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

D5.4: Simulated components network execution module v1†

Abstract: This document is the result of the first iteration of task "T5.3 – Simulated components network execution" activities. This task develops the simulator execution module, that will carry out the execution of the simulated Components' network as derived from the task "T5.1 – Simulation Environment". This mainly includes the caption of events within the simulated components as the training process evolves and the notification of the Training and Visualization Tools.

Contractual Date of Delivery	30/11/2019
Actual Date of Delivery	30/11/2019
Deliverable Security Class	Public
Editor	*Torsten Hildebrandt, Dirk Wortmann (SIMPLAN)*
Contributors	*George Hatzivasilis (FORTH), Michael Vinov (IBM), Marinos Tsantekidis (TUBS)*
Quality Assurance	*Hristo Koshutanski (ATOS), Martin Kunc (CZNIC)*

† *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 Universität 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. *Hristo Koshutanski (ATOS),*
2. *Martin Kunc (CZNIC)*

Revisions

Version	Date	By	Overview
0.5, 0.6	20/11/2019	Editor	Changes as suggested by internal review; moved section 2 to 3.3
0.4	07/11/2019	Editor	Added contents of Section 3
0.3	01/11/2019	Michael Vinov (IBM)	Added Section 2
0.2	01/11/2019	George Hatzivasilis (FORTH)	Added Section 4
0.1	20/09/2019	Editor	First Draft with ToC

Executive Summary

This deliverable is the initial outcome of the task "T5.3 – Simulated components network execution". Here, we develop the simulator execution module as a part of the Simulation Tool of the THEAT-ARREST platform. This task is complementary to the task "T5.1 – Simulated components' network generator" (see the deliverable "D5.2 – Simulated components and network generator v1"). While T5.1 focuses in the specification and establishment of the simulation model required for the Cyber Threat and Training Preparation (CTTP) model-driven training, T5.3 is related to the dynamic aspects of executing the simulation as part of an ongoing training session. The documented module captures the various events within each simulated component and notifies accordingly other involved simulated/emulated components as well as the Training and Visualization tools. Task T5.3 started in month 4, therefore this deliverable documents the work done from months 4 to 15. At the end of the task's activities in month 30, this deliverable will be updated and extended in "D5.7 – Simulated components network execution module v2".

Table of Contents

1 INTRODUCTION 8

2 EVENTS MANAGEMENT IN THE SIMULATION TOOL 9

2.1 DEFINING THE BEHAVIOUR OF SIMULATED COMPONENTS 9

2.2 SIMULATION MODELLING/EXECUTION BY EXAMPLE 11

2.3 EXECUTING REAL AND SYNTHETIC EVENT LOGS 14

3 CONNECTION TO CTTP 17

3.1 INSTANTIATING A SIMULATED NETWORK 17

3.2 CAPTURING AND REPORTING THE SIMULATION EVENTS 19

3.3 TRAINEE'S EVALUATION 19

3.4 SUMMARY OF THE STEPS FOR THE MODEL-DRIVEN TRAINING PROCESS 20

4 CONCLUSION 22

REFERENCES 23

List of Abbreviations

API Application Programming Interface

CSP Constraint Satisfaction Problem

CTTP Cyber Threat and Training Preparation

DFP Data Fabrication Platform

GUI Graphical User Interface

GPS Global Positioning System

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IoT Internet of Things

JVT Jasima Visualization Tool

REST Representational State Transfer

STOMP Simple Text Oriented Messaging Protocol

TCP Transmission Control Protocol

TLS Transport Layer Security

UDP User Datagram Protocol

UML Unified Modelling Language

VM Virtual Machine

WP Work Package

XML Extensible Markup Language

List of Figures

Figure 1 Simulation Tool architecture and external communications (see D1.3)..... 9
Figure 2 General architecture of the Jasima software library for discrete-event simulation (see D5.2)..... 10
Figure 3 Simulated Sensor Component (1/4)..... 11
Figure 4 Simulated Sensor Component (2/4)..... 11
Figure 5 Simulated Sensor Component (3/4)..... 12
Figure 6 Simulated Sensor Component (4/4)..... 12
Figure 7 Example use of the component in a training scenario 13
Figure 8 Example Visualization View of the JVT Showing the State of Simulated Sensors .. 14
Figure 9 The UML class diagram of the simulated on-deck equipment (Source D5.2) 17

1 Introduction

This document is the result of the first iteration of the task "T5.3 – Simulated components network execution" activities. This task develops the simulator execution module as a part of the Simulation Tool of the THEAT-ARREST platform. Simulated cyber system components developed using the discrete event simulation engine *Jasima*[1] will be used within THREAT-ARREST where it is not possible or useful to fully emulate those components within the Emulation Tool (developed as part of the work package "WP2 – Emulation Tool", see the deliverables "D2.1 – Emulated components' generator module v1" and "D2.3 – Interlinking of emulated components module v1"). Examples of this include simulating human actors in a training session (like automated attackers trying to exploit a security problem), cases where special hardware is required (like a GPS sensor), or where a more abstract simulated representation of cyber system components is more appropriate than a very detailed representation using emulated software environments (e.g., a denial of service attack with a large number of real data packets sent in an emulated network versus a more abstract representation of a flow rate attribute between simulated network nodes being increased while an attack is ongoing).

Task T5.3 is responsible for the execution of a network of simulated components integrated with emulated components within the Emulation Tool and the Data Fabrication Platform (DFP) developed within the task "T5.2 – Statistical profiling of real event logs and generation of synthetic events logs" of "WP5 – Simulation environment". Work on this task is closely linked to task "T5.1 – Simulated components' network generator" (see "D5.2 – Simulated components and network generator v1"). While T5.1 is more concerned with the structural side of specifying and creating the simulation model required for a Cyber Threat and Training Preparation (CTTP) model-based training, T5.3 is related to the dynamic aspects of executing the simulation as part of an ongoing training session.

This task continuously collaborates with the Training and Visualisation components developed in work package "WP4 – Training and Visualization tools". A simulation run will be started by the Training Tool, then the Jasima Visualization Tool (JVT) developed as part of "T4.1 – Visualization tools" (see "D4.1 – THREAT-ARREST visualization tools v1") will be used to show the state of the cyber system comprised of simulated and emulated components during a training session.

This document is structured as follows. Section 2 describes the way the Simulation Tool defines simulated component's behaviour, allowing them to exchange messages/trigger events in other simulated components or in emulated components from the Emulation Tool of the THREAT-ARREST platform. The connection of simulation and the DFP, used to generate synthetic security event logs, is also described. Section 3 shows the connection of this work to the execution of a CTTP model-driven training, documenting how a training program is executed by the various platform components based on this model. Finally, this document is concluded by Section 4 with a short summary and outlook towards the next tasks to be performed as part of T5.3.

[1] The Jasima simulator: https://www.simplan.de/en/software-2/jasima/

2 Events management in the Simulation Tool

2.1 Defining the behaviour of Simulated Components

The general architecture of the Simulation Tool is shown in Figure 1, depicting its main components and data flows/connections to other components of the THREAT-ARREST platform (see the deliverables "D1.3 – THREAT-ARREST platform's initial reference architecture" and "D5.3 – The Simulation component IO module v1" for more details). A central component in this architecture is the *Simulation Controller* managing the lifecycle of a simulation run (e.g. creating, starting, pausing, etc.) accessible via a Representational State Transfer (REST) Application Programming Interface (API). It is also responsible for interfacing the Simulation Tool with the platform's RabbitMQ message broker to communicate events from the simulation to other platform components such as the Visualization or the Training Tool and to be able to receive events required for the succession of a simulation run. This message-broker-based communication is indicated in Figure 1 by the interfaces named *Data Source Service*.

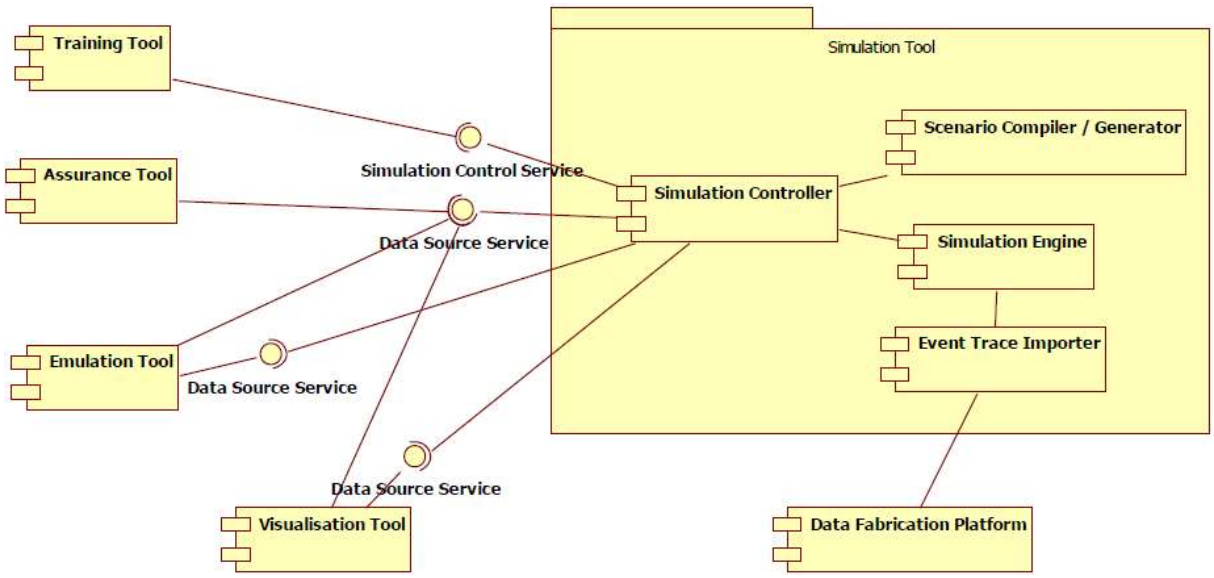

Figure 1 Simulation Tool architecture and external communications (see D1.3)

Simulated Components for the THREAT-ARREST platform use the Jasima library developed by SimPlan, as the Simulation Engine of the Simulation Tool. Jasima is a software library for discrete-event simulation written in the Java Programming language. The components to be used for a certain training scenario are defined within the CTTP Simulation Sub-Model (see the deliverable "D3.3 – Reference CTTP Models and Programmes Specifications v1") and finally submitted to the Simulation Controller as an Extensible Markup Language (XML) file. This file contains a hierarchy of simulation components with their specific parameter values to use. Internally, this maps to a tree of Java objects configured according to the needs of the scenario. This overall process is described more closely in the deliverable "D5.2 –Simulated Components and Network Generator v1".

All simulated components are therefore instances of a Java class. Therefore, they are Java objects that can have parameters (Java fields with getter/setter methods), can store simulation state in Java fields. Accordingly, the behaviour of components is defined in Java methods. Events sent between simulated components correspond to method calls, either by invoking them directly if they are supposed to occur at the same point in simulation time or indirectly via the *simulation kernel* that is responsible for executing events in the right temporal and logical order.

Restating information already contained in D5.2, the architecture of Jasima is shown in Figure 2, providing a more detailed view on the architecture of the *Simulation Engine* component shown in Figure 1. The core of the library consists of the discrete-event simulation kernel itself (responsible for managing and executing simulation events in the right order) and functionality required by most simulation models (like loading/saving simulation models, statistical analysis functions, functions to generate random numbers using a large variety of statistical distributions, etc.). Building on this, basic modelling elements like queues and abstract servers are provided as building blocks to create more complex domain-specific packages of simulation components. The simulation components developed within THREAT-ARREST can be seen as an example of such a domain-specific package for the application domain of cyber-security training. The architecture of Jasima is flexible and can be extended on all levels as required. Furthermore, it can easily be integrated in other software, to create solutions not necessarily intended to be used by simulation experts.

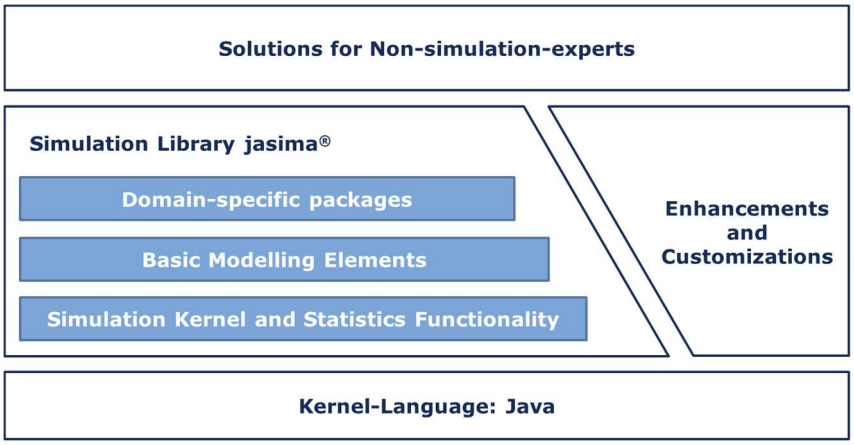

Figure 2 General architecture of the Jasima software library for discrete-event simulation (see D5.2)

All classes (i.e. simulated component types) used by the training scenarios of the three THREAT-ARREST pilots, are defined in a component library. This component library is maintained in a git repository accessible to all THREAT-ARREST platform tools. To define completely new training scenarios or modify existing ones, this repository can be cloned/branched by training developers. Existing components from this library can then be used as a kind of blueprint to create additional components or modify existing ones.

Within the THREAT-ARREST platform, simulated components will be used to model parts of the cyber-system that cannot easily be represented by emulated components. An example of such components would be to model human actors of a training session, e.g., to have simulated attackers trying to perform actions on emulated or simulated components in response to trainees' action.

Another use of simulated components is when replicating a real software environment in virtual machines (VMs) with the Emulation Tool would be very complex and the training scenario would not require the full complexity containing all details of a cyber system. In such a case, simulated components can provide an abstracted view on the system. For instance, train to detect and defend a distributed denial of service attack on a network, one could create a set of emulated components that actually send a large number of data packets in the emulated network. Instead, it might be sufficient to just define a set of simulated network nodes, specifying the communication links between them. The data rate exchanged between the nodes could then just be a set of state variables within the simulation. When an attack begins, these variables would just have to be increased, which could easily be shown in the Visualization Tool. Such a scenario would be much easier to set up and computationally far less demanding.

Whether this level of abstraction is sufficient for a certain training scenario depends very much on the target audience of a particular training session and the training goals of the scenario.

2.2 Simulation Modelling/Execution by Example

A simple example derived from the smart energy pilot is described in the following sections. It models a simple sensor as an example of a simulated component. The sensor has a current sensor value reading (a temperature reading). It furthermore has a state to indicate whether the sensor is working or in a failed state.

```
12 public class Sensor extends NetworkNode {
13
14     public enum SensorState implements SimpleState {
15         INITIAL, WORKING, BROKEN
16     }
17
18     // parameter
19
20     private SimulatedGateway gateway;
21     private double tempSlope, tempVariance;
22
23     // state variables used during simulation run
24
25     private ObservableValue<SimpleState, Sensor> state;
26     private ObservableValue<Double, Sensor> currentTemperature;
27
28     private DblStream tempChangeStream;
29     private IntStream newStateStream;
```

Figure 3 Simulated Sensor Component (1/4)

The component *Sensor* (see Figure 3) is derived from a base class *NetworkNode* (line 12). *NetworkNode* is a generic class from the component library defining common attributes of a network node. The sensor has a certain number of attributes (lines 20 and 21) and defines two variables to store the simulated inner state of a simulated sensor (lines 25 and 26). The fields defined in lines 28 and 29 use core functionality of the Simulation Engine to define random changes to the state variables. The two state variables *state* and *currentTemperature* are of type *ObservableValue*. This allows to store the current value and provide an efficient means to inform other components of changes to these values.

```
34
35     @Override
36     public void init() {
37         super.init();
38
39         // initialize state variables with some initial values
40         state = new ObservableValue<>(this, "state", SimpleState.class, SensorState.INITIAL);
41         currentTemperature = new ObservableValue<>(this, "currentTemperature", Double.class, 20.0);
42
43         // create random number streams used during simulation
44         tempChangeStream = initRndGen(new DblNormal(getTempSlope(), getTempVariance()), "tempIncrements");
45
46         // assign new states randomly using a fixed distribution
47         double[] probabilities = { 0.2, 0.5, 0.3 };
48         newStateStream = initRndGen(new IntEmpirical(probabilities), "newStates");
49     }
```

Figure 4 Simulated Sensor Component (2/4)

Before a simulation run starts, each component is initialized by calling it's *init()* method. For the sensor example this method is shown in Figure 4. In lines 40 and 41 it first initializes the observable state variables to some initial values (*state* to *INITIAL* and *currentTemperature* to *20.0*). Lines 44 and 48 then use functionality of the core Jasima classes to define streams of random numbers that are used to modify a sensor's state during the simulation run.

```
51    @Override
52    public void beforeRun() {
53        super.beforeRun();
54
55        // temperature can change every 5 minutes
56        schedulePeriodically(0.0, 5 * 60.0, Event.EVENT_PRIO_NORMAL, this::changeTemperature);
57
58        // state can change every 7.5 minutes
59        schedulePeriodically(0.0, 7.5 * 60.0, Event.EVENT_PRIO_NORMAL, this::changeState);
60    }
61
62    protected void changeTemperature() {
63        // new temperature is old value plus some increment
64        double newTemp = currentTemperature.get() + tempChangeStream.nextDbl();
65        currentTemperature.set(newTemp);
66    }
67
```

Figure 5 Simulated Sensor Component (3/4)

How a sensor component behaves during a simulation run is shown in Figure 5. It shows the use of the method *beforeRun* which is used to schedule any initial simulation events before the simulation starts. In the example we define two processes periodically updating the state variables. The field *currentTemperature* is updated for the first time at time *0.0* and then every *5* minutes (line 56). The variable *state* is also first updated at time *0.0* and subsequently every *7.5* minutes of simulation time (line 59). What happens during a temperature update is defined in the method *changeTemperature()* (lines 62-66 in Figure 5). The new value is calculated in line 64 as the old value plus some normally distributed random number. As a result, this creates a random walk with a linear trend to change the value of *currentTemperature*. Finally, the set-method invoked in line 65 will set the new value. As *currentTemperature* is an observable value, this will automatically trigger value update messages via the message broker so all components interested in this value will be notified.

```
67
68    protected void changeState() {
69        // new state is just determined randomly using some distribution
70        SensorState newState = SensorState.values()[newStateStream.nextInt()];
71        state.set(newState);
72    }
73
74    // boring getter/setter below
75
```

Figure 6 Simulated Sensor Component (4/4)

Similarly the method *changeState()* shown in Figure 6 is triggered every *7.5* minutes of simulated time to determine a new value of the *state* variable.

```
1  <jasima.core.simulation.SimulationExperiment>
2    <initialSeed>42</initialSeed>
3    <logLevel>TRACE</logLevel>
4    <simulationLength>4320.0</simulationLength>
5    <initialSimTime>0.0</initialSimTime>
6    <statsResetTime>0.0</statsResetTime>
7    <simTimeToMillisFactor>60000</simTimeToMillisFactor>
8    <simTimeStartInstant>2019-01-01T00:00:00Z</simTimeStartInstant>
9    <rootComponent class="threatarrest.simulation.common.SimulationScenario">
10     <name>scenario1</name>
11     <components>
12       <threatarrest.simulation.common.Network>
13         <name>network1</name>
14         <components>
15           <threatarrest.simulation.smarthome.Sensor>
16             <name>sensor1</name>
17             <ipAddress>192.168.32.1</ipAddress>
18             <initialTemperature>20.0</initialTemperature>
19             <tempSlope>0.0</tempSlope>
20             <tempVariance>0.5</tempVariance>
21           </threatarrest.simulation.smarthome.Sensor>
22           <threatarrest.simulation.smarthome.Sensor>
23             <name>sensor2</name>
24             <ipAddress>192.168.32.2</ipAddress>
25             <initialTemperature>20.0</initialTemperature>
26             <tempSlope>0.0</tempSlope>
```

Figure 7 Example use of the component in a training scenario

The sensor component defined in this way can later on be used in a training scenario. Instances of this component type can be created and parameterized based on the CTTP simulation sub-model. An example in Jasima's internal XML format is shown in Figure 7. It is defining some general simulation parameters, like the simulation start time and the simulation length in lines 2-8, before defining the hierarchy of simulation components starting in line 9. The simulation scenario *scenario1* defined there contains an example network *network1*, containing a number of sensor components. Each sensor is configured using a set of parameters, e.g., *tempSlope* and *tempVariance*.

The state variables *currentTemperature* and *state* can be observed by other THREAT-ARREST platform components. For instance, the Visualization Tool relies on this to be able to provide an up-to-date view on the state of the cyber system. In the example visualization view shown in Figure 8 (figure from "D4.1 – THREAT-ARREST visualization tools v1") the Visualization Tool subscribed to the values of 5 simulated sensors and always shows their most recent *currentTemperature* value graphically as bar/column charts.

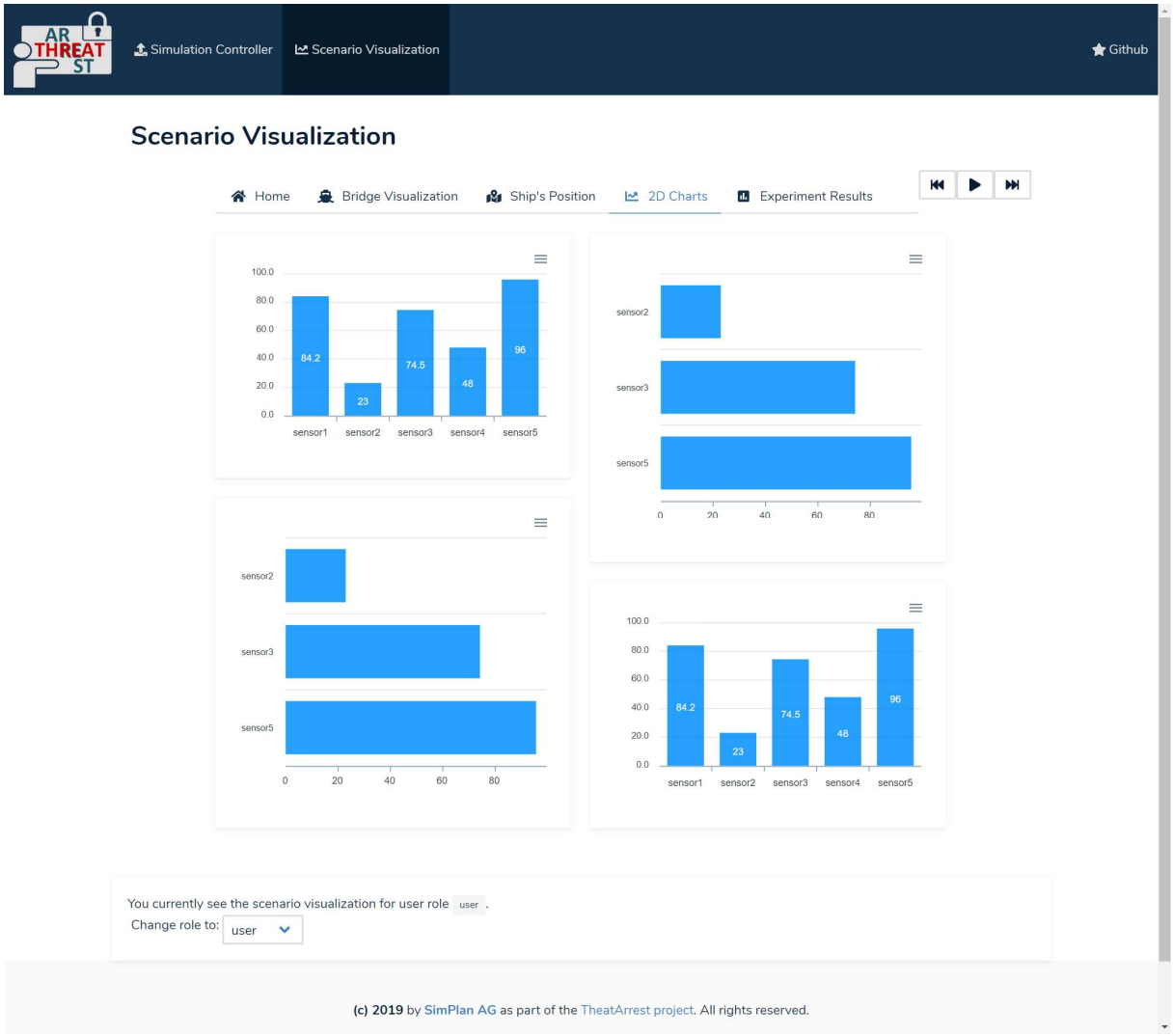

Figure 8 Example Visualization View of the JVT Showing the State of Simulated Sensors

2.3 Executing real and synthetic event logs

To support the THREAT-ARREST requirements, the IBM's Data Fabrication Platform (DFP) has been enhanced with the ability to generate sequences of simulated cyber-events in general, and synthetic security event log files in particular (see task T5.1 and D5.2). For such a case, the DFP needs to be properly initialized (based on definitions found in a CTTP model) before a user or a client sub-system can get any synthetic log file.

First, a virtual Computer Network topology should be defined. This includes declaration of network-attached computers, switches and other relevant hardware. Each hardware node should be augmented with properties and "installed" software applications and services. User-provided rules and constraints should complement the network definition to guide the fabrication engine, how to choose values for hardware and software properties.

Then, an Event Scenario, built of connected Actions and Activities, should be defined. In case a cyber-attack log is required, an attack Scenario should be defined over the virtual Network. After being properly configured with the Network topology and the Scenario definition, DFP creates a Constraint Satisfaction Problem (CSP) based on those definitions, solves the Problem with the CSP Solver, producing pseudo-random property and function call parameter values, satisfying all the definitions, rules and constraints. Finally, DFP simulates the Scenario, calling in application functions, declared by the Scenario actions, propagating events from one network

node to another, and stores the resulting event messages down to some persistent storage, producing event log files.

To connect a simulation with the DFP, the special simulation component *Event Trace Importer* (see Figure 1) will be used. The concept of the current prototype version of the component will be described here. When a first integrated version of the THREAT-ARREST platform is operational, this component will be thoroughly tested and refined.

As input to the *Event Trace Importer*, we expect data from three main sources:

1. Synthetic log files prefabricated for a training scenario by the DFP
2. Synthetic log files generated by the DFP dynamically during a training session
3. Real event logs from attacks to "replay" them.

From a simulation perspective, cases 1 and 3 would be very similar with the log files being available as static input files (e.g. from a git repository). Case 2 would use the REST API of the DFP to generate synthetic log files dynamically (see D5.3).

Event traces / logs are assumed to be textual data consisting of one line per event. Each event is characterised by a timestamp and a list of values defining the type of the event along with additional parameters characterizing it. An example of such a log file taken from Section 2.2. of D5.1 is shown below:

Client Computer log

<22>1 2019-06-20T10:12:43.341Z client.victimnet.net IBM Mail Client App 1.0 - Login [UserInfo Username="victim"] User logged in

<22>1 2019-06-20T16:25:18.197Z client.victimnet.net IBM Mail Client App 1.0 - Receive mail [Mail Sender="attacker@attackerdomain.com" Recipient="victim@victimdomain.com" Subject="This is a phishing mail!!!" Attachment="trojan_horse.exe"] Mail received

<14>1 2019-06-20T16:28:47.813Z client.victimnet.net IBM Client Computer File System 1.0 - File save [File Filename="trojan_horse.exe"]

The *Trace File Importer* now reads such a log file line by line. Currently this log file has to be accessible as a file on the computer running the simulation. In future versions also accessing the log via the REST API of the DFP will be supported.

Each line is parsed in order to split the various fields of information contained in it. Each line/event must have at least a timestamp and a type specifier. For each event contained in the log file an event is scheduled in the simulation. When the simulation time reaches this point in time, a handling method depending on the type of the event is invoked given the full details of the event as contained in the log file. This handling method is then responsible to trigger appropriate actions in a scenario-specific way. Such actions can include:

- trigger events/actions in other simulated components
- trigger events/actions in emulated components in the Emulation Tool
- directly send platform messages via the message broker to be consumed by, e.g., the Training or the Visualization Tool.

A simulation scenario can contain one or more *Event Trace Importers*. Each *Event Trace Importer* is expected to handle a single event log. Both the format of the log file and the actions to be triggered are assumed to be scenario-specific. Therefore, new classes to parse log entries

and classes to handle parsed events can be defined in the component library of the Simulation Tool, so that the *Event Trace Importer* can be configured appropriately.

3 Connection to CTTP

In this Section, we link our contributions so far and present how we monitor the trainee's actions and his/her interaction with the simulated components (e.g. (Alexandris et al., 2018; Hatzivasilis et al., 2019a; Hatzivasilis et al., 2019b; Hatzivasilis et al., 2017; Soultatos et al., 2019; Cesena et al.; 2017)). Firstly, we refer to the example from the deliverable D5.2 where we instantiate a simulated network based on the CTTP model (documented in "D3.2 – CTTP Models and Programmes Specification Tool"). Then, we describe the main architecture of the Simulation Tool (based on D1.3) as well as how it captures the trainee's actions and exchanges this information with the rest of the tools (based on D5.3). Finally, we combine all the above in order to present how the Training Tool receives the actual trace of the trainee's actions from the Simulation Tool at runtime and evaluates his/her performance on a specific training scenario (i.e. GPS spoofing). This is done via the *expected-trace*, which constitutes the set of the correct actions that must be performed in this specific scenario and is included in the related CTTP model.

3.1 Instantiating a simulated network

In the deliverable D5.2, we describe how to instantiate a network of simulated components for the smart shipping scenario based on the CTTP simulation sub-model. There, we determine the PAL/SAL and Hardware components of a smart vessel and how to instantiate the on-deck navigation equipment. In short, we have three different simulated navigation modules – compass, GPS, and live-maps – which capture the current position of the ship and present relevant measurements to the user at each simulated time point. The operational condition of the three modules is determined during the scenario initialization via the instantiation script (part of the CTTP simulation sub-model) and can be normal or faulty, while the GPS and live-maps could also be compromised by a malicious entity.

The next figure depicts the related UML class diagram for the navigation equipment and the main developed class attributes and methods.

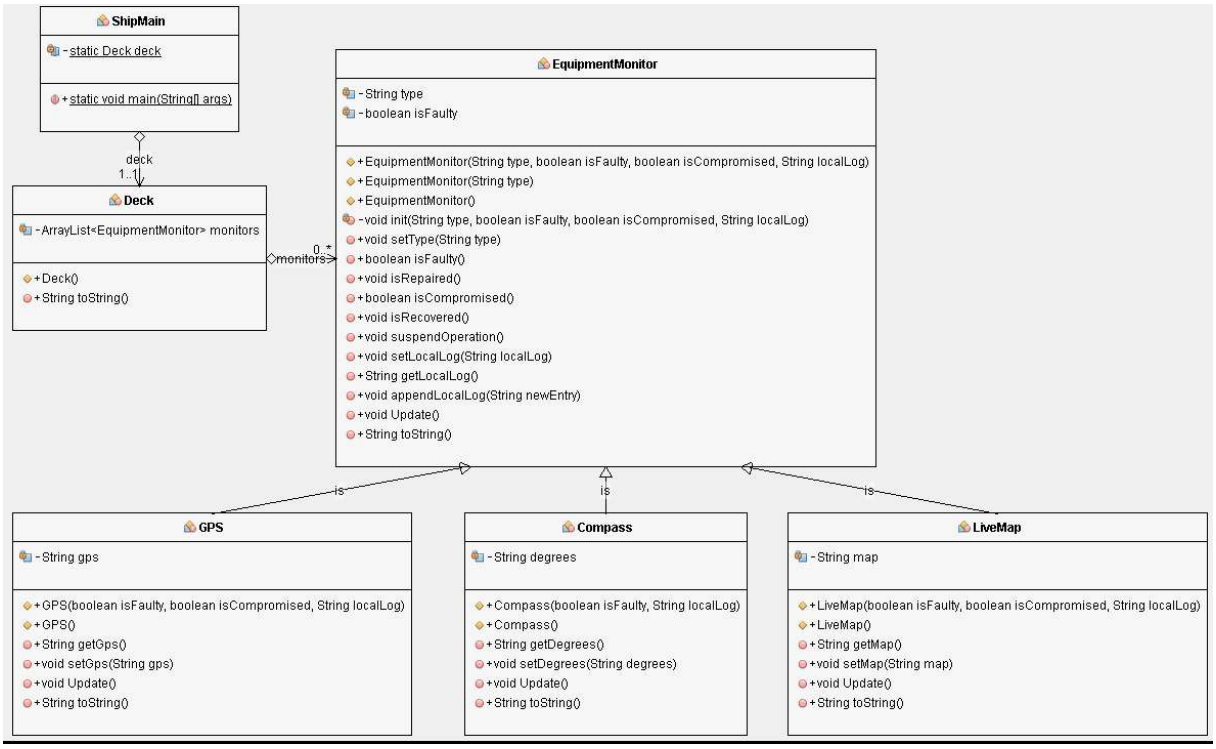

Figure 9 The UML class diagram of the simulated on-deck equipment (Source D5.2)

The following piece of code, taken from D5.2, describes an example instantiation script that we use in order to build the simulation of the on-deck equipment. The *component* and *subcomponent* elements reflect directly to the Java classes of a simulated component (see Section 2.2) and their attributes. We create a deck which is consisted by a *faulty GPS equipment*, along with a compass and the live-map modules that work properly. The *subcomponent* description determines the simulated component's constructor (in Java) that will be called. GPS is instantiated as 'faulty', while the default behaviour of the other two components, which is implemented by the default constructor (without input arguments), is the normal operation.

```
<Instantiation>
    <tool>Simulation</tool>
    <template_name>NO</template_name>
    <installation_script>
        <file_name>DCC-sim-scenario.xml</file_name>
        <duration>10<duration>
        <component>Deck</component>
          <subcomponent-list>
                    <subcomponent>
                              <name>GPS</name>
              <isFaulty>true</isFaulty>
              <isCompromised>false</isCompromised>
              <localLog>'the navigation log with fabricated events that are reported'</localLog >
                              </subcomponent>
            <subcomponent>
                              <name>Compass</name>
                              </subcomponent>
            <subcomponent>
                              <name>LiveMap</name>
                              </subcomponent>
          </subcomponent-list>
    </installation_script>
</Instantiation>
```

Via the model-driven approach that is promoted by the THREAT-ARREST platform, we can create CTTP models that capture all the different states of the three navigation monitors and their combinations (covering various learning goals and training difficulty levels). The CTTP models are established via the CTTP editor (D3.2) and are maintained locally in a platform repository.

Through the Dashboard, the trainee can select and activate the simulated scenario. The Training Tool retrieves the instantiation script (from one of the underlying CTTP models of this scenario) and configures the Simulation Tool accordingly (D5.2-D5.3). When the simulation is up and running, the trainee is notified and can interact with the simulated components via the modules of the Visualization Tool. For more details, refer to the deliverables D5.2 and D5.3.

The main actions that we model here, is the monitoring of the navigation measurements/values of the tree modules by the trainee and the choice to suspend the interaction of a module with the navigation system.

- By default, the ship is automatically navigated by the GPS.
- If the GPS is suspended, the ship is navigating via the information from the LiveMap.
- If both GPS and LiveMap are suspended, the ship is navigated with the use of the compass.

- If all modules are suspended, we consider that the emergency authorities are informed about and the captain navigates the ship manually based on what he/she sees from the deck.

Thus, the trainee has to correlate the measurements of the three monitors and figure out if some modules do not work properly and make the correct navigation choices.

3.2 Capturing and reporting the simulation events

The architecture of the Simulation Tool along with its main sub-components and their operation is presented in the deliverable D1.3 and was already recalled briefly in Section 2.1. For a more in-depth description of the main interconnections of the Simulation Tool with the rest of the platform's tools the reader is referred to D5.3.

As can be seen in Figure 1, the main interaction point is the *Simulation Controller*. It gets as input the information of the CTTP simulation sub-model, instantiates the simulation, monitors the operation of the simulated components, and reports ongoing events to the other tools.

The eventing mechanism is described in the deliverable D5.3. This asynchronous communication is enabled via the RabbitMQ message broker. The Simulation Tool publishes the upcoming events to its predefined Exchange, to which the related tools (i.e. Training and Visualization Tools) have been subscribed in order to get notified. It is agreed that all these messages bear a well formed routing key that complies the following structure:

```
SimulationTool.<ScenarioID>.<TrainingSessionID>[.<CyberSystemComponentID>]+.<CyberSystemComponentAttributeID>
```

Next, we extended the offered operation of the Jasima simulator (D5.2/D5.3) in order to capture the changes of the simulated components as time progresses and update the Exchange. This functionality is implemented by the *ReceiveSimVisUserActions* class. An example message that indicates the current GPS coordinates as they are reported by the GPS module would look like this:

```
{
  "simTime":1500,
  "simTimeAbs":"2019-10-28T20:00:39.638Z",
  "wallTime":1564432617032,
  "valueName":"scenario1.ShipMain.Deck.GPS.getGPS",
  "gps":35.341846,25.148254
}
```

The event is parsed by the Visualization Tool which presents this information to the trainee. For more details, refer to the deliverable D5.3 and D4.1, respectively.

3.3 Trainee's evaluation

In this deliverable (D5.4), we describe how we can monitor the trainee's actions on the simulated components and report the *actual interaction trace* back to the Training Tool, which starts the simulation and maintains the status of the overall training procedure. Except from the instantiating information, a full CTTP model (which is documented in the deliverable D3.2) determines which is the *expected-trace* that must be followed by the trainee in order to consider that he/she has performed the required actions for the specific simulated scenario (i.e. suspend the operation of the navigation tools that do not work properly or turn to manual operation if all three modules are faulty or compromised).

The following piece of code presents the *expected-trace* for the referred example where the GPS is faulty and must be suspended.

```
<expected-trace>
        <valueName>scenario1.ShipMain.Deck.GPS.suspendOperation</valueName>
</ expected-trace >
```

Thus, if the trainee presses the related button in the Visualization Tool, the method *'void suspendOperation()'* is performed by the underlying GPS object and the object *ReceiveSimVisUserActions* updates the simulation's Exchange with the following event:

```
{
  "simTime":1500,
  "simTimeAbs":"2019-10-28T20:00:39.638Z",
  "wallTime":1564432617032,
  "valueName":"scenario1.ShipMain.Deck.GPS.suspendOperation"
}
```

Then, the Training Tool correlates the upcoming event with the entry in the *expected-trace* and signifies that the correct action has been performed by this trainee (in this example, once a navigation module is suspended, it cannot be reactivated throughout the simulation).

In general, the *expected-trace* can be much more complex in order to represent various event sequences. The full supported functionality is detailed in the deliverable D3.2.

3.4 Summary of the steps for the model-driven training process

The overall steps of the CTTP model-driven process for this simulated scenario are summarized below:

- The instantiation script specifies (retrieved by the Training Tool and passed to the Simulation Tool):
    - the Java objects and their connections that will be instantiated and simulated (the on-ship deck with the three underlying monitors)
    - that status of the three monitors (compass, GPS, and live-maps) by setting the parameters of their constructors.
- The *expected-trace* specifies the set of the correct actions that the trainee must perform:
    - for each of the three modules that has been instantiated as faulty or compromised, there is one relevant entry in the trace
    - the trainee must deactivate each faulty/compromised equipment from the navigation service by pressing a button that performs the method *'void suspendOperation()'* in the relevant *EquipmentMonitor* object (superclass of the classes *GPS*, *Compass*, and *LiveMap*, see **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.**)
    - via the class *ReceiveSimVisUserActions*, the *Simulation Controller* captures all the monitored trainee's actions and updates the Exchange of this simulation instance
        - the Visualization Tool has been subscribed in this Exchange and presents the navigation values of the three modules to the trainee
        - the Training Tool has been subscribed in this Exchange and collects all the updates of the trainee's actual trace:

- if one of these actions is also contained in the *expected-trace* of the running CTTP model (i.e. the deactivation of a faulty/compromised component), the Training Tool signifies that the trainee has perform a correct action

- When the simulation training process is over, the Training Tool evaluates the trainee's performance and updates his/her record in the THREAT-ARREST platform.

4 Conclusion

This document presented the work done as part of the task T5.3 from month 4 until month 15. In Section 2, it described how the behaviour of simulated components is defined in Jasima – the discrete-event simulation library used as the core of the Simulation Tool of the THREAT-ARREST platform. Then, Section 3 first recalled the connection of the Simulation Tool to the CTTP-model and described in the main part the execution logic of a training session, integrating multiple components of the THREAT-ARREST platform to create a coherent view on the cyber system for trainees.

Next steps in T5.3 will be driven on one hand, by the integration of the Simulation Tool towards the first integrated version of the THREAT-ARREST platform due in month 20 (to be documented in "D6.1 – Initial Prototype of Integrated THREAT-ARREST platform"). Furthermore, for the three THREAT-ARREST pilots additional simulated components will be implemented. They will make use of the basic functionality as described here but use them to create more complex/realistic behaviour of simulated components as required by the pilot's training scenarios.

References

[1] Alexandris, G., et al., 2018. Blockchains as enablers for auditing cooperative circular economy networks. 23rd 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.

[2] Cesena, M., et al. 2017. SHIELD Technology Demonstrators. CRC Press, Book for Measurable and Composable Security, Privacy, and Dependability for Cyberphysical Systems, pp. 381-434.

[3] Hatzivasilis, G., et al., 2019a. The CE-IoT Framework for Green ICT Organizations. 1st International Workshop on Smart Circular Economy (SmaCE), Santorini Island, Greece, 30 May 2019, IEEE, pp. 1-7.

[4] Hatzivasilis, G., et al., 2019b. MobileTrust: Secure Knowledge Integration in VANETs. ACM Transactions on Cyber-Physical Systems – Special Issue on User-Centric Security and Safety for Cyber-Physical Systems, ACM, vol. 4, issue 3, Article no. 33, pp. 1-15.

[5] Hatzivasilis, G., et al., 2017. Real-time management of railway CPS. 5th EUROMICRO/IEEE Workshop on Embedded and Cyber-Physical Systems (ECYPS 2017), IEEE, Bar, Montenegro, 11-15 June 2017.

[6] Soultatos, O., et al., 2019. Pattern-Driven Security, Privacy, Dependability and Interoperability Management of IoT Environments. 24th 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