

Published in *DevOps for Trustworthy Smart IoT Systems* by Nicolas Ferry, Hui Song, Andreas Metzger and Erkuden Rios (eds.). 2021. ISBN 978-1-68083-824-4. E-ISBN 978-1-68083-825-1.

Suggested citation: Luong Nguyen, Vinh Hoa La, Wissam Mallouli and Edgardo Montes de Oca. 2021. "Validation, Verification and Root-Cause Analysis" in *DevOps for Trustworthy Smart IoT Systems*. Edited by Nicolas Ferry, Hui Song, Andreas Metzger and Erkuden Rios. pp. 173–213. Now Publishers. DOI: 10.1561/9781680838251.ch8.

8.1 Motivation

Verification and validation are two significant software development processes for checking that software meets its requirements and specifications and fulfills its intended purpose. In these processes, various test cases (e.g., unit tests, integration tests, regression tests, system tests) need to be designed and executed in a production-like environment that reproduces the same conditions where the software under test would run. However, having access to such an environment is usually tricky or close to being an impossible task. It is even particularly challenging in the IoT arena. The access to IoT devices might be nontrivial or limited due to many factors. Networks of physically deployed devices are typically devoted to production software. Testing applications on top of those networks might involve additional testing software, which might affect overall performance and the revenue generated by the devices (e.g., applications need to be stopped to load their new versions).

Software simulators proved to be valuable in easing the verification of the software requirements. They provide software developers a testing environment to at least manage the execution of test cases. IoT Testbeds play a similar role in testing IoT applications. They offer a deployed network of IoT devices where developers can upload their applications and test their software in a physical environment. IoT-Lab [1] and SmartSantander [9] are good examples of IoT testbeds. Testbeds often have a predefined fixed-configuration and architecture. They are also usually shared with other users, which can be a problem for measuring application quality. Hence, this problem might make simulators more attractive since they provide a more customized and controlled environment. Furthermore, simulators avoid the need for a more expensive physical network of devices.

In recent years, both academia and the commercial market have proposed solutions for the IoT simulation field. These solutions are often entirely different, although their objectives are similar. The academic solutions implement cutting-edge technology as proofs-of-concept, and they are usually not ready for production systems. By contrast, the commercial solutions are designed to be stable and flawless, even though the technology behind them might not be state-of-the-art.

The ENACT project has brought an opportunity to create the Test and Simulation (TaS) tool. Collaborating with universities and research institutions such as SINTEF and CNRS, we provide a state-of-the-art test and simulation tool with cutting-edge technology behind it. We have evaluated our solution with several industrial use cases, such as eHealth (Tellu), Smart Building (Tecnalia), and Intelligent Train System (Indra). The case studies have shown that it is stable and ready for production systems.

The TaS provides the possibility to test the IoT system based on test scenarios using pre-prepared datasets. The datasets can be the recorded data from a real system or the data generated using some data mutation operators. The TaS also allows stressing the boundaries of the scenarios to detect potential problems.

We focus on the network of sensors and the applications on top of them. Therefore, we do not consider the physical behavior of the sensors. We take it for granted that the sensors are reliable and correctly react to the physical changes (e.g., if the physical temperature rises 2 degrees, the sensor will immediately send a message with a 2 degree higher reading).

On the other hand, it is also important to note that failures usually propagate in complex systems through causal chains and produce evolving fingerprints of noisy symptoms. One of the first tasks to accomplish for an automated tool helping humans troubleshoot a system is to group events that are causally connected (and keep unrelated events separated). Achieving this is often not straightforward since components of a system can exhibit similar symptoms of two unrelated failures. We need a higher level of granularity in the monitoring indicators and a deeper analysis to distinguish two unrelated failures. Moreover, it is frequent that failures are recurrent. The system administrators, who have some experience dealing with failures, can react more quickly and efficiently against their recurrence. They can take the impact estimation and the mitigation action (e.g., reset a particular server every night) promptly.

Indeed, all aforementioned points lead to the need for a Root-Cause Analysis (RCA) tool which enables systematizing the experience in dealing with faults and problems to identify the root cause of a newly detected issue. Thanks to RCA results, remediation actions and reactions could be timely and wisely taken to prevent or mitigate the damage of the recurrence of problems.

The IoT world has promised to connect everything and create systems with an enormous number of devices. The need for RCA to implement and operate IoT systems is evident; IoT represents a generic framework that an RCA solution can target. However, several characteristics of these types of systems need to be considered: First, IoT networks are often very dynamic environments, with devices frequently joining and leaving a system (e.g., mobile devices connecting to a particular antenna). Nevertheless, most of the communications are likely to be wireless. This can introduce a higher degree of unreliability. The failure, however, can present symptoms very similar to a normal activity. For example, when we no longer receive sensed data from a sensor, it is difficult to determine whether the sensor is no longer in range or the communication has failed. Second, in many cases, the number of components/ indicators to be taken into the analysis could be enormous. This can lead to a big volume of data processed. Reducing the data dimension by avoiding less relevant attributes (i.e., noises) is a natural need. Finally, battery-powered devices may have a low-activity mode to extend their operation autonomy. In this mode, inputs may not be synchronized and have the same frequency as other information RCA uses for the diagnosis. Therefore, RCA must be able to deal with out-of-order data. In the context of ENACT, our RCA enabler would try to address all the challenges we mentioned above.

In summary, this chapter focuses on TaS and RCA, two primary parts empowering the validation and verification in an IoT DevOps cycle, which have been developed and evaluated in the context of the ENACT project. To the best of our knowledge, no similar tool had ever been created for IoT. On the one hand, the TaS tool enables the simulation and testing of an IoT system. It collects the events of a running IoT system without impacting its normal behavior. The recorded events can be used to simulate the system, inject different kinds of "problems", and collect all relevant data for detecting errors, failures, and unwanted symptoms. On the other hand, the RCA tool monitors the real system and performs the diagnostic analysis when some errors or failures occur. The enabler allows determining unknown incidents' symptoms and evaluating how much the unknown incident is similar to a known/learned one. We discuss more technical details regarding TaS and RCA in the following sections.

8.2 Test and Simulation (TaS)

In this section, we first present an overview of the TaS enabler. We then give the details of the enabler.

8.2.1 Overview and Approach

The TaS enabler is a test and simulation solution well adapted to IoT environments. It allows simulating different IoT topologies and performing various tests to detect potential errors and security failures. We first present the main features and components of the enabler that simulate an IoT system. Then, we give a detailed description of the TaS enabler architecture.

8.2.1.1 Smart IoT system components

Figure 8.1 shows some main components in a Smart IoT System:

- **The Sensor Node:** captures, pre-process, and sends the sensor data to the gateway that can be a Raspberry PI, an Arduino, etc. It implements some basic modules:
  - The Sensor module captures the environment information.
  - The Onboard Processing module reads the sensor data and pre-processes it (e.g., performs calculations and formalizes and validates data). It can be an IoT application, a Node-RED flow, etc.
  - The Communication module component communicates with the gateway to send or broadcast the processed data.
- **The Gateway device:** receives data from the sensor nodes and processes or just forwards it to the other components/services such as cloud-based application and control center.
- **The Actuator node:** reacts and controls the actuator based on the reactions of the IoT system. It contains some basic modules:
  - The Actuator module triggers a change on the IoT device, such as opening a door and activating an alarm system, etc.
  - The Onboard processing module reads the actuator data signal and converts it into an action.
  - The Communication module communicates with the gateway to receive the actuation data signal.
- **Other components:** Other components are higher-level components that can provide a service or an application that receives and processes the data and performs actions depending on the business logic.

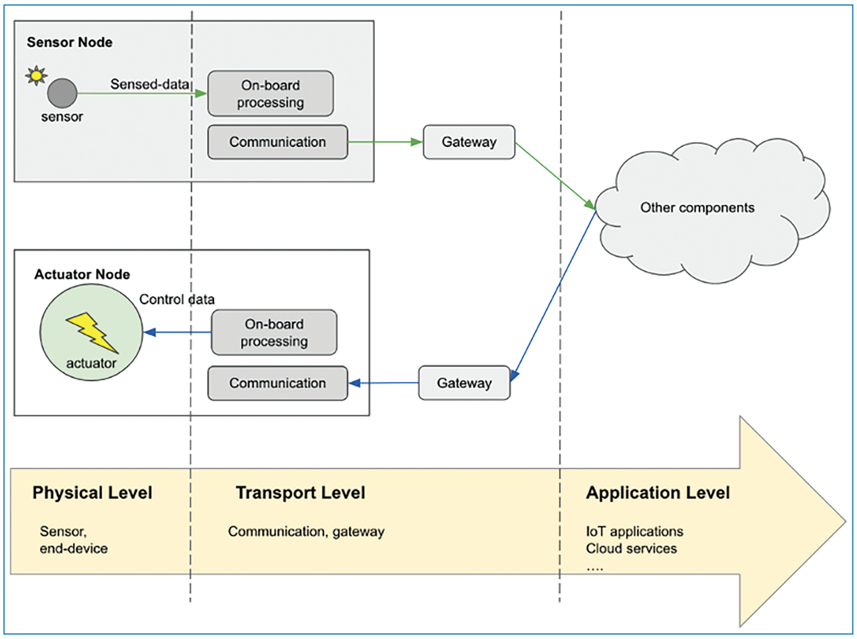


Figure 8.1. IoT system components.

The SIS components can be organized in a multi-layer architecture which we present in the next section.

8.2.1.2 Simulating a smart information system (SIS)

An SIS can be represented in three levels depicted in Figure 8.1. The physical level contains all the physical components, such as sensors and actuators, produced by a manufacturer and cannot be changed by developers. The transport level is responsible for transmitting the data within the SIS network. Developers can configure the transport level to use a specific port number or protocol. Finally, the highest level is the application level, which contains the application written by developers. The application receives the data from sensors, processes it, and produces an action to be performed by the actuators (e.g., turn the light on or off). Software engineers usually work on the application level.

When it comes to developing a software application, a software application needs to be tested every time there is a change in its source code or in the infrastructure it uses. The planned tests aim to cover many scopes involving different testing scenarios. While coping with multiple test scenarios, the testing environment needs to be flexible for manipulating input and measuring output. It is not easy to have such a testing environment for IoT applications since sensors at the physical level depend on the physical environment. Therefore, only the scenarios matching the current condition of the environment can take place.

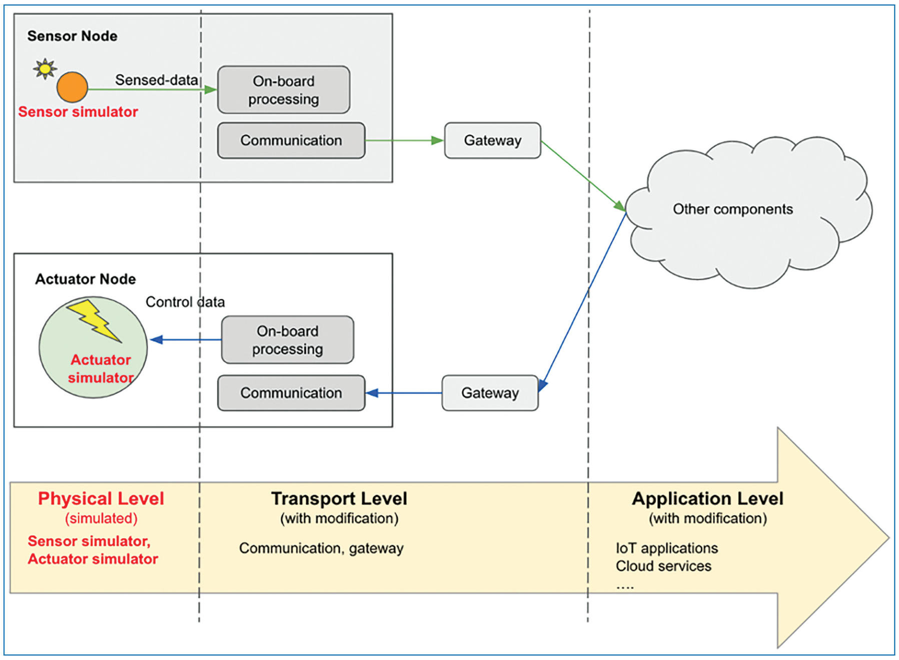


Figure 8.2. An IoT network architecture with simulated components.

Within an IoT network, a sensor captures the information of its surrounding environment at a specific time. This information is transformed into a digital format. Since it is not reasonable to wait for the change in the environment to test IoT applications, simulating various sensor measurements is very beneficial for testing them. It allows the developer to control sensor values and, thus, to simulate and test the IoT application in all scenarios without waiting for environment changes.

An actuator presents the SIS reactions in a specific situation, for example, switching on a light bulb. In such cases of testing system reactions, it is sufficient to measure the actuated data sent by the SIS to actuators. In the TaS enabler, the actuator is simulated by simply creating a hub to receive the actuated data instead of using a real actuator. Note that the impact of an actuator on a sensor is not yet considered.

With the TaS enabler, we only need to simulate the components at the physical level. The other (software) components can be cloned from the system under test and configured to work in a classical test and simulation environment, avoiding the need for communicating with a production environment, as we can see in Figure 8.2.

8.2.1.3 The TaS enabler's global approach and architecture

In this subsection, we present the architecture of the TaS enabler, which is based on the concept of Digital Twins [3]. Figure 8.3 illustrates the TaS enabler architecture.

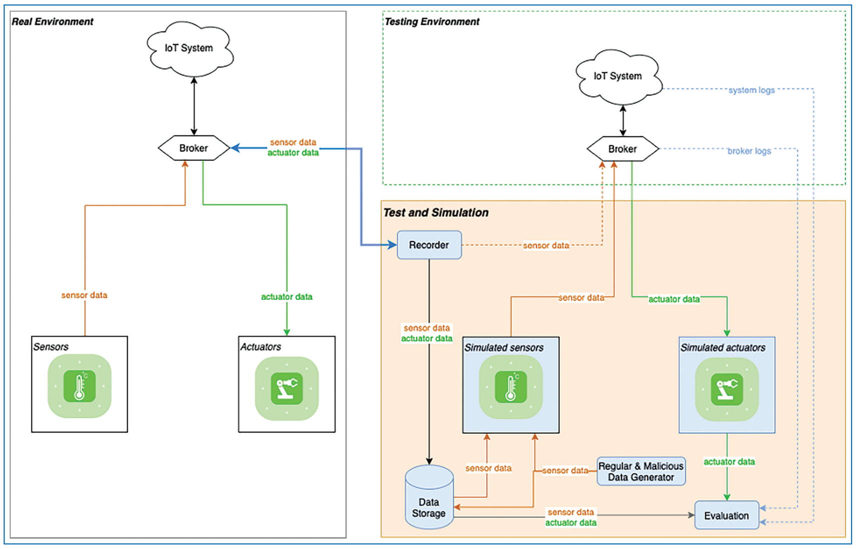


Figure 8.3. Test and Simulation (TaS) Enabler approach and architecture.

On the left-hand side, we have the system in a real (production) environment. The communication between the sensors, actuators with the IoT component is typically done via a broker. The sensors capture and send the surrounding information (e.g., temperature) to the IoT system. Based on input data, the IoT system reacts differently and sends actuation data to change the actuator settings (e.g., "change the heating level").

On the right-hand side of the figure, we have the SIS in a test environment and the TaS enabler. The system under test is the SIS that needs to be tested. The TaS enabler simulates sensors and actuators. The topology on the left side is very similar to the topology on the right side. The only difference is the simulated sensors and actuators. The simulated actuators collect the actuation data sent from the IoT system. The simulated sensors play the same role as the physical sensors providing the data signal to the IoT components. However, they are much more valuable than a physical sensor in terms of testing in the following ways:

- Firstly, by using the dataset recorded from the physical environment, the simulated sensors can repeatedly simulate the surrounding environment at a specific time. In reality, an event may happen only once, but the simulated sensor can generate the same event as many times as needed for testing purposes.
- Secondly, the physical sensors passively capture the state of the surrounding environment. It can be challenging to obtain different data from the physical sensors. In contrast, the simulated sensors use the dataset in the Data Storage as a data source. Therefore, we can generate various testing scenarios by modifying the event in the Data Storage.

- Moreover, the TaS enabler also provides a module to manipulate the data from the sensors. The Regular and Malicious Data Generator can generate regular data to test the functionalities, operations, performance, and scalability. It can also generate malicious data to test the resiliency of the system to attacks.

Besides the simulated sensors and actuators, the TaS enabler also provides some modules which support the testing process 8.3. The *Data Recorder* module records all the messages going through the broker in the physical environment. Each message can be considered as an event happening in the physical environment. Then, the recorded messages are forwarded to the broker in the testing environment. In this way, we have a “twin version” of the physical environment. What has happened in the physical environment is reproduced in the testing environment. Besides, the recorded messages are stored in a *Data Storage* as a dataset for later testing. The recorded dataset can be modified (muted) to create a new dataset, e.g., “change the event order”, “delete an event”, “add a new event”. All the testing datasets are stored in the Data Storage. The *Regular and Malicious Data Generator* enables the simulation of different sensor behaviors, from normal behavior to abnormal behavior, such as a DOS attack (the sensor publishes massive data messages in a short time), node failure (the sensor stops sending data). With data mutation, the TaS enabler can help build datasets for testing many different cases hard to produce in real life. Finally, the *Evaluation* module analyses the simulation input and output and combines them with the logs collected from the IoT system to provide the final result of a testing process.

The next section presents more details on how the TaS enabler simulates an SIS.

8.2.2 Simulation of a Smart IoT System

Most of the testing scenarios are defined by the information about the surrounding environment captured by sensors. The following section goes into detail about the simulation of sensors.

8.2.2.1 The simulation of sensor

The sensor provides the input data of an IoT system. The simulation of a sensor corresponds to the simulation of the data stream it provides. The simulated sensor has been designed for flexibility in the following ways:

- It supports different types of data report formats:
  - PLAIN_DATA: the measurement value is published directly without any transformation, it can be a number, a string or an object, for example: 15

- JSON_OBJECT: the measurement value is transformed to be an object in JSON format, with the key is set by user, for example: "temp":15
- IPSO_FORMAT: the reported data follows the Internet Protocol for Smart Object (Object and Resource Registry). A temperature sensor can report the data in IPSO format as follows (Temperature Sensor in IPSO):

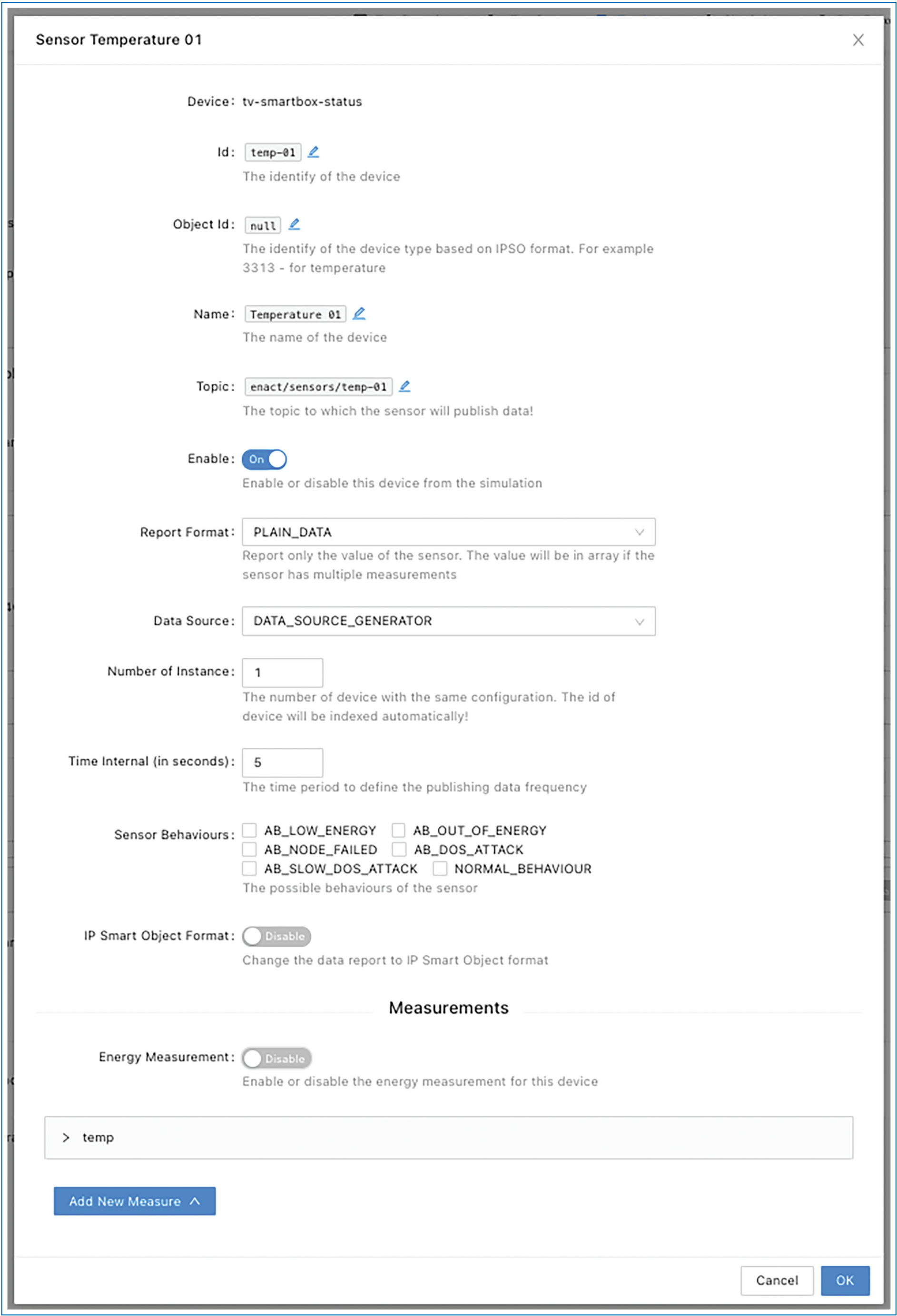
```
{
    "InstanceId": 5,
    "ObjectId": 3303,
    "TimeStamp": 1601498832,
    "TimeAccuracy": 364449977,
    "Resources": {
        "5700": 15,
        "5701": "celcius"
    }
}
```

- It supports different data sources which are used for simulation:
  - Dataset: The data source is from the data storage where the data has been recorded or created before simulating.
  - Data Generator: The data will be generated at run-time during the simulation.
  - Data Recorder: The data source is the data recorded from a real system and forwarded to the testing system.
- It supports simulating several abnormal behaviours, such as, low energy, node failure, DOS attack, and slow DOS attack.
- It supports multiple measurements with the different data types, such as Boolean, Integer, Float and Enum. For each measurement, there are several abnormal behaviours that can be selected, such as "fixed value", "value out of range", and "invalid value".

Figure 8.4 presents the definition of a temperature sensor, which generates (data source: *DATA_SOURCE_GENERATOR*) a measurement value every 5 seconds. The measurement value is published in *PLAIN_DATA* format to a MQTT/MQTTS message bus communication channel defined by the topic *enact/sensors/temp-01*. The sensor does not have any abnormal behaviour.

8.2.2.2 The simulation of actuator

An actuator can be considered as a device that receives the IoT system reaction based on the input data. We simulate the actuator as a component that will receive the reaction signal (actuation data) from the IoT system. Figure 8.5 shows the configuration of a Heater. The actuator listens for the actuation data on an MQTT/MQTTS message bus communication channel defined by the topic:



Sensor Temperature 01

Device: tv-smartbox-status

Id: temp-01
The identify of the device

Object Id: null
The identify of the device type based on IPSO format. For example 3313 - for temperature

Name: Temperature 01
The name of the device

Topic: enact/sensors/temp-01
The topic to which the sensor will publish data!

Enable: On
Enable or disable this device from the simulation

Report Format: PLAIN_DATA
Report only the value of the sensor. The value will be in array if the sensor has multiple measurements

Data Source: DATA_SOURCE_GENERATOR

Number of Instance: 1
The number of device with the same configuration. The id of device will be indexed automatically!

Time Internal (in seconds): 5
The time period to define the publishing data frequency

Sensor Behaviours: ☐ AB_LOW_ENERGY ☐ AB_OUT_OF_ENERGY ☐ AB_NODE_FAILED ☐ AB_DOS_ATTACK ☐ AB_SLOW_DOS_ATTACK ☐ NORMAL_BEHAVIOUR
The possible behaviours of the sensor

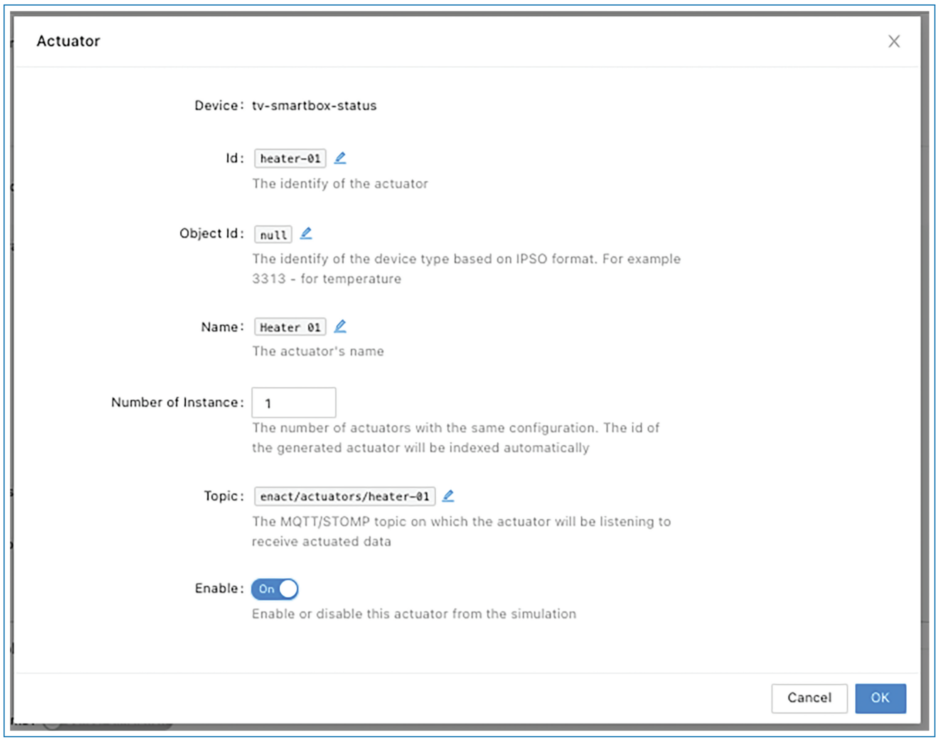
IP Smart Object Format: Disable
Change the data report to IP Smart Object format

Measurements

Energy Measurement: Disable
Enable or disable the energy measurement for this device

> temp

Figure 8.4. A temperature sensor.


Actuator

Device: tv-smartbox-status

Id: heater-01
The identify of the actuator

Object Id: null
The identify of the device type based on IPSO format. For example 3313 - for temperature

Name: Heater 01
The actuator's name

Number of Instance: 1
The number of actuators with the same configuration. The id of the generated actuator will be indexed automatically

Topic: enact/actuators/heater-01
The MQTT/STOMP topic on which the actuator will be listening to receive actuated data

Enable: On
Enable or disable this actuator from the simulation

Figure 8.5. A Heater actuator.

enact/actuators/heater-01. The topic defines the channel on which the actuator will connect to obtain the actuation data.

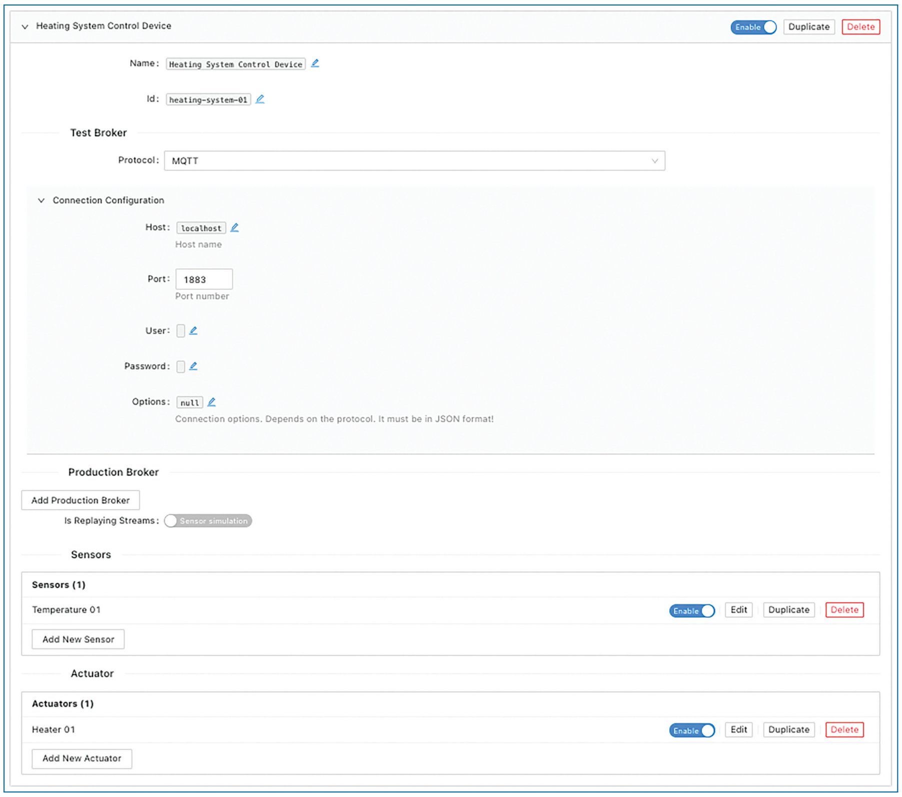
8.2.2.3 The simulation of an IoT device

In an IoT system, the sensor and actuator are usually part of the same device. An IoT device can contain one to many sensors as well as one to many actuators. Figure 8.6 illustrates the configuration of a Heating System Control device. The device has one sensor and one actuator. The data is published by the sensor and received by the actuator via the MQTT protocol.

8.2.2.4 The simulation of a network topology

Figure 8.7 presents a simple simulated network topology.

A list of simulated IoT devices forms the simulated network topology. Besides the list of devices, a network topology can also provide the identifier of the dataset (*datasetId*),which contains the data to simulate the SIS in a given time, the global replaying options, the configuration to connect with the database, and



Heating System Control Device

Name: Heating System Control Device

Id: heating-system-01

Test Broker

Protocol: MQTT

Connection Configuration

Host: localhost
Host name

Port: 1883
Port number

User:

Password:

Options: null
Connection options. Depends on the protocol. It must be in JSON format!

Production Broker

Is Replaying Streams: Sensor simulation

Sensors

Sensors (1)

Temperature 01

Actuator

Actuators (1)

Heater 01

Figure 8.6. A Heating System Control device.

the definition of the new dataset where the data generated from the simulations will be stored.

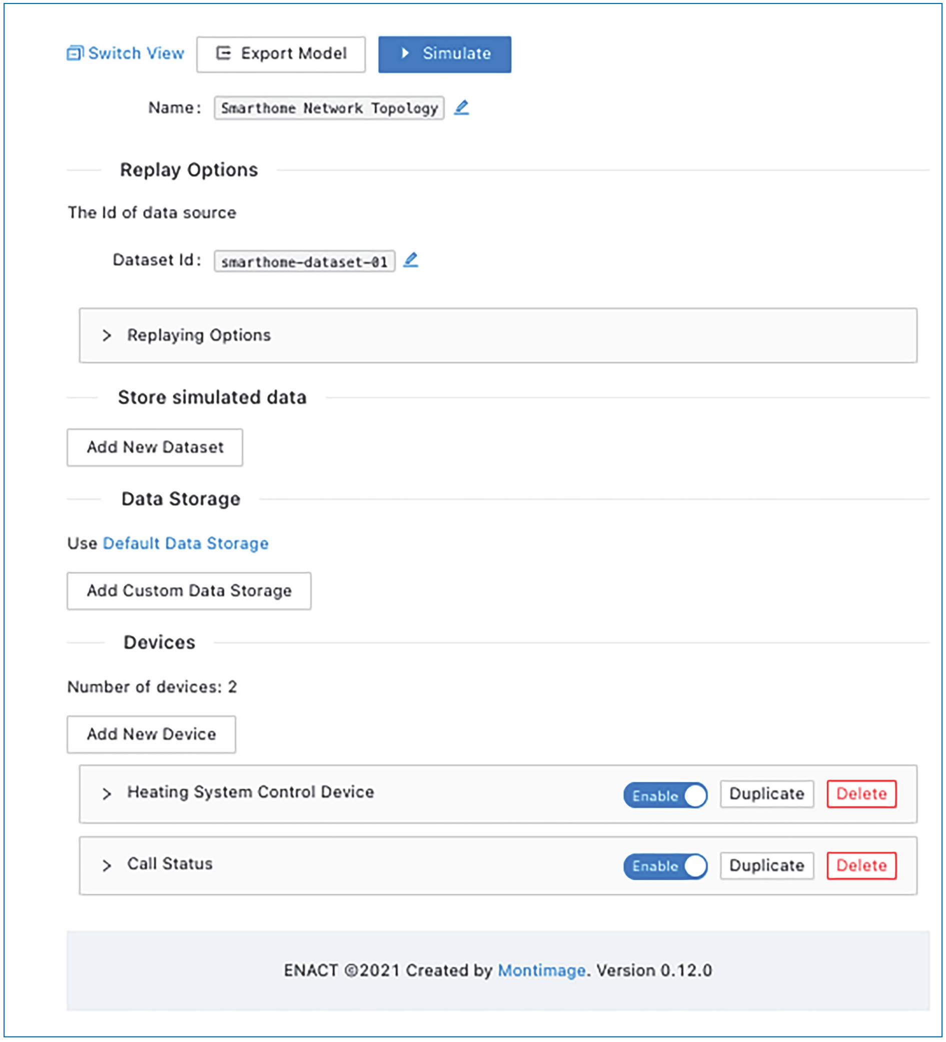
8.2.2.5 The communication between the TaS enabler and the system under test

In the ENACT project, the communication between the TaS enabler and the system under test has been implemented based on message queue protocols such as MQTT and MQTTS. Figure 8.8 presents the Message Bus class diagram, similar to the interface for all Message Queue Protocols.

Some basic message queue bus protocol methods have been implemented, such as subscribe, unsubscribe, publish, connect, and close. By design, each IoT device can have its way of communication with the SIS systems.

8.2.3 The Testing of a SIS

In this section, we present the testing methodologies and techniques we have adapted in the TaS enabler.



Name: Smarthome Network Topology

Replay Options

The Id of data source

Dataset Id: smarthone-dataset-01

Replaying Options

Store simulated data

Add New Dataset

Data Storage

Use Default Data Storage

Add Custom Data Storage

Devices

Number of devices: 2

Add New Device

Heating System Control Device Enable Duplicate Delete

Call Status Enable Duplicate Delete

ENACT ©2021 Created by Montimage. Version 0.12.0

Figure 8.7. Smarthome network topology.

8.2.3.1 The testing methodologies

This section covers the testing methodologies that the TaS enabler can support. In this first version of the enabler, we have implemented only data-driven and data-mutation testing methodologies. The other ones described below are possible future extensions.

Data Driven Testing

Figure 8.9 presents the data flow of the Data-Driven Testing method. The Data Storage contains the datasets recorded from the IoT system or entered manually. Each dataset contains sensor data (inputs for TaS) and expected actuator outputs.

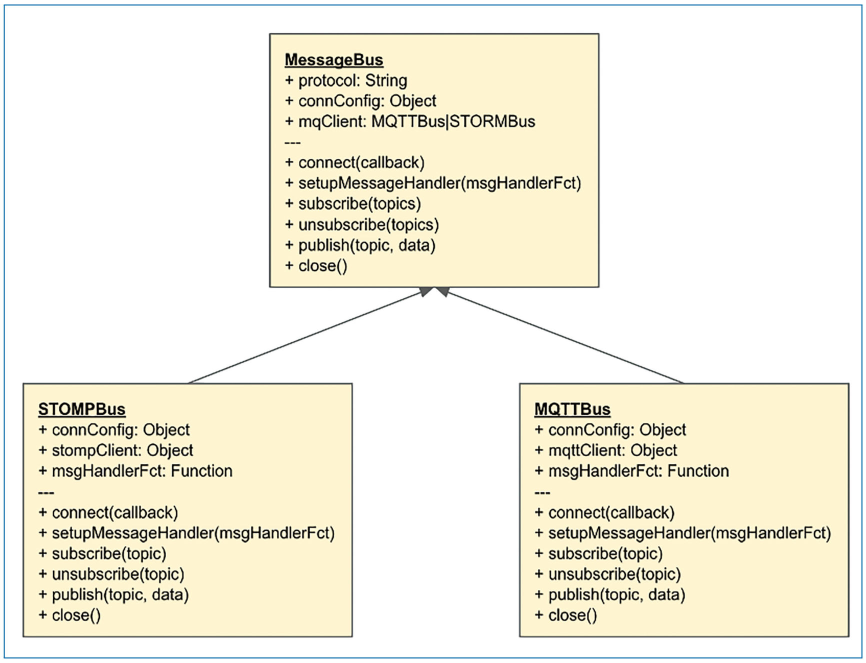


Figure 8.8. Message queue bus class diagram.

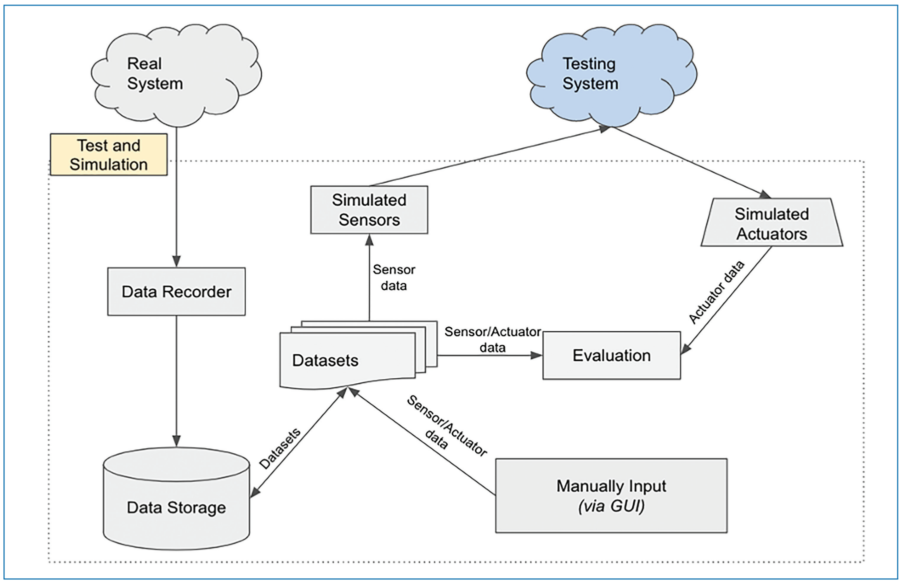


Figure 8.9. Data Driven Testing.

The expected actuator outputs can be the value recorded from the IoT system in a normal scenario. Engineers can also enter them manually via the Graphical Interface. The Evaluation module will use the expected outputs to compare them with the simulation output to determine if they match. A test case passes if the simulation output is the same as the expected output. The Data-Driven Testing method is suitable for functional and regression testing.

The Data-Driven Testing has been implemented as the main testing methodology of the TaS enabler.

Data Mutation Testing

Figure 8.10 illustrates the Data Mutation Testing architecture. The Mutant Generator generates new sensor data from existing data stored in the Data Storage by applying one or many mutated functions, such as "change the event order", "change a value", and "delete an event". The mutated data are input for the simulation. The Evaluation module generates a report about the output differences when testing the system with the mutated and the original input data. The Data Mutation Testing method is for penetration, robustness, security, and scalability testing (e.g., mutating the device identifier to obtain new devices). In the TaS enabler, we can mutate the device identity to generate many devices while testing the system scalability. There is also an interface to apply some mutation functions to a dataset manually.

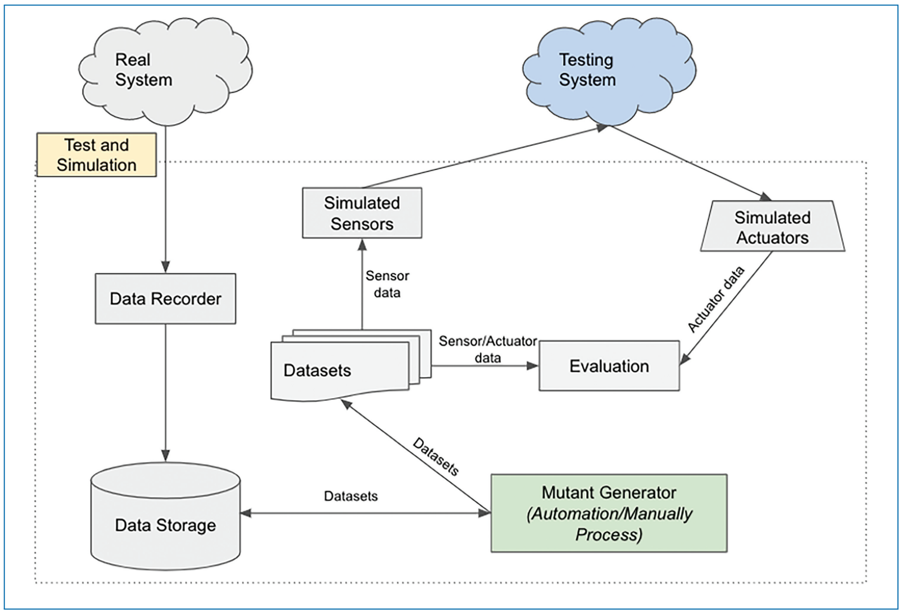


Figure 8.10. Data Mutation Testing.

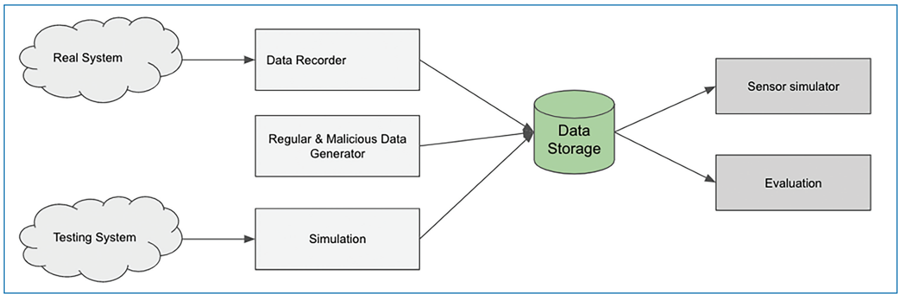


Figure 8.11. DataStorage.

Model-Based Testing [10] and Risk-Based Testing [6] are two other methodologies that we have studied but not yet implemented in the TaS enabler at the time of writing of this book chapter.

8.2.3.2 The testbeds

To simulate and test an IoT system in some specific scenarios, one of the easiest methods is to use a testbed. With testbeds, the developer can define exactly what is the input and what should be the corresponding output. This way, the tests can be done automatically and easily integrated in the DevOps Continuous Integration and Continuous Deployment processes. In TaS, testbeds are built from datasets which are recorded from a real system or generated by the TaS based on scenarios.

DataStorage

The Data Storage contains all the datasets for testing and simulating.

As depicted in Figure 8.11, the datasets are fed into the DataStorage via three sources: the data from the real system recorded by the Data Recorder, the data generated by the Regular and Malicious Data Generator, and the data generated by the simulation. The datasets in the Data Storage are used to simulate the sensors and to validate the simulation output.

The database connection of the TaS enabler is flexible. Two simulations can use different databases. If there is no configuration specified, the TaS enabler uses a default database. The database to connect to can be any database that the TaS enabler can reach.

Event

An event represents a message sent through the communication channels. It can be a data message sent by a sensor or data received by an actuator. Figure 8.12 presents the format of the event Schema.

The *timestamp* attribute indicates the time when the event has been captured. The *topic* represents the MQTT/MQTTS bus channel related to the event. It is the

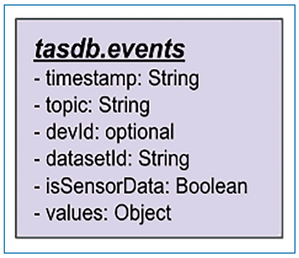


Figure 8.12. Event Schema.

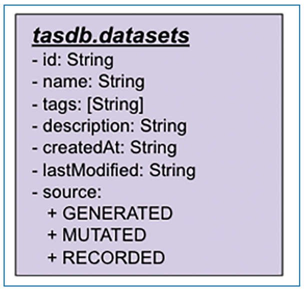


Figure 8.13. Dataset Schema.

source channel of the event where the data message corresponds to sensor data. It is the destination channel of the event where the data message is data received by an actuator. Its value is crucial for identifying the event to be replayed. The *datasetId* attribute represents the dataset to which the event belongs. The *isSensorData* is set to True if the event presents a data message sent by a sensor. It is False if the event is a data message received by an actuator. The *values* attribute contains the value of the message data. This value can be a number, a string, or an object. This design helps make the event generic and making it possible to consider any message data type.

Dataset

A dataset contains a series of events for a specific scenario. Figure 8.13 presents the schema of a dataset. Each dataset has a unique *id*, *name*, and *description* to describe the dataset objective. The *source* attribute indicates the dataset source. A dataset can be created from a recording session by the Data Recorder (source: RECORDED) or generated by the Regular and Malicious Data Generator (source: GENERATED). A dataset can also be derived by cloning and modifying data from another dataset (source: MUTATED).

By grouping the events by the dataset Id, we have all the events belonging to a dataset.

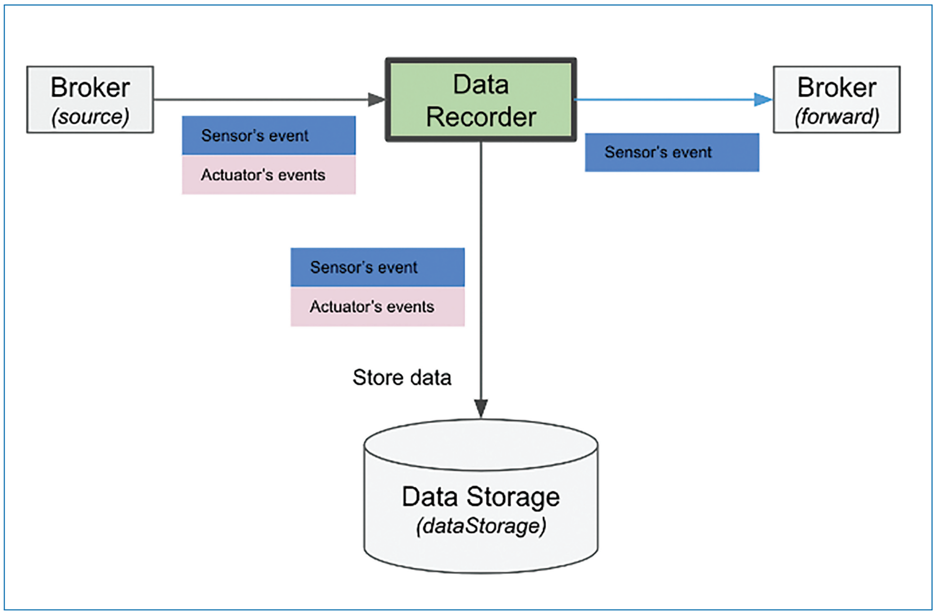


Figure 8.14. Data Recorder data flow.

8.2.3.3 The data recorder and digital twins concept

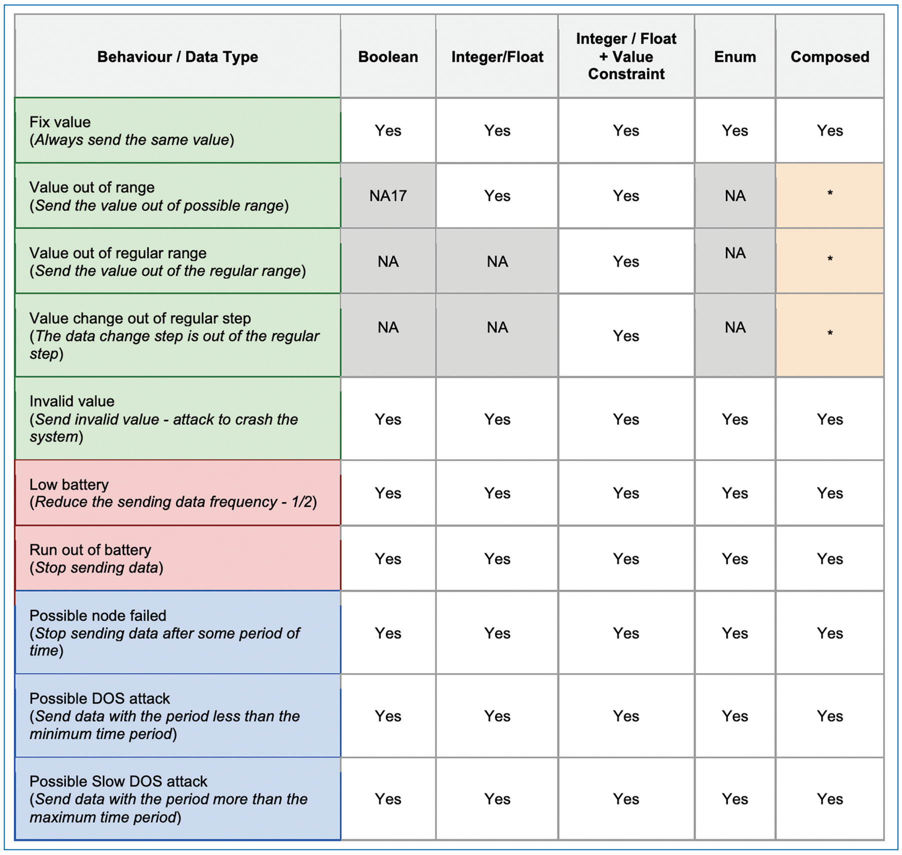
The TaS enabler provides the possibility to simulate an IoT system using historical data. To this end, a Data Recorder module is needed.

Figure 8.14 presents the data flow of the Data Recorder. All the events in the real system (coming from the broker) will be recorded. This data (including both sensor and actuator data) is stored in the Data Storage as a dataset. The sensor data can be forwarded directly to the testing system (using the forwarding broker). With the recorded data from the real system, the SIS can be tested with real input. The more data from sensors are recorded, the more test scenarios are tested. By synchronizing the Sensor simulator timestamps with the Data Recorder, it is possible to simulate a particular SIS (following the Digital Twin concept). By monitoring the SIS input and output, we can build an automatic testing process for a complex IoT system.

The recorded data can be used as a source for simulation. It can also be mutated so that it can contain different values for obtaining a modified testing scenario. In the next section, we will explain how to generate a new dataset using a given behavior profile.

8.2.3.4 The regular and malicious data generator

When testing the IoT system, there are many testing scenarios and cases that do not frequently occur in reality. With the real IoT system, it is almost impossible to collect the datasets for many testing scenarios. The TaS enabler provides a powerful



Behaviour / Data Type	Boolean	Integer/Float	Integer / Float + Value Constraint	Enum	Composed
Fix value (*Always send the same value*)	Yes	Yes	Yes	Yes	Yes
Value out of range (*Send the value out of possible range*)	NA17	Yes	Yes	NA	*
Value out of regular range (*Send the value out of the regular range*)	NA	NA	Yes	NA	*
Value change out of regular step (*The data change step is out of the regular step*)	NA	NA	Yes	NA	*
Invalid value (*Send invalid value - attack to crash the system*)	Yes	Yes	Yes	Yes	Yes
Low battery (*Reduce the sending data frequency - 1/2*)	Yes	Yes	Yes	Yes	Yes
Run out of battery (*Stop sending data*)	Yes	Yes	Yes	Yes	Yes
Possible node failed (*Stop sending data after some period of time*)	Yes	Yes	Yes	Yes	Yes
Possible DOS attack (*Send data with the period less than the minimum time period*)	Yes	Yes	Yes	Yes	Yes
Possible Slow DOS attack (*Send data with the period more than the maximum time period*)	Yes	Yes	Yes	Yes	Yes

Figure 8.15. Abnormal behaviours based on the data type and the constraints.

tool to solve this problem. The Regular and Malicious Data Generator module helps developers create a testbed. It enables generating sensor data for various scenarios, e.g., making the temperature too high or too low. By combining multiple data, one can create a testbed that includes many incidents or attack scenarios, such as DDoS and data poisoning. The Data Storage stores all the generated data for further use. Based on the data type and constraints on the time, values, or energy use, many abnormal behavior types may exist as depicted in Figure 8.15.

The abnormal sensor behaviors are defined by energy, reporting time, and value constraints.

Figure 8.16 illustrates how a data value is generated based on the selected behaviours of the sensor. In the beginning, the energy constraint is checked. There are two behaviors related to energy. If the sensor is in the low-battery mode, we can reduce the reporting frequency. Notice that the user initially sets the frequency. If the sensor is out of battery, it stops sending data. In the next step, we consider the time constraint. We can select among three behaviors. The possible DOS attack

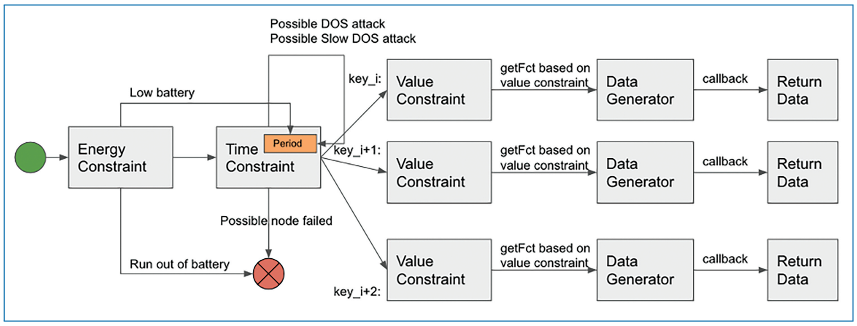


Figure 8.16. Data generating flow.

increases the reporting frequency to simulate a DOS attack, and the sensor sends a lot more data than what is considered normal. If there is a constraint on the maximum delay time for reporting data, we can simulate a behavior like a possible slow DOS attack. For example, if the system expects to receive the temperature information every 5 seconds maximum, then a slow DOS attack could change the sensor behavior that the sensor sends a message every 6 seconds. Based on the time constraint, we can simulate a sensor that stops sending data for a certain period (node failed). Finally, for each measurement provided by a sensor, the value constraint is checked. There are many behavior types based on the measured data type, such as invalid value or fixed value. Based on the selected behavior, the Data Generator function returns a specific value for the measurement.

Besides the sensor's behavior, it is also possible to change the behavior of the IoT devices. For example, the GATEWAY_DOWN behavior makes the simulated IoT device stop working after some time. When an IoT device stops working, all the sensors and actuators belonging to that device also stop working.

8.2.3.5 Automatic testing

The TaS enabler has been designed to be easily integrated into any Continuous Integration and Continuous Delivery processes. Figure 8.17 illustrates the TaS enabler concept. One of three events trigger the TaS process: code commit, new component (software module or hardware device) added, new scenario added. Several tests are executed by simulating the different testing scenarios on the system under test. The tests can cover functional, operational, security, performance, and scalability testing. If all the tests pass, we can deploy the new changes in the real environment.

Following the process we depict above, we can automatically test every change in the system and cover every test scenario.

Test case

While testing an IoT system, we may want to test different network topologies, such as adding a new device, removing a device, or just changing the way to

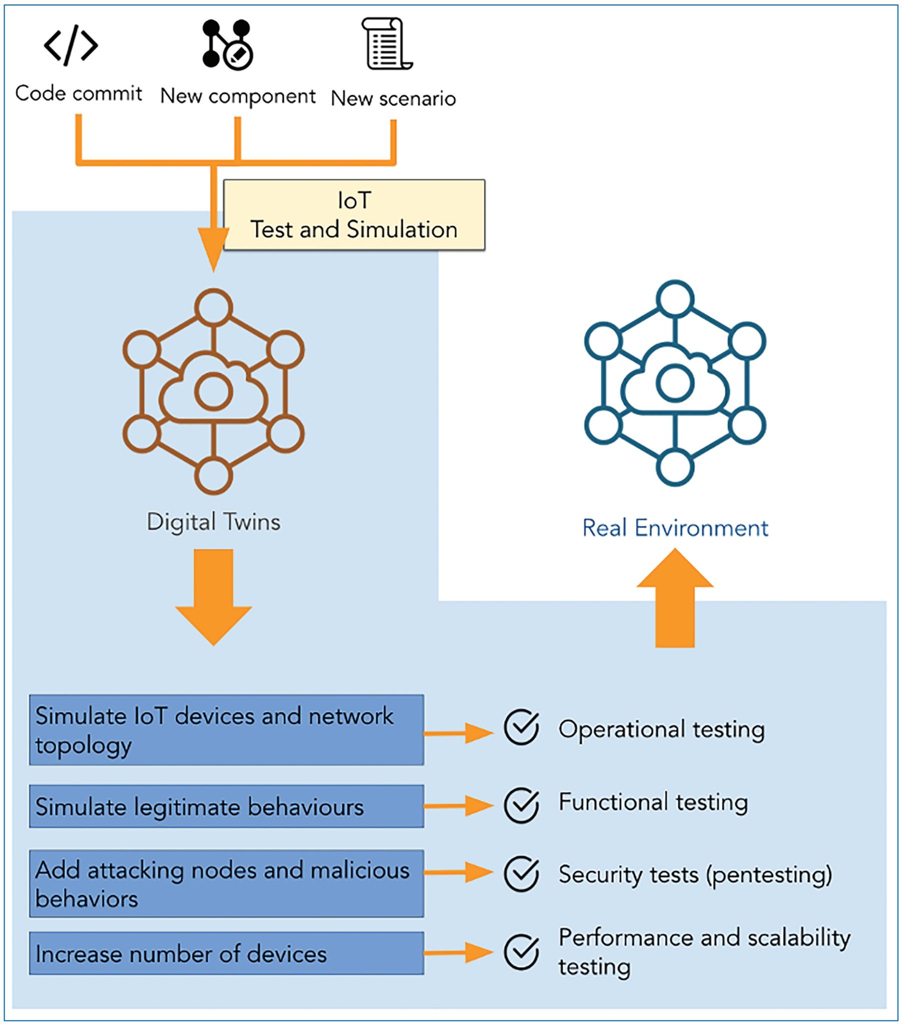

Code commit
New component
New scenario
IoT Test and Simulation
Digital Twins
Real Environment
Simulate IoT devices and network topology
Simulate legitimate behaviours
Add attacking nodes and malicious behaviors
Increase number of devices
Operational testing
Functional testing
Security tests (pentesting)
Performance and scalability testing

Figure 8.17. Test and Simulation (TaS) Workflow.

connect devices. A test case is a collection of tests executed on one network topology. There can be many different testing types (e.g., functional, security, or scalability testing). A dataset defines a test. For test execution, the TaS enabler runs the simulation and testing using each dataset by following the test order in the test list. We can change the order of the datasets via the web interface.

Test campaign and the integration into DevOps cycle

While the test case groups the test by the defined network topologies, the test campaign contains all the test cases that should be executed for each change in the IoT system. The test campaign is the global test that covers every testing scenario and testing aspect. The test campaigns are executed automatically every time there is

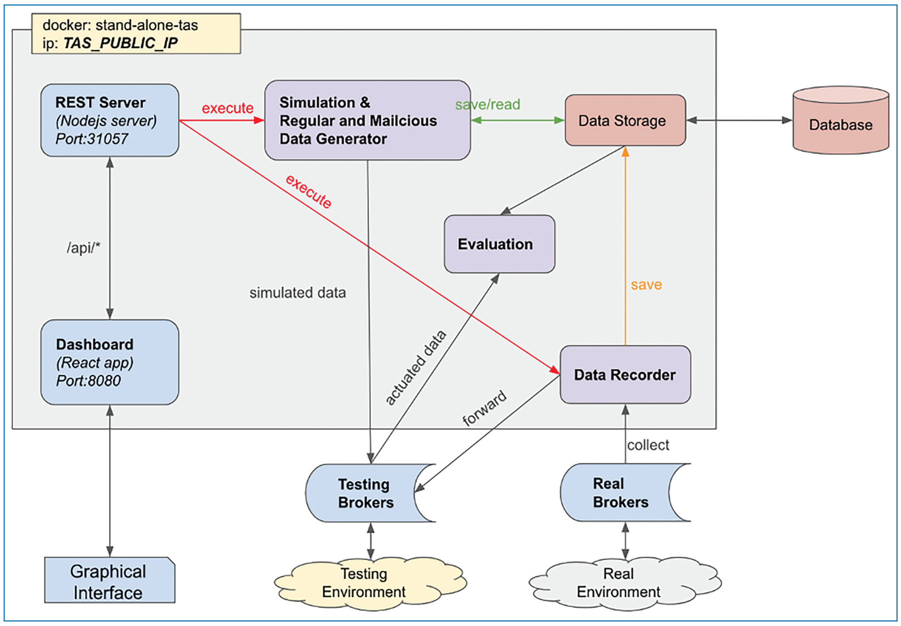


Figure 8.18. Test and Simulation (TaS) Enabler Docker image.

a change in the system. For each test campaign, the TaS enabler runs test cases according to the order in the list.

8.2.3.6 The evaluation module

The Evaluation module collects the simulated actuator data as well as the other metrics of the system. Then, it performs the evaluation based on the testing methodology (see Section 8.2.3.1 for more details).

The next section presents the implementation of the TaS enabler.

8.2.4 Implementation

8.2.4.1 The test and simulation (TaS) docker image

The TaS enabler has been designed to be portable. It can be installed as a Node.js application and packaged as a docker image. Figure 8.18 presents the communications between the modules inside a docker container and between the docker container and other modules.

The REST Server provides an API to interact with the tool. Via this API, we can execute the module Data Recorder, Simulation, and Regular and Malicious Data Generator. The Database is external to the docker container and can be connected via the Data Storage module. The dashboard is the graphical interface implemented using ReactJS [27].

Table 8.1. Basic APIs to integrate into a DevOps cycle.

Path	Method	Data	Response
/devops/	GET		Get automation testing configuration
/devops/	POST	{webhookURL, testCampaignId}	Update the automation testing configuration
/devops/start	GET		Trigger the simulation and testing process
/devops/stop	GET		Stop the simulation and testing process
/devops/status	GET		Get the status of the current execution

8.2.4.2 Basic APIs

Table 8.1 presents the list of basic APIs exposed by the tool for integration into a DevOps cycle.

8.2.5 Evaluation

The TaS enabler has been evaluated in several use cases in the ENACT project.

8.2.5.1 Itelligent train system

Figure 8.19 shows the data flow of the TaS enabler in the Intelligent Train System (ITS) use case. The Data Recorder records the WSN Coordinator data from broker-01 part of the ITS system. The recorded data is stored in the Data Storage. The Simulated Wire Sensor Network (WSN) Coordinator uses the recorded data to simulate a several WSN Coordinators for testing the scalability of the ITS system. All the simulated WSN Coordinators publish the data to broker-02 which is in the ITS system under test. By evaluating the gateway status, we can assess the scalability of the ITS system.

8.2.5.2 E-Health system

The TaS enabler is used in an e-Health use case to test if the parsing data function works correctly in the e-Health gateway. Figure 8.20 presents the data flow of the e-Health use case. The simulated sensors send some valid and invalid data messages to the internal broker, and then these messages are consumed by the CloudAgent. By mutating data messages, we can test the CloudAgent in various test scenarios, such as "invalid data format" and "invalid value".

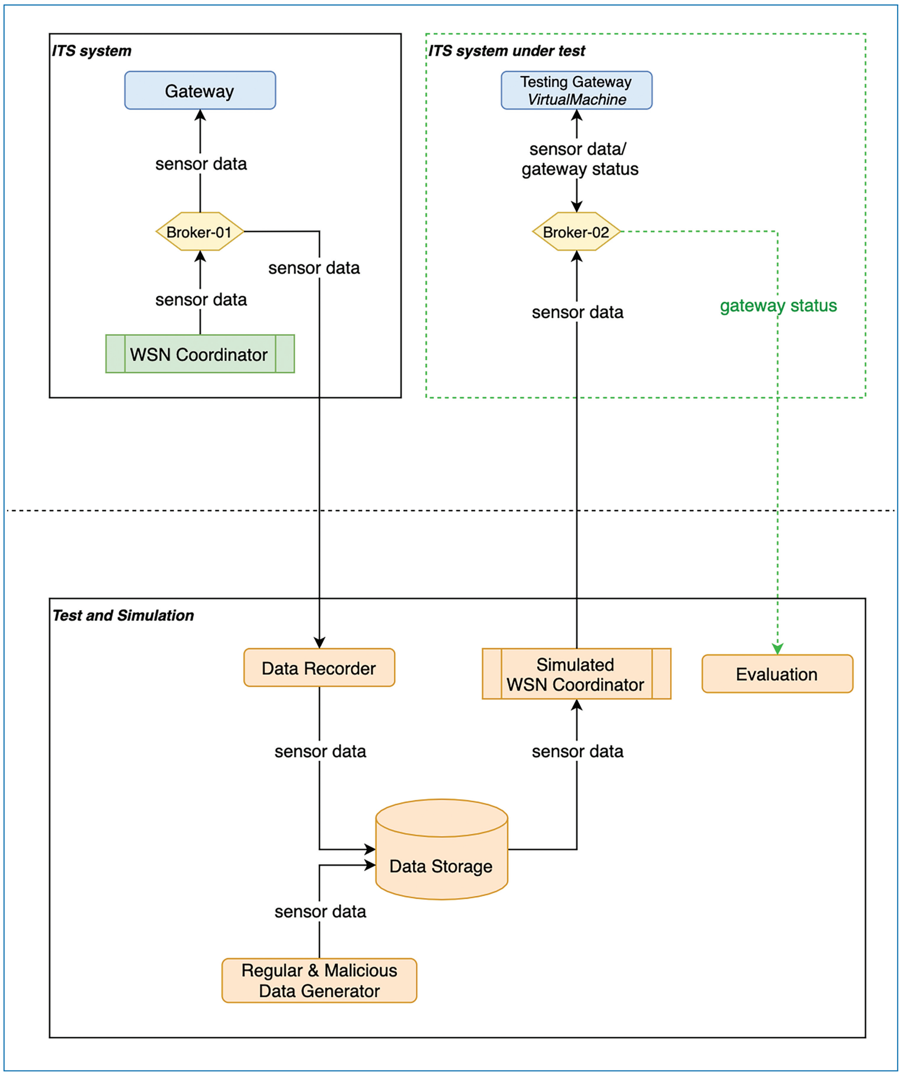


Figure 8.19. Intelligent Train System.

8.2.5.3 Smart home system

In the Smart Home use case, we used the TaS enabler as part of the DevOps cycle. Figure 8.21 shows the data flow of the TaS enabler. First, the Data Recorder records and builds the testing dataset from the real system. For each system change, such as new features, and software updates, the TaS enabler uses the recorded dataset to check the system reaction. By comparing the recorded system output with the simulated system output, we can detect miss behaviors in the updated system.

Using the TaS enabler, we can automatically test the SIS in various test scenarios. However, due to the SIS complexity, a problem may happen at any moment while

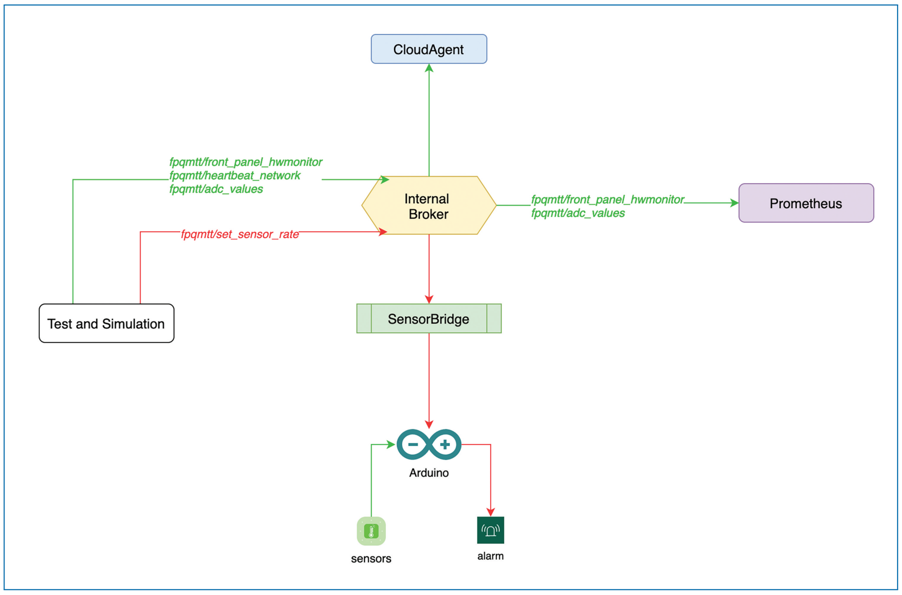


Figure 8.20. E-Health System.

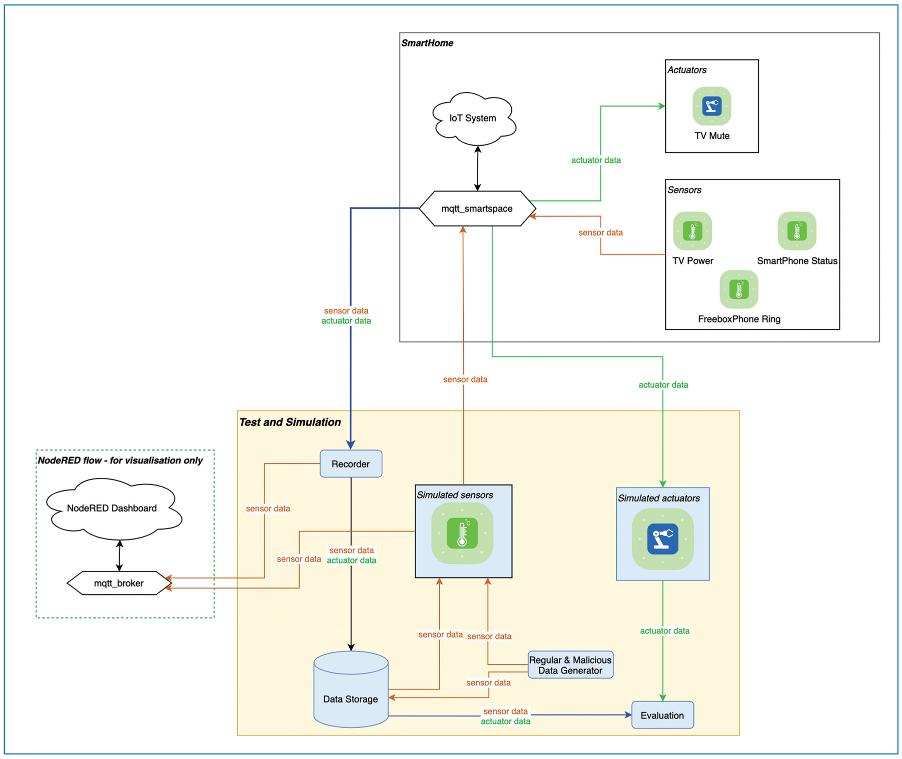


Figure 8.21. Smart Home System.

the system is running. Knowing the root cause of the problem is very important to find the solution. Therefore, we need a tool to identify the root cause of the problem while running an SIS.

8.3 Root-Cause Analysis (RCA)

Root Cause Analysis (RCA) is a systematic process for identifying "root causes" of problems or events and for responding to them. System administrators and DevOps engineers use RCA not only for detecting the problems but also for understanding their root-causes to prevent the recurrences and/or mitigate the impact. 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 similarly observed ones. Figure 8.22 presents the high-level architecture of the implemented enabler.

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 enhances its extension to new data formats. Parsing such data allows extracting various attribute values relevant to the origin of any detected incident. We automatically select the most relevant attributes by using several machine learning algorithms. These attributes increase the analysis accuracy and reduce the data dimensions as well as the computation resources needed.

The **historical data** is a set of data used for learning purposes. It consists of labeled records collected over time. These records describe the original cause of several incidents (e.g., a sensor is no longer permitted to send data to the central gateway) and the relative attribute values (e.g., downstream data bit-rate measured

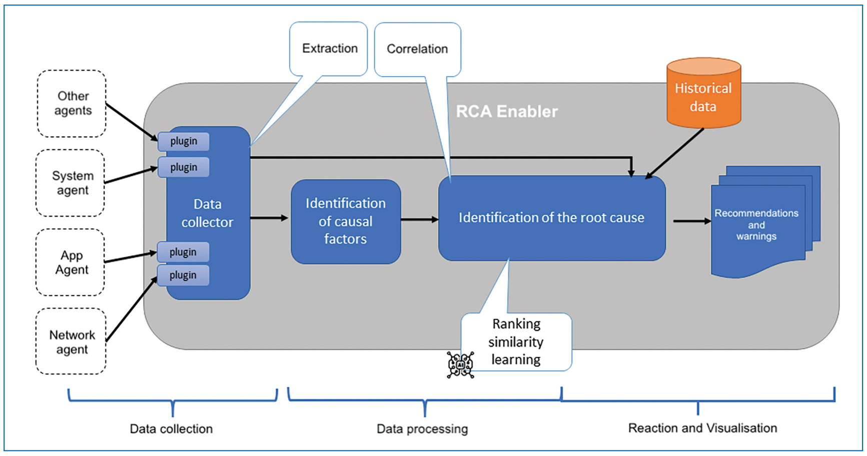


Figure 8.22. RCA Enabler high level architecture.

in the central gateway decreased). The **historical data** is constructed by two means:

- Active learning: We deal with controlled systems. Therefore, the collected data can be easily labeled by actively performing different tests injecting known failures and attacks.
- Passive learning: Once an incident is detected and alerted (e.g., by a third-party tool) without knowing its origin, thanks to the aid of the system experts, manual RCA is performed by debugging different logs and correlating various events to determine the corresponding root causes. The result of this work can be stored in the database with its relevant attribute values.

The historical data are derived from these two sources. The idea is to determine when the system reaches a known undesirable state with a known cause. It involves using the concept of Similarity Learning [8], i.e., Ranking Similarity Learning. The RCA tool calculates the similarity of the new state with the known ones. It presents the most similar states in the relative similarity order. The final goal is to recognize the incident's root origin by using historical data. In this way, the tool can recommend to the operator which countermeasures to perform based on known mitigation strategies.

The RCA Enabler works following two phases: the knowledge acquisition phase (Figure 8.23) and the monitoring phase (Figure 8.24). The former is for building a historical database of known problems and incidents. The latter consists of monitoring the system in real-time, analyzing the newly-coming incident by querying the historical data, and suggesting possible root causes. It is worth noting that passive learning in the knowledge acquisition phase can be continuously run during the monitoring phase. We describe the details of each module in the following subsections.

8.3.1 Data Collection

Analyzing an SIS requires different statistics and data, i.e., the logs, metrics, network traffic, and any data that could identify the system state. A data collector is necessary and can be provided by the system (e.g., in the ITS use case, the metrics are collected and sent to the RCA enabler via the MQTT broker), or an enabler can be deployed to collect different types of data, namely:

- Capturing network traffic: For example, the MMT-Probe [11] (TCP/IP networks) and the MMT-IoT [4] (IoT- 6LoWPAN) are able to sniff and record the network traffic.

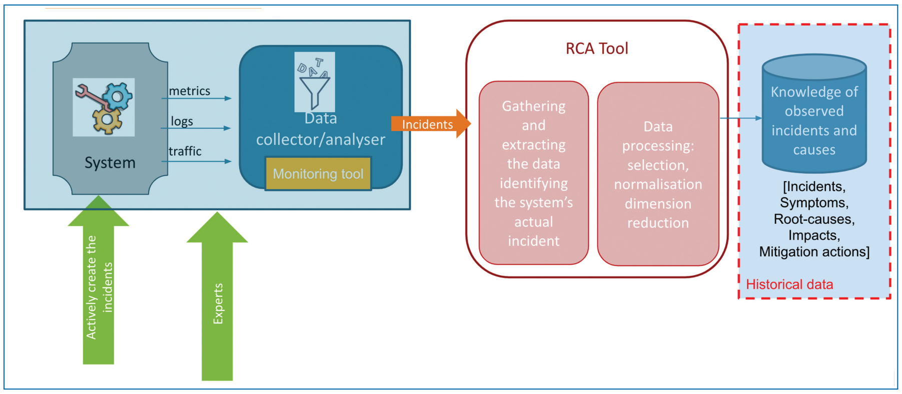

System
metrics
logs
traffic
Data collector/analyser
Monitoring tool
Incidents
RCA Tool
Gathering and extracting the data identifying the system's actual incident
Data processing: selection, normalisation dimension reduction
Knowledge of observed incidents and causes
[Incidents, Symptoms, Root-causes, Impacts, Mitigation actions]
Historical data
Actively create the incidents
Experts

Figure 8.23. RCA-Knowledge acquisition phase.

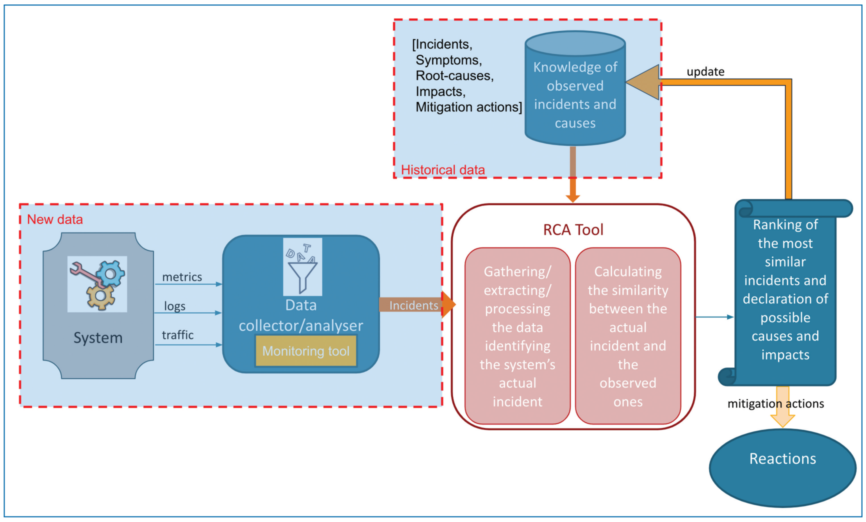

[Incidents, Symptoms, Root-causes, Impacts, Mitigation actions]
Knowledge of observed incidents and causes
Historical data
update
New data
System
metrics
logs
traffic
Data collector/analyser
Monitoring tool
Incidents
RCA Tool
Gathering/ extracting/ processing the data identifying the system's actual incident
Calculating the similarity between the actual incident and the observed ones
Ranking of the most similar incidents and declaration of possible causes and impacts
mitigation actions
Reactions

Figure 8.24. RCA-Monitoring phase.

- Reading and extracting logs: The current version of the RCA enabler supports by default reading the data input in the form of JSON and CSV files. Other formats can be rapidly taken into account thanks to the extensibility of MMT-Probe (e.g., creating new plugins).

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

- Actively injecting or reproducing known failures and attacks, then collecting labeled corresponding data.
- Passively monitoring the system, debugging different logs and traces, correlating various events and particularly by consulting the system experts to determine the corresponding root causes as well as the relevant data.

In the monitoring phase, the data is collected and transmitted to the RCA enabler in real-time. In theory, there is no restriction in the type of data to be gathered. On the contrary, a maximum of data for identifying the system functionalities is desirable. Even though some data could be redundant, data processing steps are performed to extract the most pertinent data.

8.3.2 Data Processing

As we mentioned in Section 8.1, there can be an enormous number of components/indicators in the analysis of an IoT system. In the following subsections, we discuss our techniques that avoid data noise, deal with heterogeneous data, and calculate the similarity between two different data sets.

8.3.2.1 Attribute selection

Attribute selection (also known as feature selection) [5] is one of the core concepts in Machine Learning that tremendously impacts the model performance. For complex systems, it is common that the data collected is too complicated or redundant. In other words, there might be some irrelevant or less important attributes (i.e., noises) contributing less to the target variable. Removing the noises helps not only to improve the accuracy but also to reduce the training time. It is the first and most essential step that should be performed automatically based on the feature selection techniques or manually by 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. The test score is the average of the scores calculated based on the chi-square test, the f test, and the mutual information classification test [5].
- Recursive feature elimination (RFE): It is about selecting 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 [7]) that assigns weights to features (e.g., linear model coefficients). The least important features are pruned step-by-step 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.

8.3.2.2 Data normalisation

Normalization is a concept informally used in statistics, and the term "normalized data" may have different meanings. In principle, data normalization means eliminating heterogeneous data measurement units and making the attributes comparable despite different value ranges.

In our perspectives, data normalization consists of two steps:

- Standardizing data to have a **mean** of zero and a **standard deviation**(s) of 1 (Equation (8.1) and Figure 8.25):

$$x_{standardized} = \frac{x - mean(x)}{s} \tag{8.1}$$

- Re-scaling the data to have values between 0 and 1 (Equation (8.2)):

$$x_{normalized} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{8.2}$$

8.3.2.3 Similarity calculation

Suppose that the system's temporal state 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 and dissimilarity of two states becomes the problem of measuring the distance of orientation (the angle) and magnitude (the length) of their two representing vectors. Figure 8.26 depicts an example in a 3-dimensional space.

The current version of the RCA enabler has been integrated with the following similarity and distance measures:

- Cosine similarity [2]
- Adjusted cosine similarity [2]
- Jaccard similarity [2]
- Euclidean distance [2]
- Manhattan distance [2]
- Minkowski distance [2]

These measures are used to calculate the similarity score whose value is between 0 and 1. The bigger the similarity score is, the more similar the two compared states are (e.g., if the similarity score is equal to 0.95, there is a 95% probability that two compared states are considered the equivalent). The similarity score can be computed based on one or multiple similarity and distance measures. In the training phase, we determine the measures. Therefore, when we compare a known

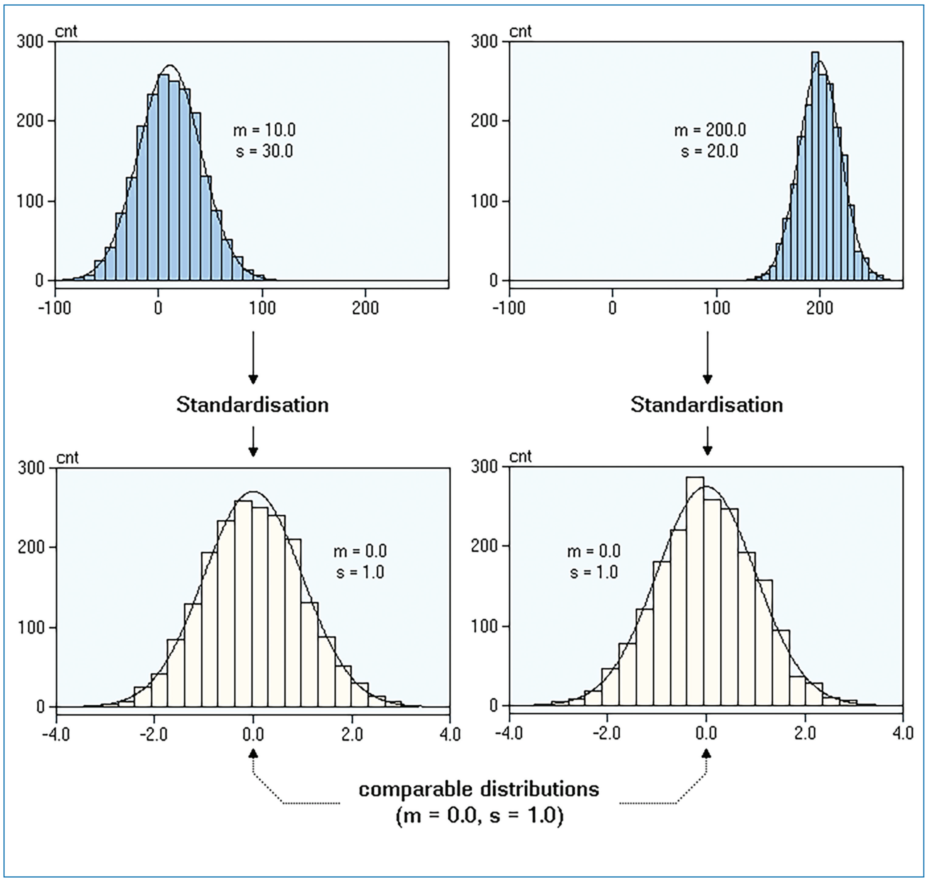


Figure 8.25. Data standardization.

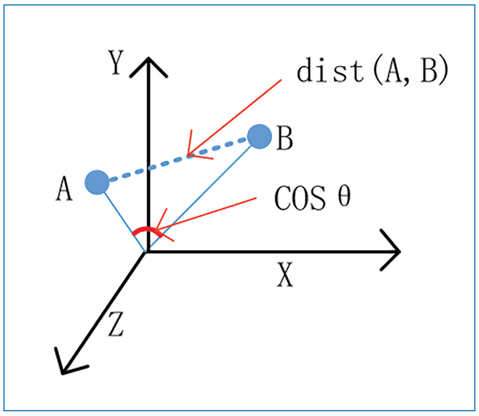
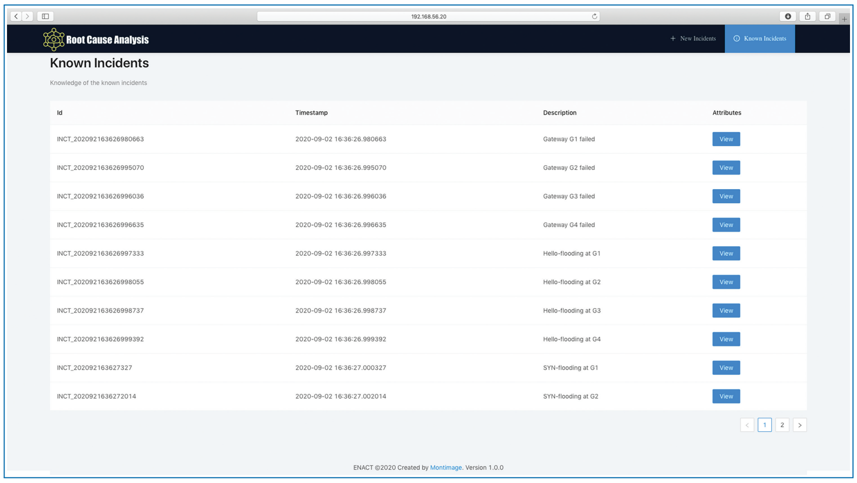


Figure 8.26. Similarity calculation in 3-dimensional space.



Known Incidents

Knowledge of the known incidents

Id	Timestamp	Description	Attributes
INCT_202092163626980663	2020-09-02 16:36:26.980663	Gateway G1 failed	View
INCT_202092163626995070	2020-09-02 16:36:26.995070	Gateway G2 failed	View
INCT_202092163626996036	2020-09-02 16:36:26.996036	Gateway G3 failed	View
INCT_202092163626996635	2020-09-02 16:36:26.996635	Gateway G4 failed	View
INCT_202092163626997333	2020-09-02 16:36:26.997333	Hello-flooding at G1	View
INCT_202092163626998055	2020-09-02 16:36:26.998055	Hello-flooding at G2	View
INCT_202092163626998737	2020-09-02 16:36:26.998737	Hello-flooding at G3	View
INCT_202092163626999392	2020-09-02 16:36:26.999392	Hello-flooding at G4	View
INCT_202092163627327	2020-09-02 16:36:27.000327	SYN-flooding at G1	View
INCT_2020921636272014	2020-09-02 16:36:27.002014	SYN-flooding at G2	View

< 1 2 >

ENACT ©2020 Created by Montimage. Version 1.0.0

Figure 8.27. Historical database of known incidents.

state and its repetition, we have a similarity score as big as possible. Besides, to avoid false positives, the similarity between a common proper state and each known malicious state should be as low as possible.

8.3.3 Reaction and Visualization

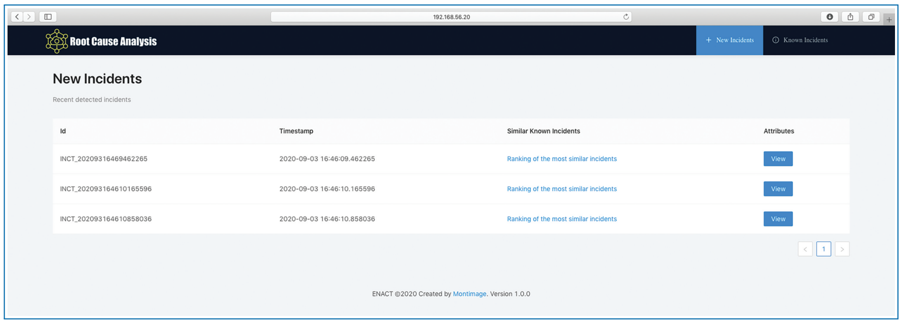
First, the RCA enabler analyses live the data originating from the system under monitoring. It reports back the similarity score of the current state, its most similar known incident, and the corresponding root causes. If the similarity score is higher than a given threshold, an alert is generated. The alert helps the system administrators foresee how the system evolves from a normal functioning state to a known fault or failure and determine the root causes to perform the most appropriate mitigation actions. For example, in the ITS use case, the RCA enabler communicates with the ITS through an MQTT connection. When the RCA enabler receives a message with all the relevant attributes, it identifies the level at which the system state and a known incident are similar. It publishes the result to the MQTT exchange.

Regarding the visualization, the results of the RCA enabler can be viewed intuitively on a GUI. Figure 8.27, Figure 8.28, Figure 8.29 displays some screenshots of RCA's GUI. More results are presented in the following sections.

8.3.4 Evaluation

8.3.4.1 Performance evaluation with generated testing data

To evaluate the RCA enabler, we first generated a learning data set in CSV format with several known records. Each record describes an "incident" with different



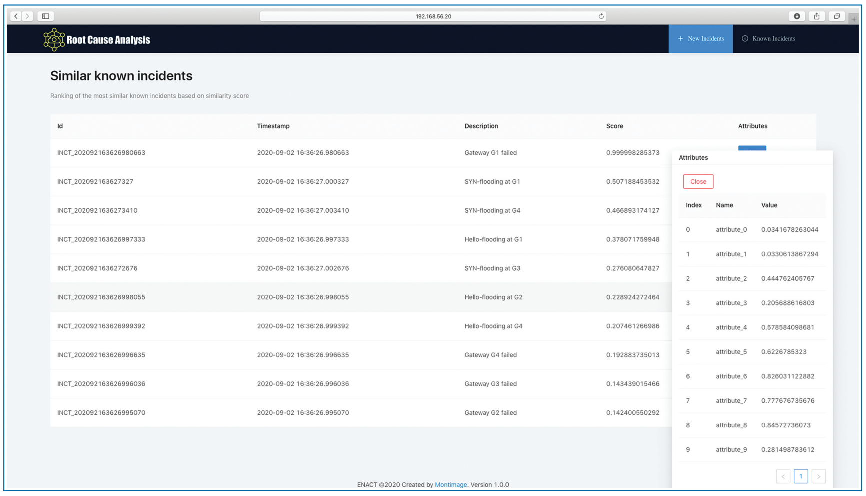
New Incidents

Recent detected incidents

Id	Timestamp	Similar Known Incidents	Attributes
INCT_20209316469462265	2020-09-03 16:46:09.462265	Ranking of the most similar incidents	View
INCT_202093164610165596	2020-09-03 16:46:10.165596	Ranking of the most similar incidents	View
INCT_202093164610858036	2020-09-03 16:46:10.858036	Ranking of the most similar incidents	View

ENACT ©2020 Created by Montimage. Version 1.0.0

Figure 8.28. Newly detected incidents.



Similar known incidents

Ranking of the most similar known incidents based on similarity score

Id	Timestamp	Description	Score	Attributes
INCT_202092163626980663	2020-09-02 16:36:26.980663	Gateway G1 failed	0.999998285373	
INCT_202092163627327	2020-09-02 16:36:27.000327	SYN-flooding at G1	0.507188453532	
INCT_2020921636273410	2020-09-02 16:36:27.003410	SYN-flooding at G4	0.466893174127	
INCT_202092163626997333	2020-09-02 16:36:26.997333	Hello-flooding at G1	0.378071759948	
INCT_2020921636272676	2020-09-02 16:36:27.002676	SYN-flooding at G3	0.276080647827	
INCT_202092163626998055	2020-09-02 16:36:26.998055	Hello-flooding at G2	0.228924272464	
INCT_202092163626999392	2020-09-02 16:36:26.999392	Hello-flooding at G4	0.207461266986	
INCT_202092163626996635	2020-09-02 16:36:26.996635	Gateway G4 failed	0.192883735013	
INCT_202092163626996036	2020-09-02 16:36:26.996036	Gateway G3 failed	0.143439015466	
INCT_202092163626995070	2020-09-02 16:36:26.995070	Gateway G2 failed	0.142400550292	

Attributes

Close

Index	Name	Value
0	attribute_0	0.0341678263044
1	attribute_1	0.0330613867294
2	attribute_2	0.444762405767
3	attribute_3	0.205688616803
4	attribute_4	0.578584098681
5	attribute_5	0.6226785323
6	attribute_6	0.826031122882
7	attribute_7	0.777676735676
8	attribute_8	0.84572736073
9	attribute_9	0.281498783612

ENACT ©2020 Created by Montimage. Version 1.0.0

Figure 8.29. A newly detected incident and its similarity scores in comparison with known ones.

"attribute" values. These learned "incidents" are stored together with the potential origin causes. The "attributes" refer to the metric values that can be gathered. Afterward, we generate new attributes and check whether the system recognizes them as a known incident among the ones that are already in the historical data. The RCA enabler measures the similarity of each new record and each learned incident. It ranks them by determining the most likely similar ones. To assess the performance, we calculate the enabler's response time with the function taking as input the number of known states and the attributes identifying a state. In this evaluation, the similarity score is the average of all similarity measures mentioned in Section 8.3.2.3.

Figure 8.30 and Figure 8.31 show the results of our experiments. As expected, more time is needed when the data volume handled increases. However, we observe that the number of known states has less impact on the response time than the

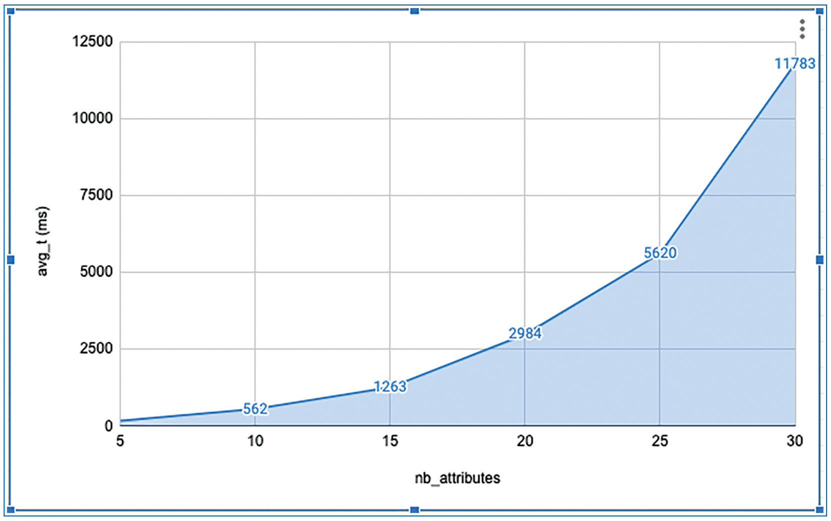


Figure 8.30. RCA Enabler's response time towards the number of attributes.

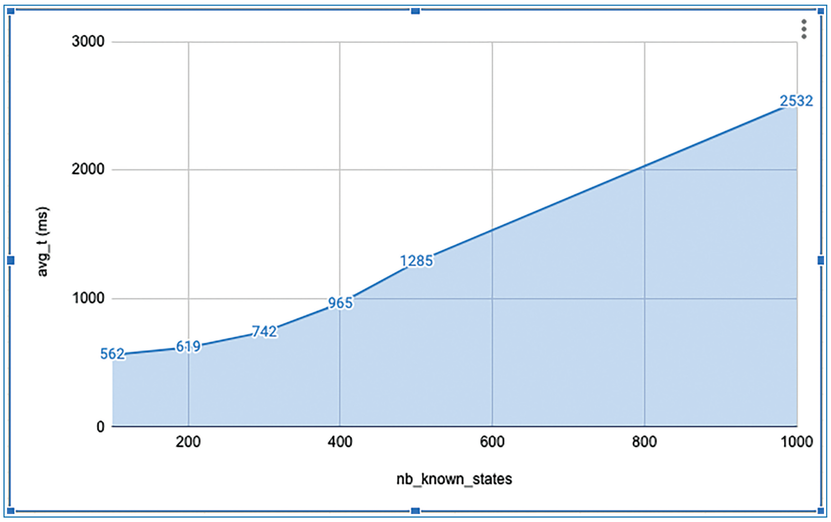


Figure 8.31. RCA Enabler's response time towards the number of known incidents.

number of the attributes has. The processing time needed increases more drastically when more attributes are under consideration than when there are more known states. This increase reaffirms the need for integrating "attribute selection" as aforementioned in Section 8.3.2.1. In our evaluation, we did not apply any selection technique because the data was generated randomly. The attributes are, thus, seemingly equal and make the selection not useful. However, there will probably be a different story when real systems are involved (further discussed in Section 8.3.4.2).

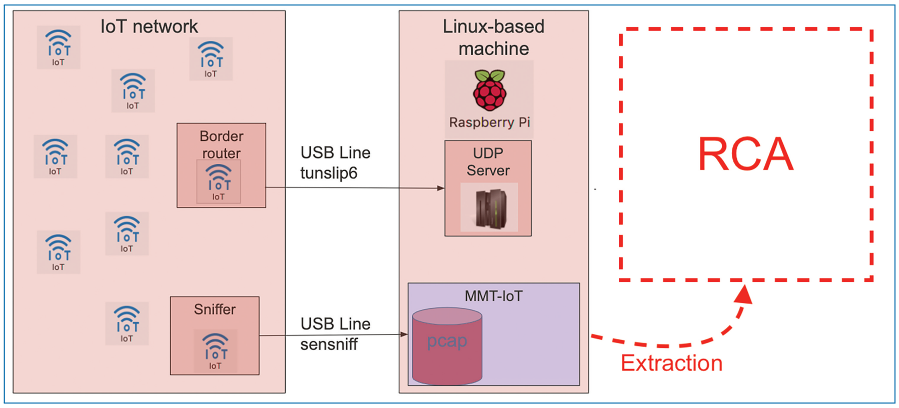


Figure 8.32. General architecture for the evaluation.

8.3.4.2 Evaluation on a real IoT Testbed

Set-up of the experiment

To evaluate the RCA enabler, we performed several experiments on a real IoT testbed called w-iLab.t[1] provided by Imec.[2] Figure 8.32 presents the general architecture of the experiments. Several IoT devices (Zolertia Re-Mote[3]) formed an IoT network where the clients reported 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 to perform sniffing tasks: capturing network traffic and piping via the USB line to the Linux-based machine where MMT-IoT was deployed to analyze the traffic and extract the metrics for the RCA enabler. Besides, the IoT network consisted of normal clients reporting sensed data every 10 seconds, and one (or several) attacker(s) behaved 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[4]).
- Dead mode not reporting data at all (node failure).

1. https://www.fed4fire.eu/testbeds/w-ilab-t/
2. https://www.imec-int.com/
3. https://zolertia.io/zolertia-platforms/
4. FCS: The FCS field contains a number calculated by the source node based on the data in the frame. This number is added to the end of a frame sent. When the destination node receives the frame, the FCS number is recalculated and compared with the number sent in the frame. If they are different, the frame is considered malformed (intentionally or not) or modified between the source node and the destination node.

Table 8.2. Attributes extracted and sent to RCA.

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

Attribute selection and data normalisation

As the first step, the RCA enabler selects the significant attributes among the 12 listed in Table 10. The selection is done by applying the techniques aforementioned in Section 8.3.2.1.

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).

Table 8.3. Feature Selection results.

	Univariate feature selection			Recursive feature elimination	
Ref.	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 8.3 summarizes the results when different Feature Selection models are used. There are six attributes, namely (1–3), (5–7), which are considered significant according to all the models. Four attributes (8), (10), (11), and (12) are concluded to be not relevant and can be left out. The attributes (4) and (9) are recommended by some models and not by others. We performed the analysis in the following subsection with these two attributes and the other six attributes recommended by all the models.

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 8.33).
- The attacker was evidently the most active device (Figure 8.34) and one end of the most active link (Figure 8.35).

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

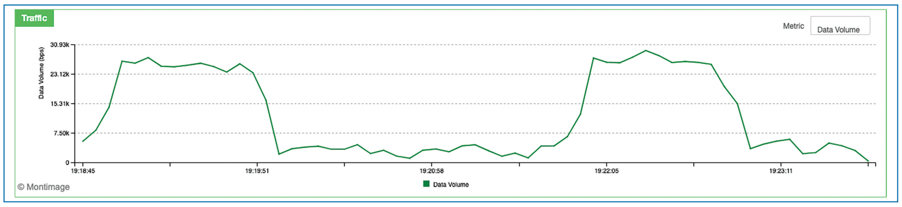
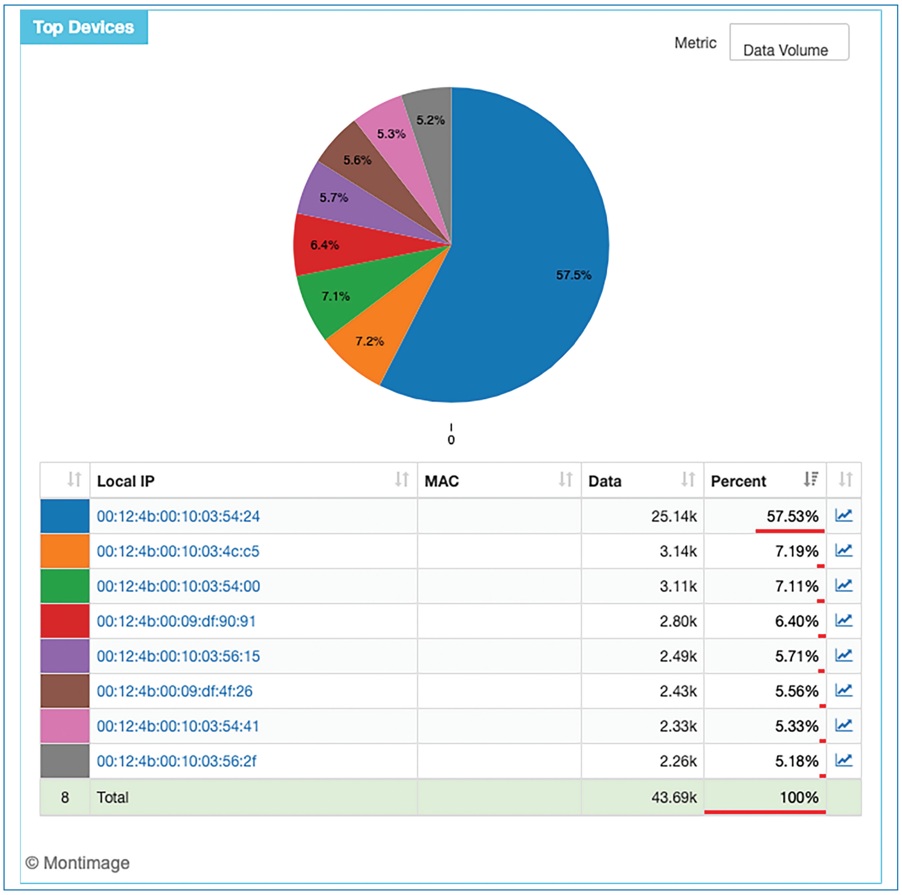


Figure 8.33. Traffic throughput increased remarkably when the attack took place.

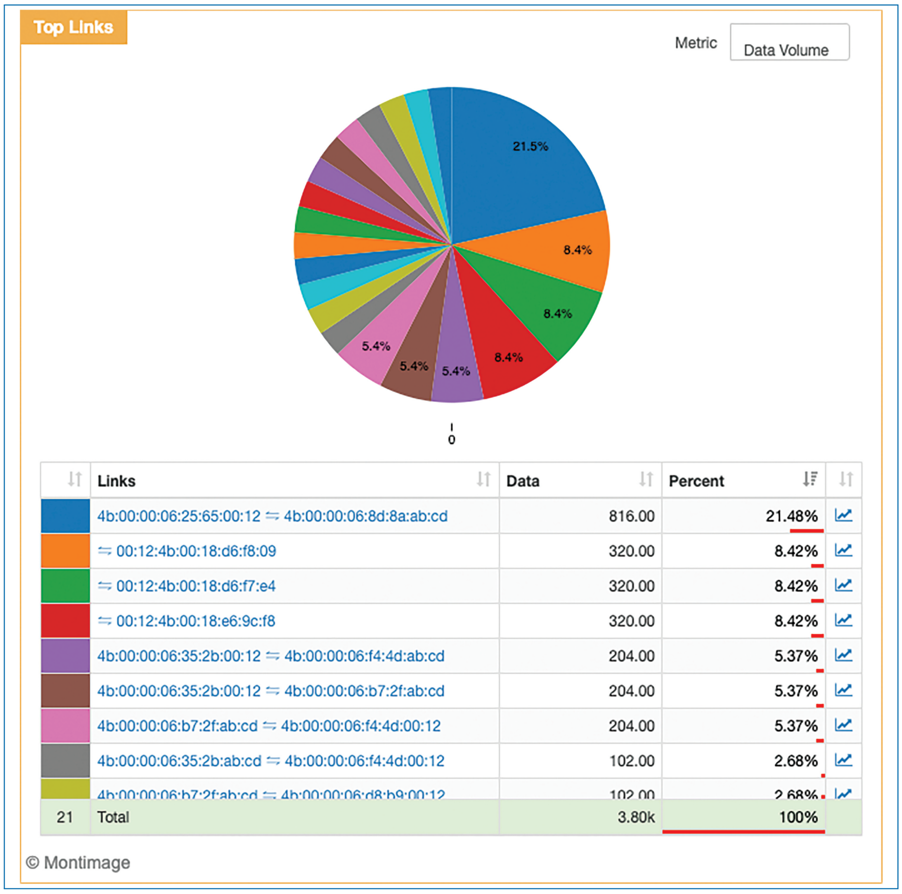


	Local IP	MAC	Data	Percent	
🟦	00:12:4b:00:10:03:54:24		25.14k	57.53%	
🟧	00:12:4b:00:10:03:4c:c5		3.14k	7.19%	
🟩	00:12:4b:00:10:03:54:00		3.11k	7.11%	
🟥	00:12:4b:00:09:df:90:91		2.80k	6.40%	
🟪	00:12:4b:00:10:03:56:15		2.49k	5.71%	
🟫	00:12:4b:00:09:df:4f:26		2.43k	5.56%	
■	00:12:4b:00:10:03:54:41		2.33k	5.33%	
■	00:12:4b:00:10:03:56:2f		2.26k	5.18%	
8	Total		43.69k	100%	

Figure 8.34. The attacker was the most active device.

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, we computed the similarity score based on the "adjusted cosine similarity".

In node failure (dead device), all the attributes related to the dead device were affected. The RCA enabler reported a similarity score between 0.84 and 0.87 when the failure repeated.



	Links	Data	Percent	
🟦	4b:00:00:06:25:65:00:12 ⇆ 4b:00:00:06:8d:8a:ab:cd	816.00	21.48%	📈
🟧	⇆ 00:12:4b:00:18:d6:f8:09	320.00	8.42%	📈
🟩	⇆ 00:12:4b:00:18:d6:f7:e4	320.00	8.42%	📈
🟥	⇆ 00:12:4b:00:18:e6:9c:f8	320.00	8.42%	📈
🟪	4b:00:00:06:35:2b:00:12 ⇆ 4b:00:00:06:f4:4d:ab:cd	204.00	5.37%	📈
🟫	4b:00:00:06:35:2b:00:12 ⇆ 4b:00:00:06:b7:2f:ab:cd	204.00	5.37%	📈
🟪	4b:00:00:06:b7:2f:ab:cd ⇆ 4b:00:00:06:f4:4d:00:12	204.00	5.37%	📈
⬛	4b:00:00:06:35:2b:ab:cd ⇆ 4b:00:00:06:f4:4d:00:12	102.00	2.68%	📈
🟨	4b:00:00:06:b7:2f:ab:cd ⇆ 4b:00:00:06:d8:b9:00:12	102.00	2.68%	📈
21	Total	3.80k	100%	

Figure 8.35. The attacker belonged to the most active link.

Lastly, when the border router was affected by the jamming attack of incorrect FCS values, its CPU usage jumped virtually. When this behavior happened again, the RCA enabler determined that it was up to 94% similar to the learned incident's observed symptoms. However, even when all the network devices worked normally, the RCA enabler identified that there was up to 78% similarity between this incident and the event "Possible jamming attack with incorrect FCS values".

8.4 Conclusion

In conclusion, This chapter presents two tools, i.e., the TaS and RCA enablers, that enable the validation and verification of IoT systems. The TaS enabler, based on the idea of "Digital Twins", is a Software-as-a-Service solution that provides

(i) a flexible simulation of sensor networks, (ii) a powerful data generator with real-time data recording, and (iii) support for Continuous Integration and Continuous Development. It helps IoT application developers save time and money on setting up the testing environment and thus supports faster application delivery. The RCA enabler systematizes the knowledge about the potential incidents that may occur in the system. It prevents the incidents or to quickly and intelligently react against their recurrences.

In practice, we can apply the TaS and RCA enablers to various systems other than IoT systems in the context of ENACT. In general, they can work on any system in which the data about the system's functioning state can be collected. For the RCA enabler, it would be beneficial if the owner or administrator has already acquired a certain level of understanding about the system to facilitate the training phase and the database creation for known incidents and root causes. Otherwise, we can perform penetration tests to discover potential vulnerabilities so that the attacks and failures can be injected and learned.

Moreover, both tools have been developed to be generic enough so that adaptations can be easily made to make them applicable to various types of systems (e.g., industrial SCADA systems, 5G mobile networks). We plan to adapt and use these two tools in several other collaborative projects in different contexts. We hope they will play a crucial role in the Montimage ecosystem and be commercialized within the MMT Monitoring solution.[5]

References

[1] C. Adjih *et al.* "FIT IoT-LAB: A large scale open experimental IoT testbed". In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. 2015, pp. 459–464. DOI: 10.1109/WF-IoT.2015.7389098.

[2] Marc Sebban Aurélien Bellet, Amaury Habrard. *Metric Learning, Similarity-Based Pattern Analysis and Recognition*. Morgan and Claypool, 2015. ISBN: 1939-4616.

[3] A. Fuller *et al.* "Digital Twin: Enabling Technologies, Challenges and Open Research". In: *IEEE Access* 8 (2020), pp. 108952–108971. DOI: 10.1109/ACCESS.2020.2998358.

[4] Vinh Hoa La, Raul Fuentes, and Ana R. Cavalli. "A Novel Monitoring Solution for 6LoWPAN-based Wireless Sensor Networks". In: *Proceedings of 22nd Asia-Pacific Conference on Communications (APCC 2016)*. 2016.

5. https://montimage.com/products/MMT_DPI.html

[5] Richard Lowry. *Concepts and Applications of Inferential Statistics*. Vassar College, 2008.

[6] Sara N. Matheu-García *et al.* “Risk-based automated assessment and testing for the cybersecurity certification and labelling of IoT devices”. In: *Computer Standards & Interfaces* 62 (2019), pp. 64–83. ISSN: 0920-5489. DOI: https://doi.org/10.1016/j.csi.2018.08.003. URL: https://www.sciencedirect.com/science/article/pii/S0920548918301375.

[7] Kjell Johnson Max Kuhn. *Feature Engineering and Selection: A Practical Approach for Predictive Models*. Taylor & Francis, 2019.

[8] Marcello Pelillo. *Similarity-Based Pattern Analysis and Recognition*. Springer, 2013. ISBN: 978-1-4471-5628-4.

[9] Luis Sanchez *et al.* “SmartSantander: IoT experimentation over a smart city testbed”. In: *Computer Networks* 61 (2014). Special issue on Future Internet Testbeds Part I, pp. 217–238. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.bjp.2013.12.020.

[10] M. Tappler, B. K. Aichernig, and R. Bloem. “Model-Based Testing IoT Communication via Active Automata Learning”. In: *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. 2017, pp. 276–287. DOI: 10.1109/ICST.2017.32.

[11] B. Wehbi, E. Montes de Oca, and M. Bourdelles. “Events-Based Security Monitoring Using MMT Tool”. In: *IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST), 2012*. Apr. 2012, pp. 860–863. DOI: 10.1109/ICST.2012.188.