



European  
Commission

Horizon 2020  
European Union funding  
for Research & Innovation

## Cyber Security PPP: Addressing Advanced Cyber Security Threats and Threat Actors



### Cyber Security Threats and Threat Actors Training - Assurance Driven Multi- Layer, end-to-end Simulation and Training

#### **D2.4: Emulation tool interoperability module v1 <sup>†</sup>**

**Abstract:** This deliverable is the result of the first iteration of task T2.4 activities. It defines the technical means and type of interfaces for interconnecting the Emulation Tool with the relevant platform components such as with the Simulation Tool, the Training and Visualization Tools, and the Data Fabrication Platform. The document is the first version of the means of communications. Its goal is to guide the Emulation Tool integration activities in the second year of the project and proper interconnection with the other platform components.

Contractual Date of Delivery	31/08/2019
Actual Date of Delivery	31/08/2019
Deliverable Security Class	Public
Editor	<i>Hristo Koshutanski (ATOS)</i>
Contributors	<i>George Hatzivasilis (FORTH), Fulvio Frati (UMIL)</i>
Quality Assurance	<i>George Bravos (ITML), Torsten Hildebrandt (SIMPLAN), George Hatzivasilis (FORTH).</i>

---

<sup>†</sup> The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 786890.

### **The *THREAT-ARREST* Consortium**

Foundation for Research and Technology – Hellas (FORTH)	Greece
SIMPLAN AG (SIMPLAN)	Germany
Sphynx Technology Solutions (STS)	Switzerland
Universita Degli Studi di Milano (UMIL)	Italy
ATOS Spain S.A. (ATOS)	Spain
IBM Israel – Science and Technology LTD (IBM)	Israel
Social Engineering Academy GMBH (SEA)	Germany
Information Technology for Market Leadership (ITML)	Greece
Bird & Bird LLP (B&B)	United Kingdom
Technische Universitaet Braunschweig (TUBS)	Germany
CZ.NIC, ZSPO (CZNIC)	Czech Republic
DANAOS Shipping Company LTD (DANAOS)	Cyprus
TUV HELLAS TUV NORD (TUV)	Greece
LIGHTSOURCE LAB LTD (LSE)	Ireland
Agenzia Regionale Strategica per la Salute ed il Sociale (ARESS)	Italy

## Document Revisions & Quality Assurance

### Internal Reviewers

1. *George Bravos (ITML)*
2. *Torsten Hildebrandt (SIMPLAN)*
3. *George Hatzivasilis (FORTH)*

### Revisions

Version	Date	By	Overview
0.7	27/08/2019	Editor	Addressed the comments by FORTH from the quality review process
0.6	22/08/2019	Editor	Addressed the comments by SIMPLAN and ITML from the quality review process
0.5	02/08/2019	Editor	ATOS' contribution to Sections 1 and 4
0.4	01/08/2019	UMIL	UMIL's contribution to Section 3 and revision of Section 2
0.3	30/07/2019	Editor	ATOS' contribution to Sections 2 and 3
0.2	18/06/2019	FORTH	FORTH's initial contribution to Section 2
0.1	20/05/2019	Editor	First Draft with ToC

## **Executive Summary**

This document is the result of the first iteration of task T2.4 and reports the work performed by month 12 of the project. It steps on and extends the results of “D1.3 – THREAT-ARREST platform’s initial reference architecture” to define the technical means and interfaces for interconnecting the Emulation Tool with other platform components, such as the Simulation Tool, the Training Tool, the Visualisation Tool, and the Data Fabrication Platform.

The goal of this first version is to guide the Emulation Tool integration activities in the second year of the project and proper interconnection with the other platform components. Particularly, this document relates to work package (WP) 6 activities on system integration starting in month 13 of the project.

Importantly, this document has two other counterpart documents – the deliverables “D4.3 – Training and Visualisation tools IO mechanisms v1” and “D5.3 – The Simulation component IO module v1”. These two other deliverables address in a similar but complementary way the interconnections of the other platform tools, and altogether provide an overall view of THREAT-ARREST platform interconnections for year 1 of the project.

## Table of Contents

<b>1</b>	<b>INTRODUCTION .....</b>	<b>9</b>
<b>2</b>	<b>MESSAGE BROKER AND REST COMMUNICATIONS.....</b>	<b>11</b>
2.1	MESSAGE BROKER-ENABLED COMMUNICATIONS .....	11
2.1.1	<i>Standard RabbitMQ Message Flow .....</i>	<i>14</i>
2.1.2	<i>RabbitMQ Topic Exchange.....</i>	<i>15</i>
2.2	REST COMMUNICATIONS.....	15
<b>3</b>	<b>EMULATION TOOL INTERCONNECTIONS .....</b>	<b>17</b>
3.1	INITIALISATION OF THE EMULATION TOOL.....	18
3.2	INTERCONNECTION WITH THE SIMULATION TOOL.....	19
3.3	INTERCONNECTION WITH THE TRAINING AND VISUALISATION TOOLS .....	19
3.4	INTERCONNECTION WITH THE DATA FABRICATION PLATFORM.....	22
<b>4</b>	<b>CONCLUSIONS AND NEXT STEPS .....</b>	<b>24</b>
<b>5</b>	<b>REFERENCES .....</b>	<b>25</b>

## List of Abbreviations

<b>AMQP</b>	Advanced Message Queuing Protocol
<b>API</b>	Application Programming Interface
<b>CoAP</b>	Constrained Application Protocol
<b>CoRE</b>	Constrained RESTful environments
<b>CTTP</b>	Cyber Threat and Training Preparation
<b>DFP</b>	Data Fabrication Platform
<b>DTLS</b>	Datagram Transport Layer Security
<b>HTTP</b>	Hypertext Transfer Protocol
<b>HTTPS</b>	Hypertext Transfer Protocol Secure
<b>IEC</b>	International Electrotechnical Commission
<b>IETF</b>	Internet Engineering Task Force
<b>ISO</b>	International Organization for Standards
<b>IoT</b>	Internet of Things
<b>MQTT</b>	Message Queuing Telemetry Transport
<b>OASIS</b>	Organization for the Advancement of Structured Information Standards
<b>QoS</b>	Quality of Service
<b>REST</b>	Representational State Transfer
<b>STOMP</b>	Simple Text Oriented Messaging Protocol
<b>TCP</b>	Transmission Control Protocol
<b>TLS</b>	Transport Layer Security
<b>UDP</b>	User Datagram Protocol
<b>VM</b>	Virtual Machine
<b>WP</b>	Work Package

## List of Figures

Figure 1 Basic steps to create an application with RabbitMQ .....	12
Figure 2 Sequence diagram for discovery operation.....	13
Figure 3 Sequence diagram for event subscription operation .....	13
Figure 4. Standard RabbitMQ message flow .....	14
Figure 5 The REST architecture and the supported operations .....	16
Figure 6 THREAT-ARREST Platform Components Communications – Emulation Tool View .....	18
Figure 7: API emulation/getVMfromXML Response Values. ....	19
Figure 8 Emulation Tool Interconnection with the Training and Visualisation Tools .....	21

## List of Code Examples

Code Example 1: RabbitMQ Java API for Topic Exchange Creation and Message Publishing	22
--	----



# 1 Introduction

This deliverable defines the technical means and interfaces for interconnecting the Emulation Tool with the relevant platform components, such as:

- the interplay with the Simulation Tool enabling a hybrid partially emulated and simulated ecosystem,
- the integration with the Training and Visualization Tools,
- as well as the input of realistic synthetic logs and/or events from the Data Fabrication Platform (DFP).

The document is the first version of the means of communications that will be used by the Emulation Tool as guidelines in the second year of the project to interconnect with the other platform components.

We note that the Emulation Tool does not need to interconnect with all tools of the platform (e.g. the Gamification Tool or the Assurance Tool) but only with those identified important for the proper functioning of the THREAT-ARREST platform. If in the second year of the project interconnections with other platform tools are identified important, these will follow any of the means established in this document.

The Emulation Tool upon initialisation creates an environment (infrastructure) where targeted cyber system components and functionalities are emulated. To do so, it offers REST APIs for proper lifecycle management of cyber system emulation, such as initialisation and finalisation. The Training Tool initialises cyber system emulation by using the corresponding API. The Emulation Tool starts the initialisation by retrieving the instantiation scripts from the deployed CTP models. To achieve a hybrid – partially emulated and partially simulated – cyber system, the Emulation Tool needs to properly interconnect the created emulation environment with the Simulation Tool's environment. In addition, the Emulation Tool needs to properly interconnect with the Training and Visualisation Tools offering real time information about the status of emulated components during the training phase in order to enable their effective visualization and user assessment of the overall progress in training, respectively. Finally, the Emulation Tool interconnects with the DFP for the generation and acquisition of synthetic data regarding security event logs or general synthetic data for the training scenario.

Upon successful cyber system emulation or simulation, the Training Tool interconnects with the Assurance Tool to initialise the monitoring of the trainee's actions against the actual cyber system of the organisation and get the necessary data for CTP programmes' evaluation. The Assurance Tool may retrieve and update CTP models stored in the platform for this purpose. The main role of the Assurance Tool is to monitor and assess the security posture of the actual cyber system of the organisation where the THREAT-ARREST training platform is used at. Given that, it has not been defined any direct communication of the Emulation Tool with the Assurance Tool.

We recall that this document has two other counterpart documents – the deliverables D4.3 (THREAT-ARREST D4.3, 2019) and D5.3 (THREAT-ARREST D5.3, 2019). These two other deliverables address in a similar but complementary way the interconnections of the other platform tools. In this way, the three documents D2.4, D4.3, and D5.3 provide an overall view of THREAT-ARREST platform interconnections for year 1 of the project.

For convenience of readers and to facilitate material comprehension, we recall Section 2 across the three documents with the aim to have a more self-contained version of the documents.

This document is structured as following. Section 2 overviews the main means of communications supporting the various platform components – communications via either a message broker (i.e. RabbitMQ (Richardson, 2014; Lonescu, 2015)) or Representational State

Transfer (REST) interfaces. Section 3 presents in detail the Emulation Tool interconnections with the abovementioned platform components. Section 4 concludes the document and outlines next steps towards the second and final version of this document (D2.7 – Emulation tool interoperability module interoperability module v2).

## 2 Message Broker and REST Communications

This section describes the communication channels between the various platform components. Two main options are supported via either a message broker or Representational State Transfer (REST) interfaces (Fielding, 2000).

### 2.1 Message Broker-enabled Communications

Message-oriented protocols typically focus on providing asynchronous data transfers between distributed devices (Hatzivasilis et al., 2018a; Hatzivasilis et al., 2018b; Lakka et al. 2019). Their focus is on reliable messaging, including message buffers and Quality of Service (QoS) facilities, controlled by centralized entities. By using the message broker-enabled communication, messages are passed through a central server (the Broker), enabling *one-to-many* and *many-to-many* interactions. This offloads the computational power needed for a component to connect many different clients in order to exchange messages.

The Message Queuing Telemetry Transport (MQTT) (Banks and Gupta, 2014) is one such message-oriented protocol, introduced by IBM in 1999 and recently standardized by the Organization for the Advancement of Structured Information Standards (OASIS)<sup>1</sup>, as the Internet of Things (IoT) developments brought it back into the limelight. It is also standardized as by the International Organization for Standards (ISO) and the International Electrotechnical Commission (IEC) as ISO/IEC 20922 (ISO/IEC, 2016). MQTT was designed as an extremely lightweight publish/subscribe messaging transport, for small sensors and mobile devices, optimized for high-latency or unreliable networks. A MQTT Broker is responsible for handling and organizing all communications between the various devices/components. Messages are published with specific *topics*, and each client can subscribe to various topics (though the Broker may require username/password authentication before allowing subscription). Topics are organized in a hierarchical manner, like the folder structure in a file system (e.g. “THREAT-ARREST/CTTP/models” could be a topic where a component can subscribe to get updates on the CTTP models). When a client publishes a message, the Broker then relays this message to all clients which are subscribed to the message's topic. Thus, all interactions are asynchronous and clients only communicate directly with the Broker. MQTT relies on the Transmission Control Protocol (TCP) and secure deployments support the use of the Transport Layer Security (TLS) protocol. The protocol is designed to be used even on lightweight devices, like mobile devices and embedded systems where bandwidth is costly and minimum overhead required. It uses a 2-byte fixed header to control everything and exchange data as byte stream. Therefore, MQTT is being used widely in IoT settings.

The Simple Text Oriented Message Protocol<sup>2</sup> (STOMP) is a simple text-based protocol with a main goal to interoperate with message-oriented middleware. The protocol wire format is suitable to allow any STOMP client to communicate with any message broker which supports the protocol. The protocol runs on any TCP-enabled communications following well-defined commands such as CONNECT, SEND, SUBSCRIBE, UNSUBSCRIBE, BEGIN, COMMIT, ABORT, etc. Importantly, STOMP is designed for *asynchronous* message passing between *lightweight* entities/clients coming from scripting languages such as Ruby, Python, Perl or JavaScript. In such a client environment, simple operations are typically carried reliably such as reliably sending single messages or consume messages on a given destination. STOMP can be seen as an alternative to other open messaging protocols, such as the Advanced Message Queuing Protocol (AMQP) (Luzuriaga et al., 2015), but covering a small subset of commonly used messaging operations. Given its design principles, STOMP has been a definitive choice for

---

<sup>1</sup> OASIS: “MQTT 3.1.1 specification,” December 10, 2015, <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>

<sup>2</sup> <https://stomp.github.com/> ; <http://stomp.github.io/stomp-specification-1.2.html>

some THREAT-ARREST components' communications such as those of the Visualisation Tool.

The Advanced Message Queuing Protocol (AMQP) (Luzuriaga et al., 2015) is an open standard for passing business messages between applications or organizations. AMQP is designed for reliable communication via message delivery guarantee primitives, like *at-most-one*, *at-least-once*, and *exactly-one* delivery, and it is built upon a reliable transport protocol, such as TCP. The protocol consists of two core components that handle communication: the exchanges and the message queues. Based on pre-defined rules, the exchanges route the messages to appropriate queues, which can store the data and later send it to the receivers. It connects systems, feeds business processes with the information they need and reliably transmits onward the instructions that achieve their goals. The protocol is designed with more advanced features in mind and has more overhead than MQTT. For this reason, AMQP is not preferred for lightweight devices (e.g. mobile), while MQTT can be used almost anywhere. But in real world application development, we may need AMQP for reliable message queue while having lightweight devices to work with. Here is the point where RabbitMQ<sup>3</sup> comes in.

RabbitMQ (Richardson, 2014; Lonescu, 2015) is lightweight and easy to deploy on premises and in the cloud. It supports multiple messaging protocols (e.g. MQTT and AMQP). It can be deployed in distributed and federated configurations to meet high-scale and high-availability requirements. This implementation can be run on a wide variety of platforms. RabbitMQ can potentially run on any platform that provides a supported Erlang<sup>4</sup> version, from multi-core nodes and cloud-based deployments to embedded systems. In particular, OpenStack supports RabbitMQ as message queue service and use it in many of its modules<sup>5</sup>. Figure 1 illustrates the basic steps for creating an application with RabbitMQ (Lonescu, 2015).

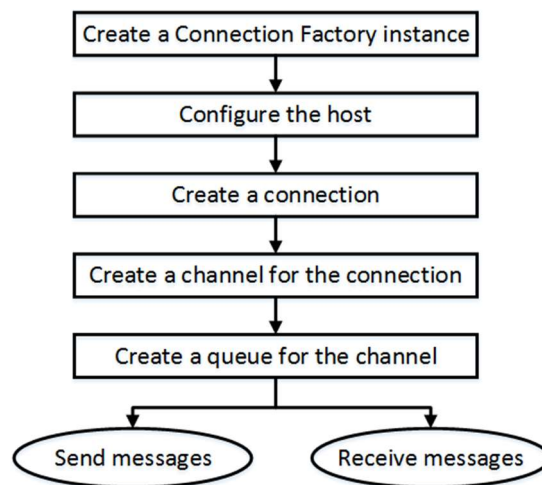


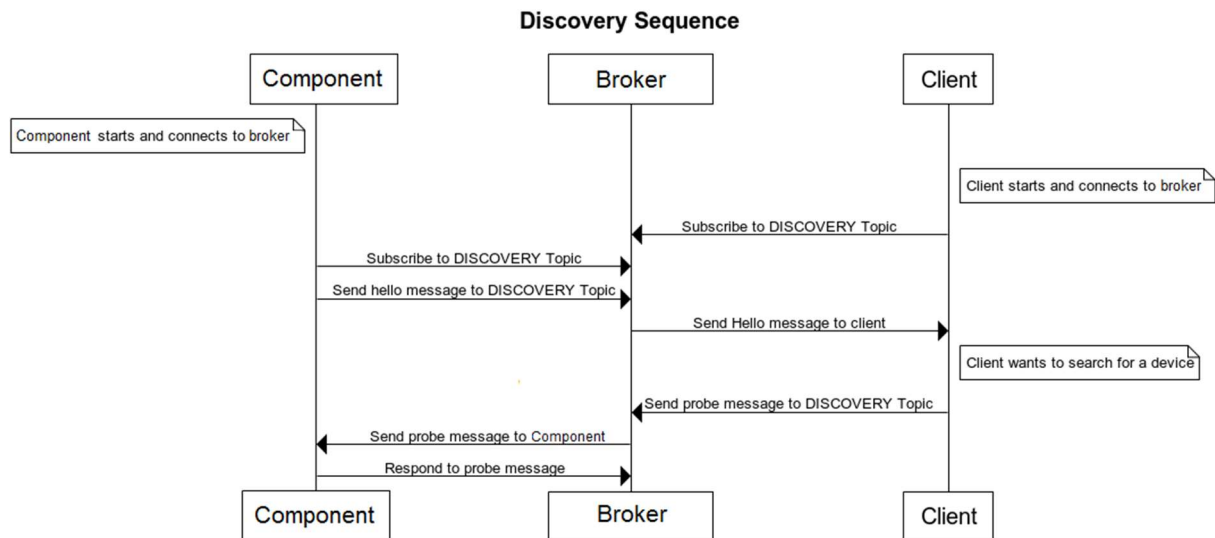
Figure 1 Basic steps to create an application with RabbitMQ

First, the components publish their profile information to the broker, including the relevant IP address. The broker can be either local or remote, enabling cross-domain interaction. In order to discover a component or service, the actuator sends a request message to all public components through the broker, which implements the corresponding multicasting functionality. The compatible entities respond to the request by sending descriptive metadata.

<sup>3</sup> RabbitMQ: <http://www.rabbitmq.com>

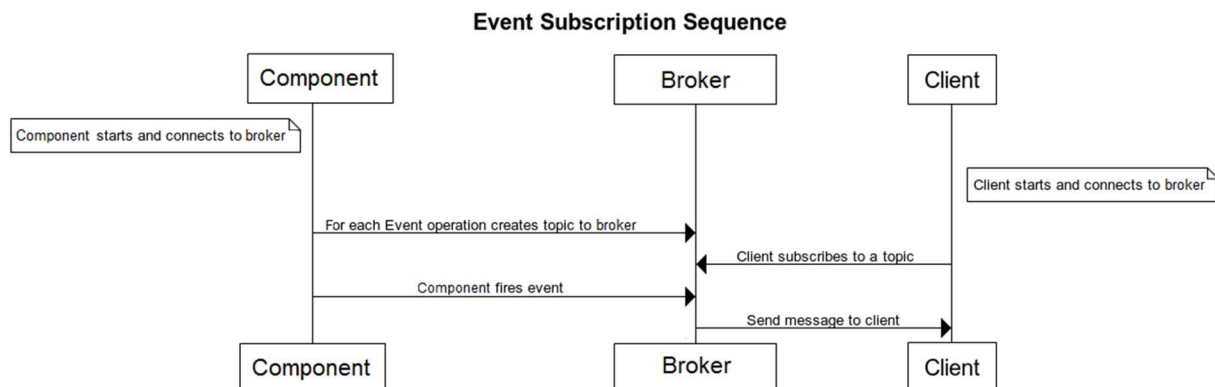
<sup>4</sup> <https://www.erlang.org>

<sup>5</sup> <https://docs.openstack.org/mitaka/install-guide-ubuntu/environment-messaging.html>



*Figure 2 Sequence diagram for discovery operation*

Figure 2 illustrates a sequence diagram of the discovery operation. For asynchronous operation, subscribe or eventing, the messages are passed through the broker. Figure 3 illustrates a sequence diagram of the event subscription operation.



*Figure 3 Sequence diagram for event subscription operation*

RabbitMQ supports AMQP, MQTT, STOMP and WEBSOCKETS as message delivery protocols. This means that consumer and producer services can be implemented not only by using different platforms and languages, but also by different messaging protocols. It has a wide community and we can find a rich documentation on many different programming languages, such as Python, Java, PHP, JavaScript, Go, etc. (Richardson, 2014; Lonescu, 2015).

The most important features of RabbitMQ for the THREAT-ARREST project include the guaranteed delivery and the message queue implementation (Lakka et al. 2019; Hatzivasilis et al., 2019). To sum-up, we choose the RabbitMQ broker for the internal THREAT-ARREST platform communications, as:

- It is an open source message queuing system.
- It constitutes an ideal choice for interoperability between applications and tools of different protocols and between different programming languages.

- The fact that we can publish messages into one environment via one protocol and consume them via one or more other protocols (simultaneously if necessary).
- It is a popular open source message queuing system that implements the AMQP.
- It well describes all supported protocols and their purpose.
- There is an active community and RabbitMQ has been utilized in very different application areas.
- RabbitMQ offers libraries/APIs available in many programming languages<sup>6</sup> allowing, with just a few lines of code, the creation of communication channels to a broker, the creation of queues, and publishing and receiving messages on channels and queues respectively.
- It is fully supported by OpenStack<sup>7</sup> the underpinning technology of the THREAT-ARREST Emulation Tool.

### 2.1.1 Standard RabbitMQ Message Flow

In the following, we will overview the basic message flow concept of RabbitMQ to facilitate the presentation in the following sections. In RabbitMQ, the producer's messages are not published directly to a consumer but instead, the producer sends messages to an *Exchange*. An Exchange is a message routing agent responsible for routing of messages to different queues. An Exchange accepts messages from the producer application and routes them to message queues with the help of header attributes, bindings, and routing keys (Johansson, 2015).

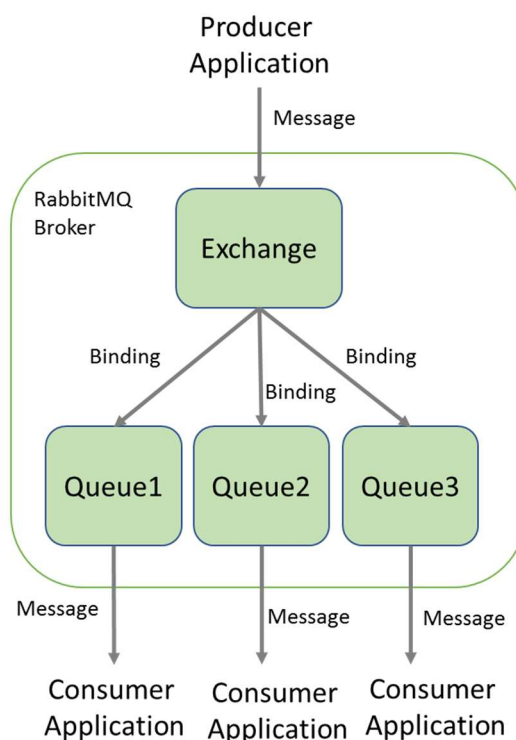


Figure 4. Standard RabbitMQ message flow

Figure 4 shows the standard RabbitMQ message flow. A producer application publishes a message to a given (selected) Exchange. When the Exchange receives the message, it is responsible for routing the message to an appropriate Queue(s). A Binding has to be set up

<sup>6</sup> <https://www.rabbitmq.com/getstarted.html>

<sup>7</sup> <https://docs.openstack.org/mitaka/install-guide-ubuntu/environment-messaging.html>



between a Queue and a given Exchange. In our case, there are bindings to three different Queues from the given Exchange. The Exchange routes the message to the Queues according to the Bindings specified. The messages stay in a Queue until they are handled by a consumer application.

A Binding is a "link" that is set up to bind a Queue to an Exchange. A routing key is a message attribute set up by the producer that allows an Exchange to look at this key and decide how to route the message to Queues depending on the Exchange type.

There are four different types of Exchange that route messages differently using different parameters and bindings setups. The most relevant to the THREAT-ARREST needs is the Exchange of type *Topic*.

### 2.1.2 RabbitMQ Topic Exchange

A Topic Exchange routes messages to Queues based on wildcard matches between the *routing key* specified in the message header and the *routing pattern* specified by the Queue *binding* (Johansson, 2015). Given the routing pattern of each Queue binding, messages are routed to one or many Queues.

The consumer indicates in which Topics is interested in, such as subscribing to a feed of a specific THREAT-ARREST platform tool. The consumer creates a Queue and sets up a binding with a given routing pattern to the selected Exchange. All messages with a routing key that match the routing pattern are routed to the Queue and stay there until the consumer consumes the message.

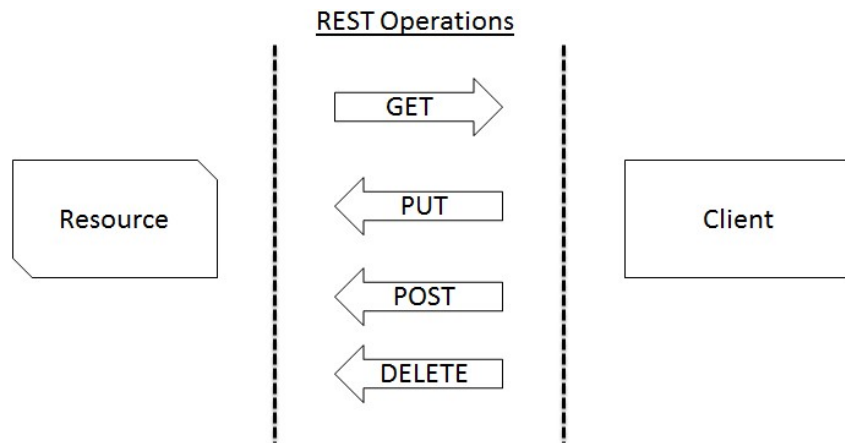
The routing key is a period (‘.’) delimited list of words, such as *EmulationTool.ehealthscenario1.vml* which identifies all events of cyber system emulation that are monitored at the Virtual Machine (VM) ‘*vml*’ of the eHealth scenario.

A routing pattern of a Queue binding can contain an asterisk ‘\*’ to indicate a match of words in a specific position of the routing key. For instance a routing pattern for a Queue1 can be *\*.ehealthscenario1.\** indicating all events from the cyber system of the eHealth scenario regardless of whether these are from simulation or emulation and regardless of what particular VMs they originated from.

A hash/pound symbol ‘#’ indicates match on zero or more words. For instance a routing pattern *EmulationTool.#* will match any routing keys beginning with *EmulationTool* resulting in capturing all events from cyber system emulation regardless of the specific scenarios currently used.

## 2.2 REST Communications

Nevertheless, except from the asynchronous communication through a broker, we also need synchronous communication options where the various modules can exchange data directly. Protocols that follow the REST architecture are adopted for this. RESTful implementations typically use the Hypertext Transfer Protocol (HTTP). In general, the REST solutions follow a request/response model, where a client may interact with the server using a subset of the HTTP methods, namely using GET, PUT, POST and DELETE on the server's resources (see Figure 5 below). RESTful systems target interoperability, performance and scalability for increasing resources consumption. They target reusability of components which can be managed or updated without affecting the system as a whole, even while it is running.



*Figure 5 The REST architecture and the supported operations*

For secure information transmission, the extension of the Hypertext Transfer Protocol Secure (HTTPS) is widely-used. The pure communication protocol (HTTP) is safeguarded by encrypting the data with the TLS protocol. However, HTTP/HTTPS are not appropriate for lightweight applications with resource, bandwidth, and/or energy restrictions.

Thus, the Internet Engineering Task Force (IETF) Constrained RESTful environments (CoRE) Working Group presented the Constrained Application Protocol (CoAP), now an IETF standard (Shelby et al., 2014). CoAP is a specialized web transfer protocol for use with constrained nodes and constrained networks in the IoT, aiming to maintain compatibility with the existing Internet infrastructure, through simple proxies. The protocol is often referred to as “the HTTP for the Internet of Things”. CoAP messages are transported over the User Datagram Protocol (UDP). Moreover, basic publish/subscribe interactions are also supported, as, by extending the HTTP GET method, a client can *observe* a specific resource. For security, CoAP applications support the Datagram Transport Layer Security (DTLS).



### 3 Emulation Tool Interconnections

There are three main types of communications that allow the Emulation Tool to interoperate with other platform components:

- REST API for offering emulation management, such as emulation initialisation and termination. The API is used by the Training Tool when trainees select new scenarios for training or terminate ongoing ones. These are *synchronous* communications indicating the result (successful or not) of actions influencing the emulation lifecycle. For instance, it is essential for the Training Tool to have a synchronous response if the emulation environment has been successfully initialised. If not successful, the training session may be terminated.
- HTTP communications for integration and operation of the Apache Guacamole<sup>8</sup> remote desktop gateway within the THREAT-ARREST Dashboard. These communications allow trainees to perform hands-on training with emulated cyber system components run on Windows or Linux virtual machines.
- Message-queue-broker-based communications with the other platform components on the operational state of the emulated cyber system. During a training session, state of the emulated cyber system changes according to the interactions of the trainee or the simulated system components with the emulated cyber system environment. Events and state of VMs hosting the cyber system components are monitored (by the Emulation Monitoring component) and communicated to other tools of the platform. These communications are *asynchronous* and subject to a large number of messages being exchanged. The Emulation Tool publishes all these events and state information to the message queue broker which dispatches all messages to the tools subscribed for that. Particularly, the Training and Visualisation Tools are receivers of such messages.

We note that this document does not address interconnection of components inside the emulated environment. We refer to the deliverables D2.1 and D2.3 for further information on how the emulated cyber system environment is interconnected.

RabbitMQ is selected as the Message Queue Broker for the THREAT-ARREST platform to enable all asynchronous communication needs among platform components.

---

<sup>8</sup> <https://guacamole.apache.org/>

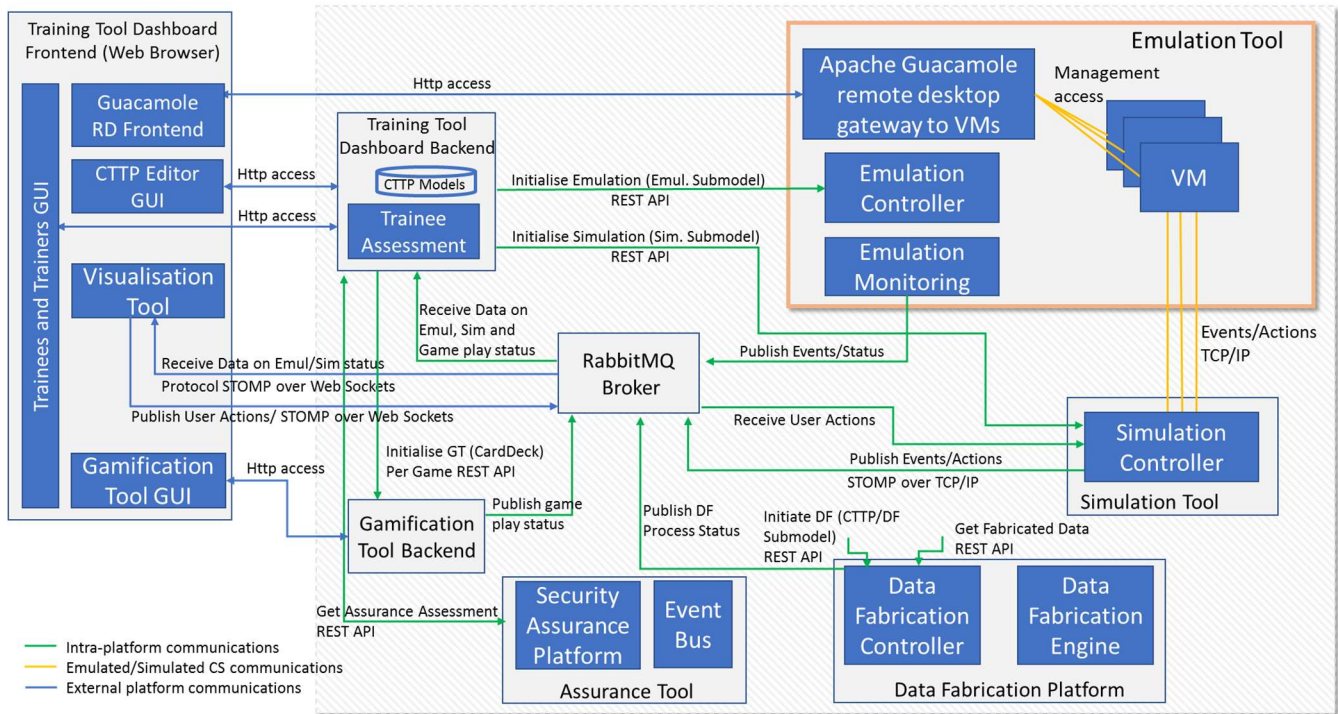


Figure 6 THREAT-ARREST Platform Components Communications – Emulation Tool View

Figure 6 shows the THREAT-ARREST platform components communications view, and particularly which components the Emulation Tool interconnects with. In the rest of the section, we will describe how the Emulation Tool interconnects with the relevant components of the platform. The Emulation Tool communications with the RabbitMQ broker are defined to use the default broker protocol – AMQP.

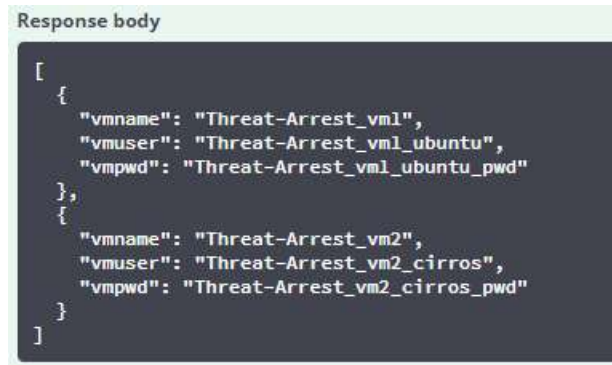
### 3.1 Initialisation of the Emulation Tool

The Emulation Controller provides a specific REST API for the initialisation of a specific training scenario. This API accepts as input the CTPP emulation sub-model and returns a list containing the VMs created for the current training scenario along with a corresponding username and password to access these (for more information regarding the CTPP sub-models refer to D3.1 while for the instantiation of emulated scenarios read D2.1 and D2.3). Trainees use the credentials in the Guacamole gateway to access the VMs.

The following code shows the initialisation API, whose behaviour has been described in Section 3 of the deliverable D2.1.

```
/emulation/getVMfromXML(String XML)
```

Then, Figure 7 shows the response returned by the API call indicating, for each VM of the scenario, username and password for the remote session.



```

Response body
[
  {
    "vmname": "Threat-Arrest_vm1",
    "vmuser": "Threat-Arrest_vm1_ubuntu",
    "vmpwd": "Threat-Arrest_vm1_ubuntu_pwd"
  },
  {
    "vmname": "Threat-Arrest_vm2",
    "vmuser": "Threat-Arrest_vm2_cirros",
    "vmpwd": "Threat-Arrest_vm2_cirros_pwd"
  }
]

```

Figure 7: API emulation/getVMfromXML Response Values.

In the second development iteration, it is planned to include the *session\_id* as an additional initialization parameter, in order to manage multiple parallel deploys of the same training scenario for a classroom of trainees.

### 3.2 Interconnection with the Simulation Tool

For a given training scenario and session, the Dashboard first initialises emulation and then simulation. The instantiation scripts for this and the overall process are detailed in D5.2. Upon emulation initialisation, the Emulation Controller also initialises the VM of the simulation tool in the *same context* (i.e. subnet or network) of the VMs of the emulation environment. We note that the Emulation Controller does not initialise the cyber system simulation for a given scenario but only the VM with the corresponding networking, computational, and storage resources. Such initialisation is necessary to ensure an environment of the simulation tool properly interconnected with the emulated cyber system (i.e. interconnected with the VMs of the emulated cyber system environment) and the external network.

TCP/IP enabled communications are identified to interconnect the simulated part of the cyber system with the emulated counterpart. There are no specific API/services necessary for the Simulation Controller to interconnect with the VMs of emulated environment. All the configurations needed to instantiate the VM dedicated to the simulation are already included in the CTTP models.

It is important to note that in case of a training scenario with cyber system simulation only, the Emulation Controller still needs to be invoked to initialise the VM of the simulation engine but without any emulation environment initialisation and interaction.

### 3.3 Interconnection with the Training and Visualisation Tools

The Emulation Tool is interconnected with the Training and Visualisation Tools through the RabbitMQ message broker. These are asynchronous communications on the state of the emulated cyber system environment. All related events and state information from the monitored environment, and in particular from the VMs of that environment, are communicated to both the Training Tool and the Visualisation Tool. These events and state information are used for different purposes – in the Training Tool for user performance assessment and in the Visualisation Tool for Dashboard visualisation of the cyber system environment. We recall that the Visualisation Tool is a JavaScript library integrated in the Dashboard front-end, refer to D5.2. Given the different purposes, each tool needs to filter and process only those events relevant for its scope. For instance, the trainee performance assessment may only need state information of specific VMs hosting specific components of the targeted cyber system, such as a mail server hosted in one VM or an IoT gateway hosted in another VM. While for the visualisation of the emulated cyber system components, it is necessary to consider all messages of all VMs in a given training session instance.

To address the different levels of events/state information needed by the Emulation Tool, it was decided to use RabbitMQ Exchanges of type Topic. In this case, *one* Exchange of type Topic is created for the Emulation Tool to serve *all* communications of state information for *all* training scenarios and training sessions. This “Topic” Exchange is created during the set up phase of the platform and its initial configuration. Provisionally, the Exchange is named *CS\_Emulation\_State*. It may change to reflect a more hierarchical or namespace format in the second year of the project.

Importantly for the Emulation Tool in this context is that all messages sent to the *CS\_Emulation\_State* are to bear a *well formed* routing key. It was agreed to use the following structure of the routing key:

```
EmulationTool.<ScenarioID>.<TrainingSessionID>.<VMID>[.<CyberSystemComponentID>]*
```

The `EmulationTool` is a constant used to indicate the name of the THREAT-ARREST platform component source of the message. The `<ScenarioID>` refers to the scenario identifier from the CTP model. The `<TrainingSessionID>` refers to the identifier of the training session and is managed by the Training Tool/Dashboard. Upon Emulation Tool initialisation, the actual training session ID is given along the CTP emulation sub-model. We note that for a given scenario there could be several training sessions for different users (trainees) of the same or different roles. The `<VMID>` refers to the identifier of the VM the message (event/state info) relates to.

The first four elements of the routing key of each message are *mandatory* and essential to determine the *namespace* of the message.

The `<CyberSystemComponentID>` refers to the identifier of a component of the cyber system that is hosted in the given VM, such as a mail server, application server, IoT gateway, network switch, etc. This element is *optional* and is subject to a decision by the Emulation Tool owner whether such granularity is suitable for a given scenario. The brackets with an upper index asterisk “`[ . . ] *`” indicate the expression “`<CyberSystemComponentID>`” can be repeated zero or more times depending on the complexity of the cyber system emulated. It is important to note that identifier information for the `<CyberSystemComponentID>` is obtained from the CTP model. In the next version of the deliverable a formal Backus-Naur Form<sup>9</sup> (BNF) specification will be provided.

As we said, there is one Exchange (*CS\_Emulation\_State*) pre-defined for the Emulation Tool that enables all messages sent by the Emulation Tool outreach all relevant components of the platform, including the Training and Visualisation Tools.

To do so, it was agreed that upon *initialisation* of the Emulation and Training Tools for a given scenario and training session, each tool *dynamically* creates Queues bound to the *CS\_Emulation\_State*. It is essential that the binding for each Queue follows the structure of the routing key discussed above. Asterisk “`*`” and pound “`#`” can be used as placeholders for a single word, or zero or more words, respectively.

For instance, the binding (matching pattern) for a Queue of the Visualization Tool, would be the following one:

<sup>9</sup> [https://en.wikipedia.org/wiki/Backus-Naur\\_form](https://en.wikipedia.org/wiki/Backus-Naur_form)

```
EmulationTool.<ScenarioID>.<TrainingSessionID>.#
```

The `<ScenarioID>` refers to the identifier of the scenario in the CTTTP model. The `<TrainingSessionID>` refers to the identifier of the current training session the Visualisation Tool is initialised for. The binding above means that the Queue will receive all messages sent by the Emulation Tool (to `CS_Emulation_State`) for a given scenario and session ID regardless of what VMs they are related to. In this case, the Visualisation Tool will receive all messages and parse those on the application level to visualise their value in the Dashboard.

Regarding the Training Tool's user performance assessment for a given scenario, the Training Tool may dynamically create separate Queues for each VM of relevance to the assessment. In this case the binding would be the following:

```
EmulationTool.<ScenarioID>.<TrainingSessionID>.<VMID>.#
```

We note that the information of VMID is made available either through the CTTTP model or through configuration files dynamically created by the Emulation Tool and shared with other platform components for a given scenario ID. Ongoing discussions are taking place on whether to use a private GitHub repository for the platform needs that will offer shared space for each scenario ID accessible by all platform components. Such decisions will be reported in the second year of the project.

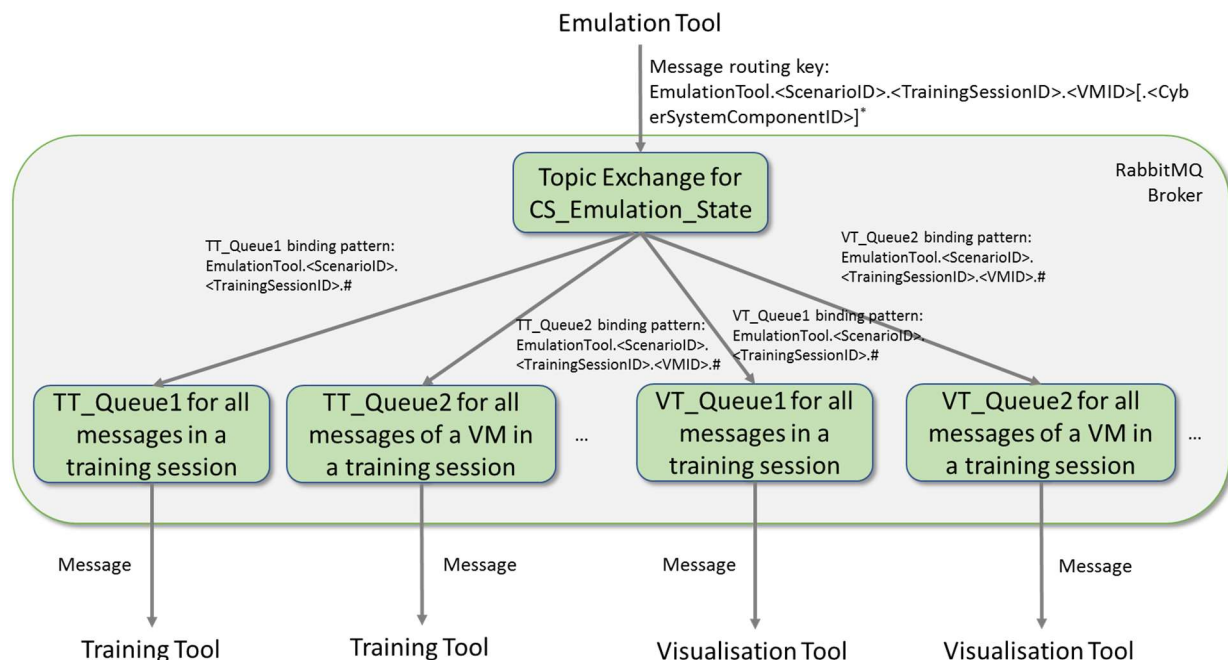


Figure 8 Emulation Tool Interconnection with the Training and Visualisation Tools

Figure 8 illustrates the interconnection of the Emulation Tool with the Training and Visualisation Tools.



```

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;

public class EmitCSEmulationState {

    private static final String EXCHANGE_NAME = "CS_Emulation_State";

    public static void main(String[] argv) throws Exception {

        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("localhost");
        try (Connection connection = factory.newConnection();
            Channel channel = connection.createChannel()) {

            channel.exchangeDeclare(EXCHANGE_NAME, "topic");

            String routingKey = "EmulationTool.Energy_BruteForceSSH.TrSess_98374767.VM_01.IoT-
Gateway";
            String message = "Log message regarding state of IoT Gateway";

            channel.basicPublish(EXCHANGE_NAME, routingKey, null, message.getBytes("UTF-8"));

        }
    }
}

```

*Code Example 1: RabbitMQ Java API for Topic Exchange Creation and Message Publishing*

Code Example 1 shows how the Emulation Tool can use the Java library/API provided by RabbitMQ<sup>10</sup> to create an Exchange of type Topic and publish a message with a routing key following the format presented above.

In the example, the following routing key is used:

```
EmulationTool.Energy_BruteForceSSH.TrSess_98374767.VM_01.IoTGateway
```

Give the example, a possible queue binding key for all messages of a training session of cyber system emulation would be:

```
EmulationTool.Energy_BruteForceSSH.TrSess_98374767.#
```

A possible queue binding key example for all messages regarding a specific VM of a training session of cyber system emulation would be:

```
EmulationTool.Energy_BruteForceSSH.TrSess_98374767.VM_01.#
```

A queue binding key example for all messages regarding a specific component of a VM of a training session of cyber system emulation would be:

```
EmulationTool.Energy_BruteForceSSH.TrSess_98374767.VM_01.IoTGateway
```

We refer to deliverables D4.3 or D5.3 for examples of how to use the RabbitMQ Java API for dynamic queue declaration with binding keys and message receipt on a queue.

### 3.4 Interconnection with the Data Fabrication Platform

The Data Fabrication Platform (DFP) offers REST APIs to other tools in the THREAT-ARREST platform to allow these to initialise (request) data fabrication and obtain generated data, respectively. We refer to the deliverable D5.3 for more details of the APIs of the DFP.

<sup>10</sup> <https://www.rabbitmq.com/tutorials/tutorial-five-java.html>

The Emulation Tool interconnects with the DFP through REST APIs when it is necessary to dynamically request synthetic data fabrication (e.g. security event logs) and obtain the generated data. Currently, it is being discussed to interface data fabrication results of each scenario with all other platform components, including the Emulation Tool, through a Git repository accessible by all platform components. In this case, the RabbitMQ broker will be used to notify all relevant platform components when the data fabrication process is completed, and when data is available in the Git (under a specific URI) so that the other components can access it. Alternatively, the DFP can use the broker to notify other tools when data is ready along with a specific identifier so that the tools can use a dedicated REST API to obtain the data.

Similar to the above described message broker means, a dedicated Exchange will be created for the DFP so that all other components including the Emulation Tool dynamically create a Queue bound to this Exchange upon initialisation and be informed when requested data is fabricated.

## 4 Conclusions and Next Steps

This first deliverable of task T2.4 presents the means used by the Emulation Tool to interconnect with the other platform components. The REST API and message-broker-enabled communications are identified serving different types of communications, such as synchronous vs asynchronous. The goal of this first version is to guide the Emulation Tool integration activities in the second year of the project, such as those in the context of WP6. We recall that this document has two other counterparts that complement the overall view of components' interconnections of the THREAT-ARREST platform – the deliverables D4.3 and D5.3.

With the adoption of REST and the RabbitMQ message broker, we addressed interoperability on the API level but also interoperability on the protocol level (e.g. (Soultatos et al., 2019)). For instance, the RabbitMQ community provides a number of libraries (APIs) for different programming languages that greatly facilitate message exchanges across different programming platforms (see Code Example 1 for the Java API use). RabbitMQ is also well-known for its multi-protocol interoperability support (Cameron, 2012). For instance, thanks to RabbitMQ, the Visualisation Tool (a JavaScript library running in a trainee's browser) can interconnect and exchange messages with other components, such as the Emulation Tool running a different message protocol. The Visualisation Tool adopted STOMP over WebSockets for browser-based message exchanges, while the Emulation Tool uses AMQP. Both tools can seamlessly exchange messages even if they are using different protocols.

Next steps in the second year of the project will target to address:

- Technical description and specification of interfaces (APIs) for the Emulation Tool communications with other platform components both through REST APIs and those through the RabbitMQ broker. We note that the technical specification of such APIs is subject to the design and technical development of individual components' functionalities.
- Interoperability on the message level to ensure that the syntax and semantics of messages (e.g. various log data from VMs) sent by the Emulation Tool are processable by the Visualisation and Training Tools as well.

The steps above will be particularly driven by the activities of WP6 regarding the platform integration and interconnection, which officially start in Month 13 of the project.



## 5 References

- [1] Banks, A. and Gupta, R. (2014) OASIS Message Queuing Telemetry Transport (MQTT), version 3.1.1, OASIS, pp. 1-81, <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.pdf>.
- [2] Cameron, B. (2012) The Polyglot Rabbit: Examples of Multi-Protocol Queues in RabbitMQ. Available at <http://assortedrambles.blogspot.com/2012/11/the-polyglot-rabbit.html>
- [3] Fielding, Roy Thomas (2000). "Chapter 5: Representational State Transfer (REST)". *Architectural Styles and the Design of Network-based Software Architectures* (Ph.D.). University of California, Irvine. [http://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)
- [4] Hatzivasilis, G., et al. (2019). Secure Semantic Interoperability for IoT Applications with Linked Data. IEEE Global Communications Conference (GLOBECOM 2019), IEEE, Waikoloa, HI, USA, 9-13 December 2019, pp. 1-7.
- [5] Hatzivasilis, G., Fysarakis, K., Soultatos, O., Askoxylakis, I., Papaefstathiou, I. and Demetriou G. (2018a) The Industrial Internet of Things as an enabler for a Circular Economy Hy-LP: A novel IIoT Protocol, evaluated on a Wind Park's SDN/NFV-enabled 5G Industrial Network, Computer Communications – Special Issue on Energy-aware Design for Sustainable 5G Networks, Elsevier, vol. 119, pp. 127-137.
- [6] Hatzivasilis, G., et al., (2018b). The Interoperability of Things. 23<sup>rd</sup> IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD 2018), IEEE, Barcelona, Spain, 17-19 September 2018, pp. 1-7.
- [7] ISO/IEC 20922 (2016). "Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1," June 15, 2016, <https://www.iso.org/standard/69466.html>.
- [8] Johansson, L. (2015) RabbitMQ Exchanges, routing keys and bindings. CloudAMQP Blog. Available at <https://www.cloudamqp.com/blog/2015-09-03-part4-rabbitmq-for-beginners-exchanges-routing-keys-bindings.html>.
- [9] Lakka, E., et al. (2019). End-to-End Semantic Interoperability Mechanisms for IoT. 24<sup>th</sup> IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD 2019), IEEE, Limassol, Cyprus, 11-13 September 2019, pp. 1-6.
- [10] Lonescu, V. M. (2015). The analysis of the performance of RabbitMQ and ActiveMQ, 14<sup>th</sup> RoEduNet International Conference – Networking in Education and Research (RoEduNet NER), IEEE, Caiova, Romania, Sept. 24-26, pp. 132-137.
- [11] Luzuriaga, J. E., Perez, M., Boronat, P., Cano, J. C., Calafate, C. and Manzoni, P. (2015). A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks, 12<sup>th</sup> Annual IEEE Consumer Communications and Networking Conference (CCNC), IEEE, pp. 1-6.
- [12] Richardson, A. (2014) RabbitMQ Essentials, PACKT Publishing, pp. 1-182. <http://www.spooch.dk/Ebooks/Programming/RabbitMQ%20Essentials%20%5BeBook%5D.pdf>
- [13] Shelby, Z., Hartke, K. and Bormann, C. (2014). The constrained application protocol (CoAP), IETF, RFC 7252. <https://tools.ietf.org/html/rfc7252>.
- [14] Soultatos, O., et al., 2019. Pattern-Driven Security, Privacy, Dependability and Interoperability Management of IoT Environments. 24<sup>th</sup> IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD 2019), IEEE, Limassol, Cyprus, 11-13 September 2019, pp. 1-6.
- [15] THREAT-ARREST D1.3. (2019). THREAT-ARREST platform's initial reference architecture. THREAT-ARREST Project. Available at <https://www.threat-arrest.eu/>
- [16] THREAT-ARREST D4.3 (2019). Training and Visualisation tools IO mechanisms v1. THREAT-ARREST Project. Available at <https://www.threat-arrest.eu/>

- [17] THREAT-ARREST D5.3 (2019). The Simulation component IO module v1. THREAT-ARREST Project. Available at <https://www.threat-arrest.eu/>