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

D5.2: Simulated Components and Network Generator v1†

Abstract: This deliverable provides a report on the design and current development status of the THREAT-ARREST simulation component, particularly on an initial set of simulated components implemented and on how to setup simulation scenarios of networks. It is a first demonstration of the work performed as part of task "T5.1 – Simulated Component's Network Generator" in the first 8 months after its beginning in month 4 of THREAT-ARREST.

Contractual Date of Delivery	31/08/2019
Actual Date of Delivery	31/08/2019
Deliverable Security Class	Public
Editor	*Torsten Hildebrandt (SimPlan)*
Contributors	*Torsten Hildebrandt, Dirk Wortmann (SimPlan),* *George Hatzivasilis (FORTH),* *Michael Vinov (IBM)*
Quality Assurance	*Vassilis Prevelak (TUBS),* *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 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. *Vassilis Prevelak (TUBS)*
2. *Martin Kunc (CZNIC)*

Revisions

Version	Date	By	Overview
0.4	23/08/2019	Editor	Incorporating comments from internal review
0.3	31/07/2019	Torsten Hildebrandt, SimPlan	Jasima/SimController sections added
0.2	23/05/2019	George Hatzivasilis, FORTH	Connection of CTTP models & simulation
0.1	12/05/2019	Editor	First Draft

Executive Summary

This deliverable documents the current state of task T5.1 of the THREAT-ARREST project "Simulated Component's Network Generator", being concerned with the development of the Simulation Module of the THREAT-ARREST platform using the discrete-event simulation engine Jasima. Simulation is an important part of a Cyber Threat and Training Preparation (CTTP) training session.

The work on task T5.1 started in month 4 of the project, so this deliverable reports on the results achieved between months 4 and 12. It presents a first version of the THREAT-ARREST Simulation Tool. It demonstrates the tool's use to create and execute a network of simulated components in a web-based setting:

- creating and parameterizing simulation runs,
- controlling the simulation lifecycle (start/stop/pause execution), and
- allowing for the simulation state to be shared/synchronized with the Jasima Visualization Tool (JVT) developed as part of task T4.1.

This deliverable focusses on the functionality required to create, parameterize, and control networks of simulated cyber-system components that will be part of a CTTP training program. This deliverable is complemented by a number of additional deliverables, detailing additional aspects of the simulation capabilities of the THREAT-ARREST platform. First, the deliverable D5.3 presents the integration of the Simulation Tool in the overall platform architecture in detail, also describing its interactions with other platform components. In month 15, the deliverable D5.4 (task T5.3) will present details on the execution of networks consisting of simulated components.

Work on task T5.1 is closely related to the development of the JVT, therefore deliverable D4.1 (in particular Section 5) complements this document by showing how the Simulation Tool can be integrated with the JVT and display information from a simulation run. Synchronization between JVT and the Simulation Tool is achieved using a message-oriented middleware that will also be used in the future THREAT-ARREST platform implementation as a means for asynchronous communication between various platform components.

Finally, this deliverable is related to deliverable D3.1, detailing the structure of a CTTP model. In the full platform implementation, the simulation sub-model of the CTTP model will define the set of simulated components and their parameters, defining their behaviour during a training session.

Table of Contents

List of Abbreviations

CTTP Cyber Threat and Training Preparation

DCC Deck Control Center

DFP Data Fabrication Platform

GPS Global Positioning System

JVT Jasima Visualization Tool

PAL Platform Level software

SAL Software Architecture Layer

SVG Scalable Vector Graphics

VM Virtual Machine

XML eXtensible Markup Language

XSD XML Schema Definition

WP Work Package

List of Tables

List of Figures

1 Introduction

This deliverable documents the current state of task T5.1 of the THREAT-ARREST project "Simulated Component's Network Generator", being concerned with the development of the Simulation Module of the THREAT-ARREST platform using the discrete-event simulation engine Jasima. Simulation is an important part of a Cyber Threat and Training Preparation (CTTP) training session. The work on task 5.1 started in month 4 of the project, so this deliverable reports on the results achieved between months 4 and 12. It presents a first version of the simulation tools, that can be used to create a network of simulated components in a web-based setting creating simulation runs, controlling the simulation lifecycle (start/stop/pause execution), and allowing for the simulation state to be shared/synchronized with the Jasima Visualization Tool (JVT) developed as part of the task T4.1.

This deliverable focusses on the functionality required to create, parameterize and control networks of simulated cyber-system components that will be part of a CTTP training program. This deliverable is complemented by a number of additional deliverables, detailing additional aspects of the simulation capabilities of the THREAT-ARREST platform. First, the deliverable D5.3 presents the integration of the Simulation Tool in the overall platform architecture, also describing its interactions with other platform components. Work on task T5.1 is closely related to the development of the JVT, therefore the deliverable D4.1 (in particular Section 5) complements this document by showing how the Simulation Tool can be integrated with the JVT and display information from a simulation run. Synchronization is achieved using a message-oriented middleware that will also be used in the future THREAT-ARREST platform implementation as a means for asynchronous communication between various platform components. Finally, this deliverable is related to the deliverable D3.1, detailing the structure of a CTTP model. In the full platform implementation, the simulation sub-model of the CTTP model will define the set of simulated components and their parameters, defining their behaviour during a training session.

This document consists of two main parts. In Section 2 the simulation environment, consisting of the simulation controller as well as a first set of simulated components and the simulated component's network generator will be developed.

Section 3 presents details of the integration of the Simulation Tool with the simulation sub-model of a CTTP model. This document has two appendices. Appendix I shows the eXtensible Markup Language (XML) Schema Definition (XSD) file of the CTTP simulation sub-model. Appendix II contains an example XML document following this schema, describing part of the example from the smart shipping pilot used in Section 3.

Finally, Section 4 concludes this work.

2 Generation and Instantiation of Simulated Network Components

2.1 The Jasima Discrete-Event Simulation Library

The THREAT-ARREST platform uses the Jasima discrete-event simulation library as the core of its simulation component. Jasima is a Java-based software library developed by SimPlan. In the THREAT-ARREST project it is extended significantly to cover the needs of model-driven cyber-security training. To achieve this goal, the current implementation work focusses on two things. On the one hand on developing a set of simulated components as required by the THREAT-ARREST pilot scenarios and on the other hand integrating Jasima into the overall platform architecture.

Work on simulation is closely related to SimPlan's work on Task T4.1 (Visualization Tools) developing the Jasima Visualization Tool (JVT, see also deliverable D4.1). As a first development step a working integration of the simulation component and JVT was developed and presented at the July 2019 Consortium Meeting in Chania. Material in this deliverable is based on first versions of both components successfully working together to visualize a running simulation session.

2.2 Simulation Tool Architecture

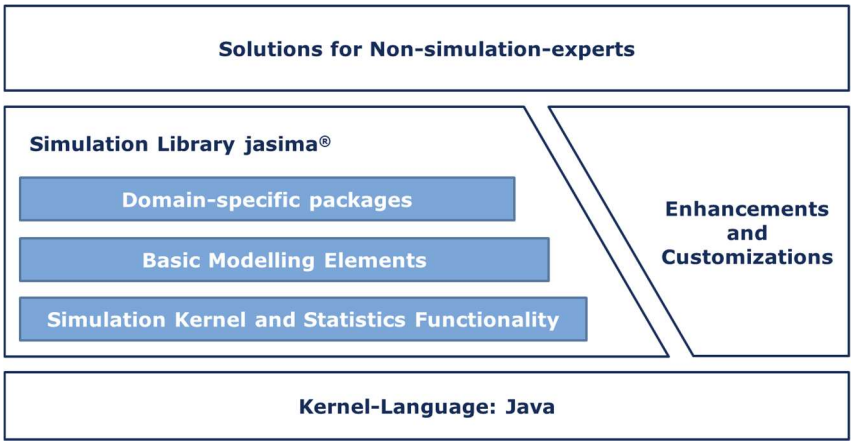

Figure 1 General architecture of the Jasima simulation library

The general architecture of the Jasima simulation library is shown in Figure 1. Jasima is a simulation originating from the production/logistics domain and developed as a flexible, extensible and fast discrete event simulation library that can be the core of decision support systems. Building on the Java programming language, it can benefit from a large ecosystem of tools and software libraries to quickly build simulation-based custom software.

It follows a three-layer approach with the kernel functionality and statistics functions as the foundational layer. Upon that there is a library of basic discrete-event modelling elements (like queues), forming the basis for domain-specific components and component libraries. The simulation components developed for the cyber-security domain within THREAT-ARREST can be seen as an example of such a domain-specific package.

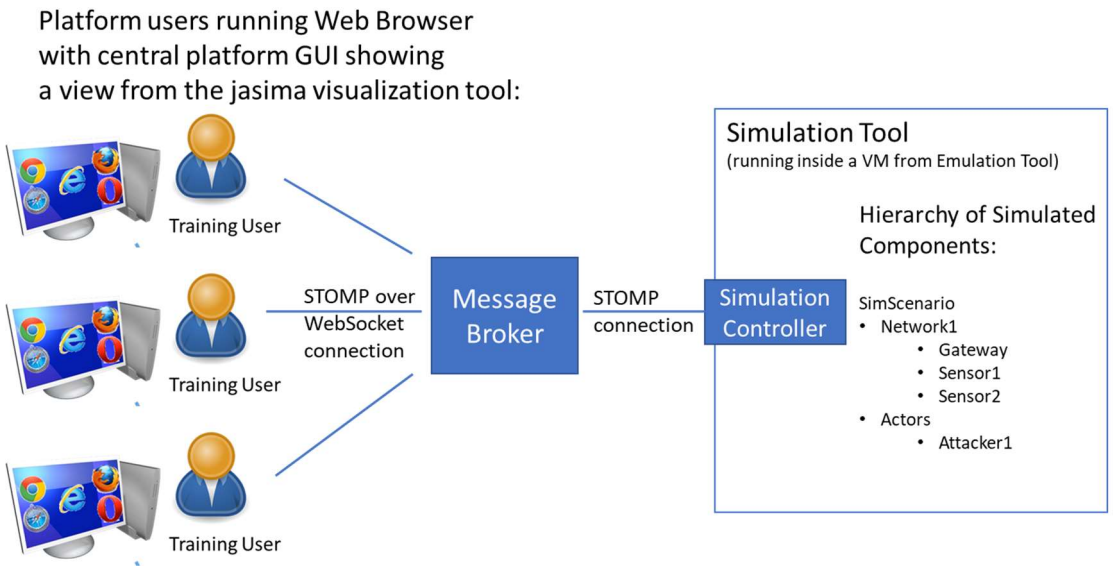

Figure 2 Simplified Platform Architecture from a Simulation Perspective

In order to integrate Jasima in a distributed, web-based architecture such as the THREAT-ARREST platform, a new component called "Simulation Controller" was developed. It offers a new execution environment for running Jasima simulations and computer experiments.

A simplified version of the THREAT-ARREST system architecture focusing only on the Simulation Tool and its integration with the Jasima Visualization Tool is shown in Figure 2. The Simulation Tool is executed on some computer. Within the Simulation Tool, the simulation controller is responsible for interfacing with external components, exposing the capabilities to create/start/pause/stop simulation runs, allow external components, such as Jasima Visualization Tool, but also the Training Tool, to access the values of certain state variables and get notified whenever there are changes of these values during a simulation run. Using a publish/subscribe-mechanism external components can subscribe to messages sent by the simulation. This is used primarily by the Jasima Visualization Tool to be notified whenever the state of a simulated component changes and the user interface has to be updated. This mechanism is also used by the Training Tool to receive feedback about a trainee's performance in a training session so the performance assessment can be updated in real-time.

In comparison to the final THREAT-ARREST system architecture, the current version 1.0 of the Simulation Tool uses a slightly simplified setup to develop a first working integration between the simulation component and the Jasima Visualization Tool. Instead of running within the Emulation Tool and using the platform's message broker, it is started directly on a computer accessible by the Jasima Visualization Tool. Additionally, the message broker is using a simple broker implementation running as part of the simulation controller. In contrast to the final platform setup, simulation execution is directly controlled by the user using controls of the Visualization Tool (see also D4.1). Using this setup, simulation state can be shared successfully and synchronized with connected instances of the JVT.

In the full THREAT-ARREST platform the Simulation Tool will be executed inside the emulation environment on a dedicated virtual machine. This way simulated components can directly communicate with emulated components running in their own Virtual Machines (VMs). In order to receive simulation tasks and connect to components outside the emulated environment, the simulation controller communicates with a broker component of the platform's message-oriented middleware. Using this mechanism, the Training Tool of the THREAT-ARREST platform will control simulation execution and orchestrate the overall

execution of a training session. For further details on the integration of the Simulation Tool in the overall THREAT-ARREST system architecture see also deliverable D5.3.

2.3 Simulation Controller API

In the current version of the simulation controller, simulation runs can be started via the (internal) message broker. To do this, the simulation controller is listening for incoming messages on a number of different queues and reacts upon them appropriately (see Table 1). In a future version this functionality will also be available via a REST interface.

Table 1 Actions implemented by the Simulation Controller

Action	Queue Name	Parameters / Description
create simulation instance	`/simulation-controller/createAndInit/{simId}`	Creates a new simulation instance accessible using the given simId. The simulation experiment's definition in XML format has to be given in the body of the message send to this queue.
run simulation	`/simulation-controller/run/{simId}`	Start executing the simulation identified by simId. No further parameters.
pause simulation	`/simulation-controller/pause/{simId}`	Pause executing the simulation after the current event is processed. Simulation is identified by simId and has to be running. No further parameters.
reset simulation	`/simulation-controller/reset/{simId}`	Stop executing the simulation identified by simId. The simulation is reset to its initial state. This method is usually directly followed by a "run" action
Start notifications on a certain state variable.	`/simulation-controller/observe-value/{simId}`	Takes as the messages' body the name of the value to be observed as a simple string. This method is usually executed by the JVT to signal interest in particular parts of the simulation state. As a result, update messages can be sent selectively by the simulation controller if a certain value is really needed.

Messages from the simulation controller in response to these messages will be published to the queues named "/topic/simulation/simcontroller-replies" for normal replies or to „/queue/errors" to indicate error conditions.

2.4 Sharing Simulation State with External Platform Components

The simulation shares information of its state (and in particular about state changes) by publishing update messages to a certain queue. Interested components such as the JVT (but also the Training Tool) can subscribe to such queues in order to be notified about changes and react upon them appropriately. This information is currently sent to a queue named /topic/simulation/values/{simId}/{valueName} where valueName and simId will identify which value changed and what simulation run it belongs to. An example message looks like this:

```
{
   "simTime":1440,
   "simTimeAbs":"2019-06-26T21:40:35.638Z",
   "wallTime":1564604321732,
   "valueName":"scenario1.mainNet.sensor1.currentTemperature",
   "oldValue":23.5,
   "newValue":26.9
}
```

The messages are in JSON format and contain information on the simulation time of the change (first two data fields), the real time on the simulation computer when the change occurred, which value was changed and both new and old values. If the current simId would be "sim123" then this message would have been published to the queue "/topic/simulation/values/sim123/ scenario1.mainNet.sensor1.currentTemperature". For the value name the first part uniquely identifies the component based on its name in the component hierarchy (scenario1.mainNet.sensor1). The last part identifies a certain attribute that is observable (currentTemperature).

The exact naming scheme might change slightly in the future to a unified naming schema across all platform components currently defined by project partner ATOS.

2.5 Initial Set of Simulation Components

Conceptionally, Jasima simulation models consist primarily of component containers and "normal" simulation components. Containers can contain simple components or other containers, forming a component hierarchy. Each component is uniquely identified by its name and can have an arbitrary set of additional attributes. Each (type of) simulation component in Jasima is a Java class defining its attributes and its behaviour. Normal components that do not have child components are usually derived from a class "SimComponent". Simulation components that can contain other components are derived from a class "SimComponentContainer".

Defining the set of possible simulation components usually takes place using a Java IDE and starts with a project template defining the necessary dependencies to use the core Jasima classes and offering a build script to export all required classes as a single jar file. In THREAT-ARREST there will be a generic project template containing a set of generic simulation

components useful across all training scenarios and specific templates derived from it containing components that are required only for certain THREAT-ARREST pilots and specific scenarios. Existing components in the project template can be used as a reference to create custom components to implement additional scenarios.

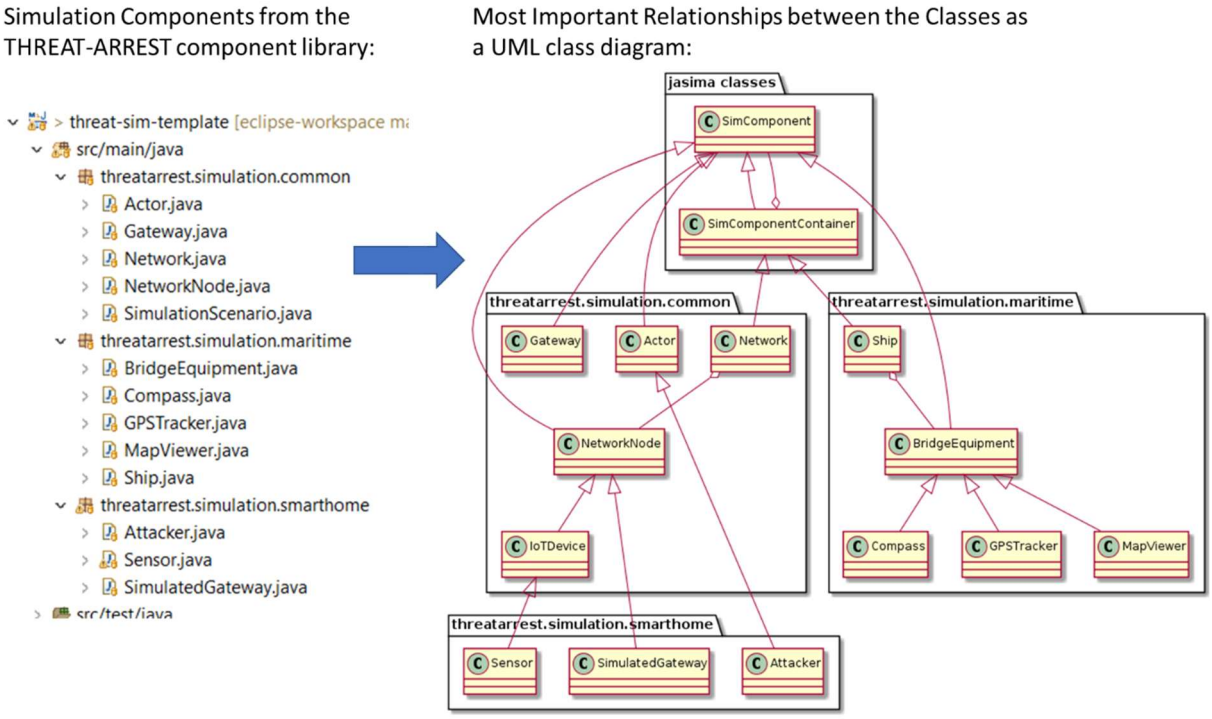

Figure 3 Exemplary Components Available in the Project Template / Component Library

2.6 Network Generation and Instantiation

As stated before, running a simulation requires a scenario definition in XML format. This XML document describes the hierarchy of components including their parameter values that are supposed to be used for particular simulated components. Internally each component is an instance of a component class, so either one of the standard components offered in the component library or a custom type defined individually for a particular message. The XML format used by Jasima to store simulation scenarios uses a generic mapping between the Java Object graph. A generator class is used internally to create and parameterize all objects contained in the XML file, creating a hierarchy of simulation components parameterized as required by the straining scenario.

Internally the component hierarchy is wrapped in a SimulationExperiment as its top-level element defining additional attributes of the simulation run such as the initial simulation time.

```
<jasima.core.simulation.SimulationExperiment>
  <name>exp1</name>
  <initialSeed>902790708899</initialSeed>
  <logLevel>ALL</logLevel>
  <simulationLength>1440.0</simulationLength>
  <initialSimTime>0.0</initialSimTime>
  <statsResetTime>0.0</statsResetTime>
  <simTimeToMillisFactor>60000</simTimeToMillisFactor>
  <simTimeStartInstant>2019-06-25T21:40:35.638Z</simTimeStartInstant>
  <rootComponent class="threatarrest.simulation.common.SimulationScenario">
    <name>scenario1</name>
    <components>
      <threatarrest.simulation.common.Network>
        <name>mainNet</name>
        <components>
          <threatarrest.simulation.smarthome.Sensor>
            <name>sensor1</name>
            <tempSlope>1.0</tempSlope>
            <tempVariance>1.0</tempVariance>
          </threatarrest.simulation.smarthome.Sensor>
          <threatarrest.simulation.smarthome.Sensor>
            <name>sensor2</name>
            <tempSlope>-1.0</tempSlope>
            <tempVariance>1.0</tempVariance>
          </threatarrest.simulation.smarthome.Sensor>
          <threatarrest.simulation.smarthome.Sensor>
            <name>sensor3</name>
```

Figure 4 Excerpt of a Simulation Scenario Definition in XML format

An example of such an XML scenario definition is shown in Figure 4. It starts with an element for the SimulationExperiment, subsequently defining certain attributes of a simulation run. The component hierarchy to be used, starts with the tag rootComponent in line 10. It defines a root component of type “SimulationScenario” named “scenario1” (just used here as a generic top-level component container). Which contains a component of type “Network” (named “mainNet”) which in turn contains come sensor components from the smart-home-specific part of the component library. One of these is named “sensor1”. This was exactly the component used by the example on how simulation state is shared among platform components with a value name of ‘scenario1.mainNet.sensor1.currentTemperature’. The fully qualified name of this sensor is scenario1.mainNet.sensor1 as it is contained in the mainNet component which itself is contained in the scenario1 root component. The value name “currentTemperature” references an observable attribute of this name that is used during simulation execution and defined in the Sensor class.

2.7 Future Integration Aspects

For the Simulation Tool, there are a number of integration requirements with other platform components. The reader is also referred to deliverable D5.3 for a more in-depth description of the interactions between the Simulation Tool and other components of the THREAT-ARREST platform. Implementation of the functionality to meet these requirements will be added in future versions of the simulation components, as the platform components and their integration in the overall THREAT-ARREST platform architecture progress further.

2.7.1 Emulation

As stated before, the simulation will run inside a VM within the emulated environment (see work package 2). This way it can directly communication with emulated components/VMs in their native protocol (as long as it is IP-based). This way it is possible for simulation components to directly interact with software running inside such VMs, trigger actions and base

further simulation actions on the information received from these emulated components (e.g., a simulated attacker could try to login in some VM using some default password).

Another aspect to be worked on in a future version is the topic of clock-synchronization. In a discrete-event simulation time can jump if there are no events during a certain time period. This is more difficult in an emulated components where time progresses continuously using normal speed. For certain scenarios it might be required to use a time-synchronization service to set the time of VM's to the current simulation but also to synchronize simulation time with the a VM's time based, e.g., on the duration a certain action took in a VM.

2.7.2 Data Fabrication

To integrate with the IBM's Data Fabrication Platform (DFP), a dedicated simulation component will be developed. It can access a fabricated event log using the DFP's REST API (for dynamically created data) or access pre-fabricated log data created for a particular training session. This log will then be executed by performing certain simulation actions at exactly the points in time and using the parameters specified in the log. This way the log file can act as a kind of script to trigger actions in the simulation synchronized with the time of the training session.

2.7.3 Interactive Simulation (User Integration)

A new aspect for Jasima is the requirement to be able to perform interactive simulation runs. Potentially multiple users at the same time have to be able to interact with a simulated component of a simulation run, view its state and potentially even interact with it and trigger actions in them. The complete functionality to implement this feature will be added later in the project; conceptually many instances of the JVT can access a single simulation session already in the current version, but concurrent actions from multiple sources will require additional implementation work.

3 CTTP Model-Driven Simulation

3.1 Development sub-model

The THREAT-ARREST platform operation will be driven by the CTTP models. Among others, a CTTP model contains information regarding how to instantiate the various platform modules and facilitate the training procedures.

The development subset of a valid CTTP model determines the operational aspects of the pilot system, how they can be deployed, and their connections. The following figure illustrates the main elements of the development sub-model and Appendix I presents an initial version of its schema.

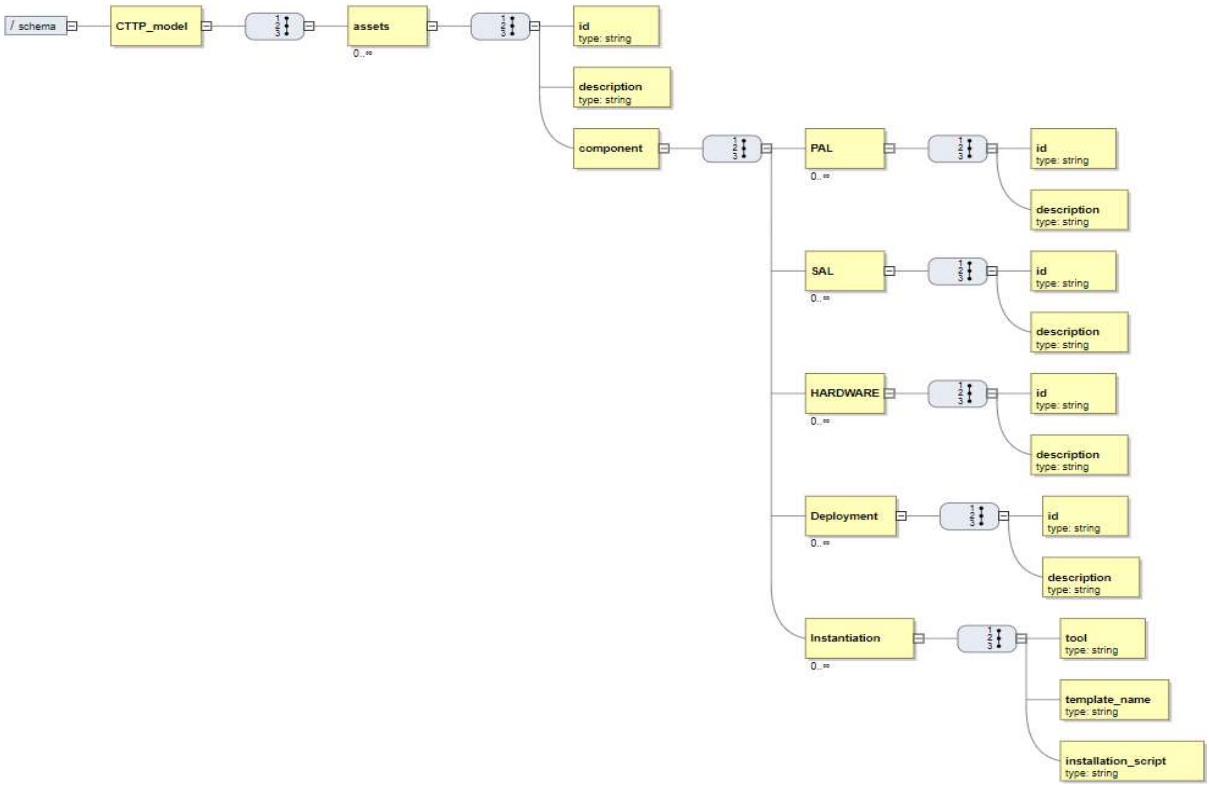

Figure 5 The development sub-model schema

Every asset is broken down into several components that describe its actual structure. These components can model the Platform Level software (PAL) (i.e. operating system), the Software Architecture Layer (SAL) (e.g. software applications), the HARDWARE modules, and how these PAL/SAL/HARDWARE elements are connected and deployed. Each of these three component types has a *unique ID* and a *brief description*, presenting to the user the main details for the component.

The Deployment sub-model also determines the procedure of developing/instantiating the component in the related platform tools. This may include the interoperation of real equipment (administrated via the Assurance Tool) or emulation/simulation of it. The CTTP designer has to define the relevant tool/s and how the platform can instantiate the component.

For the simulated components, two options are supported:

1. **Build from the scratch:** a tool-specific script that describes how to build and install a new instance of the component
2. **Pre-set:** if the component has been set in advance, we can deploy new template instances of it (e.g. in case of simulation or emulation). This is very helpful, as some

aspects of a real system and its interplay with the THREAT-ARREST modules (e.g. fabricated logs, embedded assurance controls, etc.) could be quite complex to be included in the model.

3.2 Simulation Instantiation script

In order to instantiate a simulated component (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)), we have to provide a valid script to the Simulation Controller (see Section 2) for configuring the Jasima simulator. The Deployment sub-model contains the related script for every simulated component (either to build it from the scratch or to instantiate an existing pre-configured simulation setting).

At the current version, the information in the asset's 'description' and the related 'instantiation script' are correlated, but they are set separately and manually by the CTTP designer in order to facilitate the implementation process.

An example of instantiating simulated components is detailed in the following subsections. Jasima can run either in a dedicate machine or in a VM. The first case is described below.

In the second case, prior the Jasima instantiation scripts, we need to instantiate the Jasima VM. This is formed as an additional instantiation script for the Emulated Tool (Ubuntu 19 OS, 20GB hard-disk, 16GB RAM, Jasima simulator), which must be executed first. Then, aforementioned simulation instantiation scripts are parsed to the Jasima installation in this VM. The deployment of VMs through the Emulation Tool (OpenStack) is presented in the related work package (WP) 2 deliverables (D2.1 and D2.3).

3.3 Example – Smart shipping scenario

This subsection presents a complete example for modelling a simulated component for the smart shipping scenario (Use Case 3) and how to instantiate it during a training session.

3.3.1 Simulated component – The on-deck monitoring equipment

As an example in the smart shipping scenario, we present the simulated component for the on-deck monitoring equipment of the vessel. This includes the closed system of the Deck Control Center (DCC) that collects information from on-ship devices (e.g. navigation equipment, sensors, etc.) and displays it to the on-deck crew and the captain.

3.3.2 Scenario – GPS spoofing

This scenario simulates a runtime attack on the ship and targets the captain (evaluable actuator with main security knowledge) or the on-deck crew (system operators with low security training). As the vessel navigates from Heraklion to Piraeus, a hacker launches a Global Positioning System (GPS) spoofing attack. The attack starts changing the GPS signals (DEP3) that are recorded by the on-ship equipment (PAL3/SAL3/HARDWARE3 and PAL4/SAL4/HARDWARE2) in an attempt to make the vessel deviate significantly from its course. The variations on the signal are smooth and gradual in order to make the detection of the malicious effects harder. The trainees are monitoring the overall navigating operations from the on-deck monitors. Normally, there are alternative means for observing and assessing the validity of the current position of the vessel as shown by different on-board equipment, such as GPS monitor, physical compass, live maps, etc. Thus, the trainees will have to monitor all of them, detect potential differences, and perform the designated actions in order to contain the misbehaving equipment and reach the correct destination.

The scenario is implemented in the Simulation Tool. The various positioning modules are modelled as different nodes of the simulated on-deck monitoring and management

infrastructure. Each module has its own operational behaviour (e.g. normal, faulty, or compromised), which have been modelled beforehand and are set by the trainer during the scenario's initialization. Figure 6 depicts the related UML class diagram for the navigation equipment.

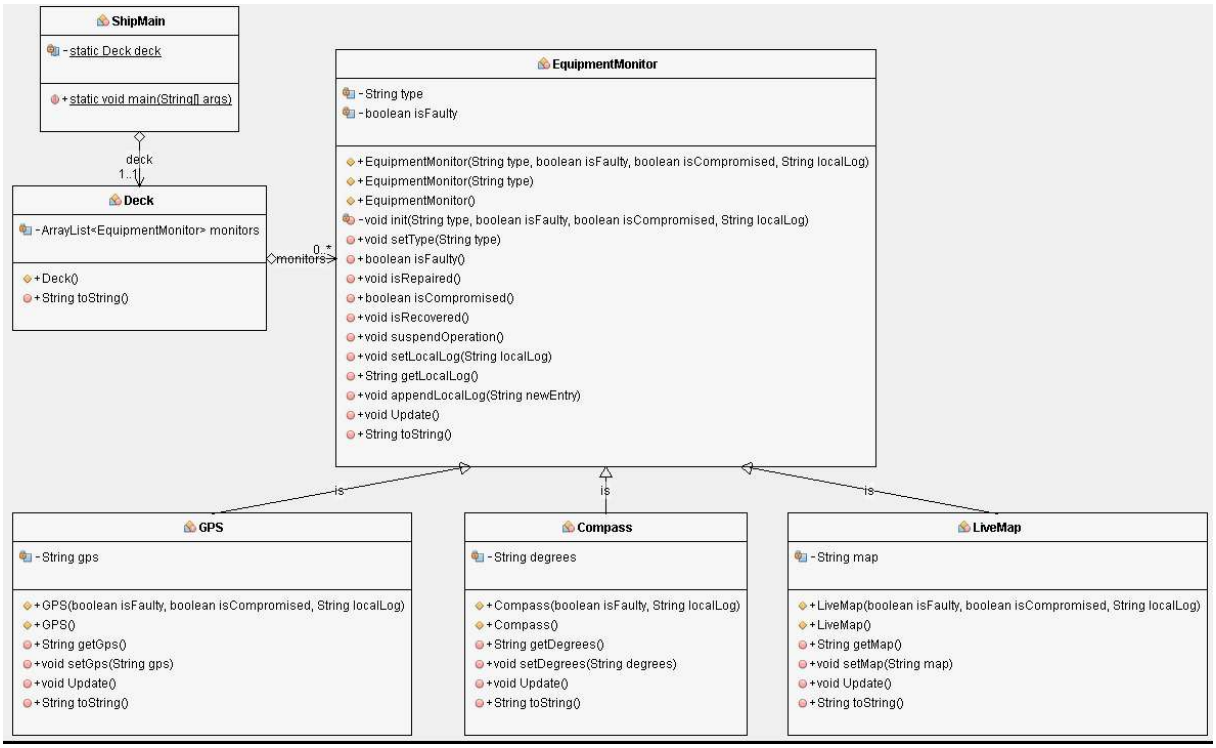

Figure 6 The UML class diagram of the simulated on-deck equipment

The simulated trip lasts 10 minutes (while the actual one would take several hours). The trainee must observe the on-deck modules, discriminate normal, faulty, and/or compromised operational behaviours, and perform the appropriate actions (e.g. suspend its operation).

3.3.2.1 Scenario propagation

- Normal operation where no anomaly regarding the vessel's position is modelled.
- Faulty operation of one positioning module (i.e. GPS) where a minor and steady deviation is recorded, compared to the actual value.
- GPS spoofing attack where a hacker manipulates the data that is processed by the on-deck GPS module and tries to change the ship's route.

The next picture depicts the three potential routes.

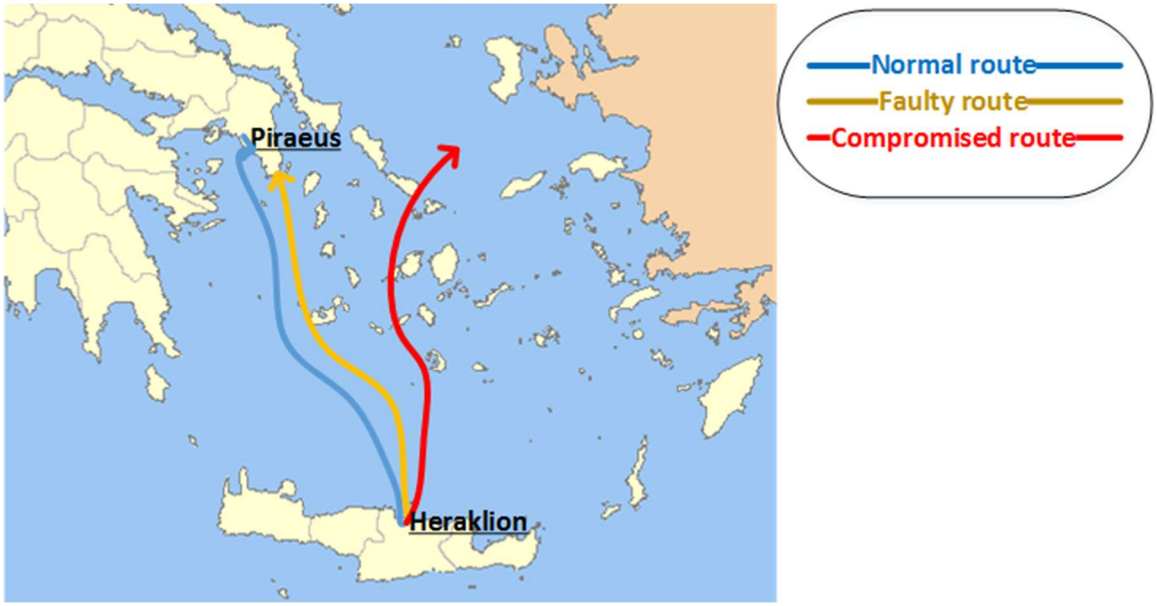

Figure 7 The three potential routes of the simulated scenario based on the state of the GPS equipment

3.3.2.2 Scenario modelling

For modelling the scenario, we first have to deploy a dedicated computer or instantiate a VM that runs the Jasima simulator. In the simulator, we model the on-deck infrastructure, where each equipment monitor is defined as a different node. Specifically for this scenario, we determine the positioning modules (GPS, compass, live maps) along with the supported operational behaviours for each one of them (normal, faulty, compromised). The various GPS signals could be generated by the IBM's DFP and drive the simulation accordingly.

3.3.3 CTTP model

The code sample in Appendix II models the development sub-model for the aforementioned asset in an XML format. The characteristics of the DCC are described in the SAL, PAL, and Deployment elements.

The physical characteristics (HARDWARE2-HARDWARE3) of the DCC are:

- Hardware
    - On-deck monitors
    - Sensory or other equipment (e.g. navigation components, on-ship sensors for fuel consumption, exhaust emissions, etc.)
    - LAN Router
- Connectivity
    - Ethernet

The software characteristics (PAL3/SAL3-PAL4/SAL4) of the DCC are:

- The monitors' visualization interfaces
- The software that performs the main functionality of each component

The Deployment feature (DEP3) that is correlated with the DCC is:

- *PAL3/SAL3-PAL4/SAL4* (software of the navigation equipment and the on-deck monitoring/management modules) runs on *HARDWARE2-HARDWARE3* (navigation equipment and their deck monitoring and management modules)

Two instantiation scripts are detailed in the subsections below. In the first one, we configure the related simulation from the scratch, while in the second case we deploy a pre-configured simulation setting of the DCC.

3.3.4 Instantiation script 1 – Build simulation from the scratch

The following piece of code describes how to build a simulation with the on-deck equipment from the scratch. 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 simple 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>
```

The Simulator Controller will extract the 'installation_script' and create a file 'DCC-sim-scenario.xml' that contains the installation script for Jasima. Then, it will run the command:

- java –jar jasima-threatarrest.jar /HOME_DIR/dir/DCC-sim-scenario.xml

The components can be also initialized with a specific navigation log, provided by the DFP at instantiation time for modelling different trips. Alternatively, specific logs of each behaviour (normal, faulty, compromised) for every component could have been produced in advance and fed into a pre-configured simulation. In a more advanced THREAT-ARREST implementation phase, the DFP would produce the navigation events and fed the simulation at runtime.

3.3.5 Instantiation script 2 – Deploy a pre-configured simulation

The next piece of code deploys a pre-configured simulation of the DCC.

```
<Instantiation>
    <tool>Simulation</tool>
    <template_name>jasima-threatarrest-DCC.jar</template_name>
    <installation_script>NO</installation_script>
</Instantiation>
```

The Simulator Controller will extract the 'template_name' of the pre-configured simulation and run the command:

- java –jar jasima-threatarrest-DCC.jar

4 Conclusion

This deliverable presents the initial version of the Simulation Tool. It is the first document of the task "T5.1 – Simulated components' generator". The Simulation Tool is based on the Java simulator Jasima. The tool gets as input the simulation sub-model from the Assurance and Training Tools, extracts the related instantiation script, and deploys the simulated components. The simulations can also utilize synthetic logs and events, which are generated by the IBM's Data Fabrication Platform (DFP). In this version, we describe how to model a training scenario for the smart transportation use case.

The CTTP models are detailed in the WP3 while the data fabrication is presented in D5.1. The overall interconnections of the Simulation Tool are defined in D5.3. In the second year of the project, the Simulation Tool will be further documented and concrete simulations will be implemented, interplaying with the other platform tools. This work will be documented in the related D5.6 until the 28th month of the project.

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.

Appendix I – CTTP Development sub-model Schema

This appendix presents an initial schema version of the CTTP development sub-model. The schema is an XML Schema Definition (XSD) file.

```
[1]  <?xml version="1.0" encoding="UTF-8"?>
[2]  <xs:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
[3]   xmlns:xs="http://www.w3.org/2001/XMLSchema"
[4]   xmlns:hfp="http://www.w3.org/2001/XMLSchema-hasFacetAndProperty">
[5]  <xs:element minOccurs="0" maxOccurs="unbounded" abstract="true" name="CTTP_model">
[6]  <xs:complexType>
[7]   <xs:sequence>
[8]   <xs:element minOccurs="0" maxOccurs="unbounded" name="assets">
[9]     <xs:complexType>
[10]     <xs:sequence>
[11]       <xs:element name="id" type="xs:string" use="required"/>
[12]       <xs:element name="description" type="xs:string" use="required"/>
[13]       <xs:element minOccurs="1" maxOccurs="1" name="component">
[14]           <xs:complexType>
[15]            <xs:sequence>
[16]              <xs:element minOccurs="0" maxOccurs="unbounded " name="PAL">
[17]               <xs:complexType>
[18]                  <xs:sequence>
[19]                     <xs:element name="id" type="xs:string" use="required"/>
[20]                     <xs:element name="description" type="xs:string" use="required"/>
[21]                  </xs:sequence>
[22]               </xs:complexType>
[23]              </xs:element>
[24]              <xs:element minOccurs="0" maxOccurs="unbounded " name="SAL">
[25]               <xs:complexType>
[26]                  <xs:sequence>
[27]                     <xs:element name="id" type="xs:string" use="required"/>
[28]                     <xs:element name="description" type="xs:string" use="required"/>
[29]                  </xs:sequence>
[30]               </xs:complexType>
[31]              </xs:element>
[32]              <xs:element minOccurs="0" maxOccurs="unbounded " name="HARDWARE">
[33]               <xs:complexType>
[34]                  <xs:sequence>
[35]                     <xs:element name="id" type="xs:string" use="required"/>
[36]                     <xs:element name="description" type="xs:string" use="required"/>
[37]                  </xs:sequence>
[38]               </xs:complexType>
[39]              </xs:element>
[40]              <xs:element minOccurs="0" maxOccurs="unbounded " name="Deployment">
[41]               <xs:complexType>
[42]                  <xs:sequence>
[43]                     <xs:element name="id" type="xs:string" use="required"/>
[44]                     <xs:element name="description" type="xs:string" use="required"/>
[45]                  </xs:sequence>
[46]               </xs:complexType>
[47]              </xs:element>
[48]              <xs:element minOccurs="0" maxOccurs="unbounded " name="Instantiation">
[49]              <xs:complexType>
[50]                  <xs:sequence>
[51]                     <xs:element name="tool" type="xs:string" use="required"/>
[52]                     <xs:element name="template_name" type="xs:string" use="required"/>
[53]                     <xs:element name="installation_script" type="xs:string" use="required"/>
[54]                  </xs:sequence>
[55]               </xs:complexType>
```

```
[1]                         </xs:element>
[2]                       </xs:sequence>
[3]                     </xs:complexType>
[4]                 </xs:element>
[5]             </xs:sequence>
[6]         </xs:complexType>
[7]         </xs:element>
[8]         </xs:sequence> </xs:complexType> </xs:element> </xs:schema>
```

Appendix II – CTTP Development sub-model for the on-deck equipment

This appendix presents the CTTP development sub-model for instantiating the on-deck monitoring and management equipment (see subsection (5.3). The scripts for the two instantiation types (build form the scratch or deploy a pre-configured setting) are detailed in the subsections 5.3.3 and 5.3.4 respectively.

```
[9]  <?xml version="1.0" encoding="UTF-8"?>
[10]   <!--CTTP Model for the vessel's on-deck equipment-->
[11]   <CTTP_model xmlns="http://www.w3schools.com"
[12]          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
[13]          xsi:schemaLocation="http://www.w3schools.com file:///C:/CTTP_Model.xsd">
[14]     <name>Smart_Shipping_CTTP</name>
[15]     <assets>
[16]       <id>ASSET2</id>
[17]       <description>PCs/devices of the closed Deck Control Center (DCC) system. The DCC will
       simulate the main components that are required for the on-deck control by the captain and the crew.
       In the current scenario, we model the navigation equipment of the ship (i.e. GPS, live-maps, and
       compass).</description>
[18]       <component>
[19]         <PAL>
[20]           <id>PAL3-PAL4</id>
[21]           <description>The OS for the sensory or other equipment, which is administrated through
       the  DCC.</description>
[22]         </PAL>
[23]         <SAL>
[24]           <id>SAL3-SAL4</id>
[25]           <description>Monitoring/management modules for the sensory or other equipment, which
       is administrated through the  DCC. It also models the software that runs on the underlying sensory or
       other equipment.</description>
[26]         </SAL>
[27]         <HARDWARE>
[28]           <id>HARDWARE2-HARDWARE3</id>
[29]           <description>Sensory or other equipment (e.g. navigation modules and sensors for fuel
       consumption, exhaust gas, etc.) and their on-deck monitors. </description>
[30]         </HARDWARE >
[31]         <Deployment>
[32]           <id>DEP3</id>
[33]           <description>PAL3/SAL3-PAL4/SAL4 are deployed upon HARDWARE2-
       HARDWARE3</description>
[34]         </Deployment>
[35]
       [36]   <!--(Optional) Instantiate the Jasima VM first-->
[37]         <Instantiation>
[38]           <tool>Emulation</tool>
[39]           <!--see D2.1-->
[40]         </Instantiation>
[41]
       [42]   <!--One of the two 'Instantiation' choices will be made by the CTTP designer-->
[43]         <Instantiation>
[44]           <tool>Simulation</tool>
[45]           <template_name>NO</template_name>
[46]           <installation_script>-- Jasima Script / Subsection 5.3.3 --</installation_script>
[47]         </Instantiation>
[48]         <Instantiation>
[49]           <tool>Simulaiton</tool>
[50]           <template_name>-- Jasima Script / Subsection 5.3.4 --</template_name>
[51]           <installation_script>NO</installation_script>
```

```
[52]            </Instantiation>
[53]        </component>
[54]    </assets>
[55] </CTTP_model>
```