Project Number 101017258

D7.4 Open-Source Software Components for Explainable EDDIs

Version 1.0
4 July 2023
Final

Public Distribution

Fraunhofer IESE, University of Hull and FORTH

Project Partners: **Aero41, ATB, AVL, Bonn-Rhein-Sieg University, Cyprus Civil Defence, Domaine Kox, FORTH, Fraunhofer IESE, KIOS, KUKA Assembly & Test, Locomotec, Luxsense, The Open Group, Technology Transfer Systems, University of Hull, University of Luxembourg, University of York**

PROJECT PARTNER CONTACT INFORMATION

Aero41 Frédéric Hemmeler Chemin de Mornex 3 1003 Lausanne Switzerland E-mail: frederic.hemmeler@aero41.ch	**ATB** Sebastian Scholze Wiener Strasse 1 28359 Bremen Germany E-mail: scholze@atb-bremen.de
AVL Martin Weinzerl Hans-List-Platz 1 8020 Graz Austria E-mail: martin.weinzerl@avl.com	**Bonn-Rhein-Sieg University** Nico Hochgeschwender Grantham-Allee 20 53757 Sankt Augustin Germany E-mail: nico.hochgeschwender@h-brs.de
Cyprus Civil Defence Eftychia Stokkou Cyprus Ministry of Interior 1453 Lefkosia Cyprus E-mail: estokkou@cd.moi.gov.cy	**Domaine Kox** Corinne Kox 6 Rue des Prés 5561 Remich Luxembourg E-mail: corinne@domainekox.lu
FORTH Sotiris Ioannidis N Plastira Str 100 70013 Heraklion Greece E-mail: sotiris@ics.forth.gr	**Fraunhofer IESE** Daniel Schneider Fraunhofer-Platz 1 67663 Kaiserslautern Germany E-mail: daniel.schneider@iese.fraunhofer.de
KIOS Maria Michael 1 Panepistimiou Avenue 2109 Aglatzia, Nicosia Cyprus E-mail: mmichael@ucy.ac.cy	**KUKA Assembly & Test** Michael Laackmann Uhthoffstrasse 1 28757 Bremen Germany E-mail: michael.laackmann@kuka.com
Locomotec Sebastian Blumenthal Bergiusstrasse 15 86199 Augsburg Germany E-mail: blumenthal@locomotec.com	**Luxsense** Gilles Rock 85-87 Parc d'Activités 8303 Luxembourg Luxembourg E-mail: gilles.rock@luxsense.lu
The Open Group Scott Hansen Rond Point Schuman 6, 5th Floor 1040 Brussels Belgium E-mail: s.hansen@opengroup.org	**Technology Transfer Systems** Paolo Pedrazzoli Via Francesco d'Ovidio, 3 20131 Milano Italy E-mail: pedrazzoli@ttsnetwork.com
University of Hull Yiannis Papadopoulos Cottingham Road Hull HU6 7TQ United Kingdom E-mail: y.i.papadopoulos@hull.ac.uk	**University of Luxembourg** Miguel Olivares Mendez 2 Avenue de l'Universite 4365 Esch-sur-Alzette Luxembourg E-mail: miguel.olivaresmendez@uni.lu
University of York Simos Gerasimou & Nicholas Matragkas Deramore Lane York YO10 5GH United Kingdom E-mail: simos.gerasimou@york.ac.uk nicholas.matragkas@york.ac.uk	

DOCUMENT CONTROL

Version	Status	Date
0.1	Initial outline	10 May 2023
0.6	Internal review version ready	22 June 2023
0.8	Internal review done by TTS	27 June 2023
0.9	Internal review done by BRSU	30 June 2023
1.0	Review changes integrated, ready for submission	4 July 2023

TABLE OF CONTENTS

TABLE OF FIGURES

EXECUTIVE SUMMARY

Executable Digital Dependability Identities (EDDIs) are meant to be deployed across significantly diverse applications and complex Multi-Robot System (MRS) architectures, also featuring Artificial Intelligence (AI) and Machine Learning (ML) models. Explainability of the embedded EDDI models can support the development of robust, comprehensive, and trustworthy MRS.

In this deliverable, we present the different tools developed and upgraded over the course of SESAME for the purpose of explaining EDDI state and/or behaviour. These tools will be available as open-source software components via the SESAME GitHub repository at https://github.com/sesame-project/explainable_eddis, accompanied by documentation and examples.

LIST OF ABBREVIATIONS

EDDI	Executable Digital Dependability Identity	ODE	Open Dependability Exchange
ODD	Operational Design Domain	ROS	Robot Operating System
ConSerts	Conditional Safety Certificates	XML	Extensible Markup Language
YAML	Yaml Ain't Markup Language	EGL	Epsilon Generation Language
SESAME	Secure and Safe Multi-Robot Systems	MRS	Multi Robot System
JAR	Java Archive	API	Application Programming Interface
XDSL	XML-based Bayesian Network format of the GeNIe toolset	GeNie	Commercial tool for Bayesian network modelling and inference
IDE	Integrated Development Environment	DT	Design Time
PCA	Principle Component Analysis	Hz	Hertz
EBNF	Extended Backus–Naur Form	RtE	Runtime Evidence
MROS	Model Based ROS	RT	Runtime

1. INTRODUCTION

In this deliverable, tools for assisting end-users in understanding the state of a Runtime EDDI (RT EDDI) are presented. Such explainability is important both when developing Multi-Robot Systems (MRS), as well as during operation. In the former case, explainability supports verification of the RT EDDI output, whereas in the latter, explainability supports overseeing the RT EDDI. While the specific tool depends on the type of underlying RT EDDI, the common theme across tools is the emphasis of visualisation-based techniques. At the time of writing, some of these tools support explainability in an ‘offline’ fashion (i.e. they allow visualisation of RT EDDI execution traces rather than at runtime).

RT EDDIs are conceptually described in deliverable D7.1 (see Figure 1). Concretely, these constituents collaboratively performing dynamic risk management are: 1. Dynamic Safety Capability Assessment with Conditional Safety Certificates (*ConSerts*), 2. Situation-Aware Dynamic Risk Assessment with Bayesian Networks, 3. Dynamic Reliability Assessment based on the *SafeDrones* tool, 4. Perception Uncertainty Monitoring with the *SafeML* tool and 5. conditional event monitoring to realize a typed communication interface between the *runtime* EDDI and the nominal functionality of the MRS. The toolset in deliverable D7.2 realizes the pipeline to make the engineered runtime models executable (i.e. generate components that can infer the runtime models based on runtime-available information) and deploy them into common robotic platform architectures.

The Robot Operating System (ROS) has been selected as the exemplary target runtime environment, to which the EDDI shall be deployed. The reason for this decision is the usage of ROS in several SESAME use cases and the general spread of ROS in the robotics domain. Thus, by having support for deploying runtime EDDIs to ROS applications, the transfer of EDDIs to industrial applications is facilitated. By splitting the runtime EDDI generator pipeline in platform-independent and platform-dependent parts, the extension towards other runtime environments is conceptually and technically simplified.

The developed tools provide users with visualisation or explanation of the state of RT EDDI models at runtime (or during a recorded runtime session). The tools described in this deliverable can be found in the SESAME public GitHub repository at: https://github.com/sesame-project/explainable_eddis.

The rest of the deliverable is structured as follows. In Section 2, we briefly review requirements and motivation related to MRS development and operation for which RT EDDI explainability would be beneficial for the end user. In Section 3, we discuss the visualisation component for the Conditional Safety Certificate (ConSert) type of RT EDDI. In Section 5, we discuss a Machine Learning (ML) explainability approach which builds upon the SafeML approach presented previously in WP4 and WP7 (see D4.1, D4.2, D7.1 and D7.2). We conclude in Section 6 summarizing the deliverable’s main points and outlining next steps.

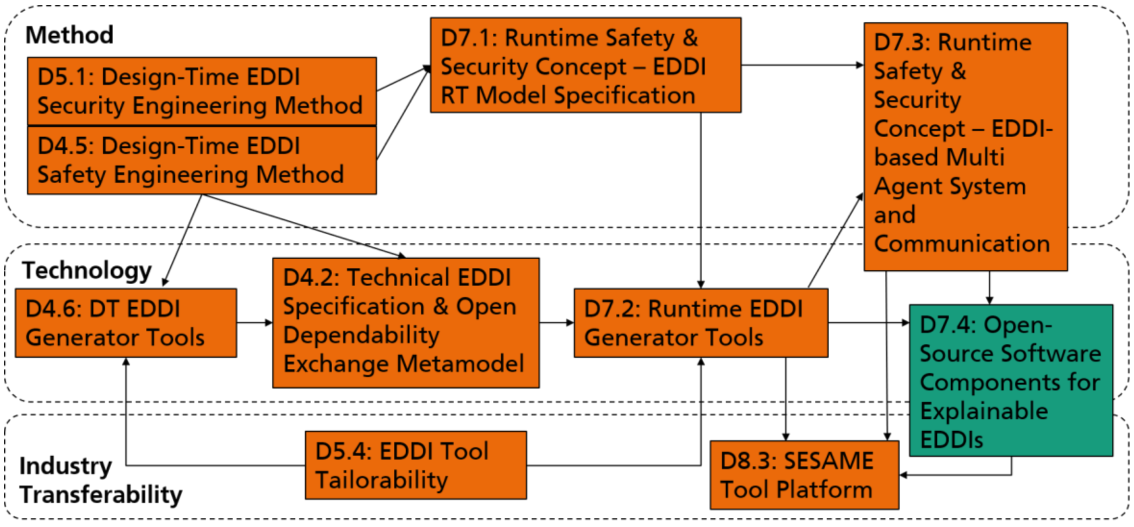

Method
D5.1: Design-Time EDDI Security Engineering Method
D4.5: Design-Time EDDI Safety Engineering Method
D7.1: Runtime Safety & Security Concept – EDDI RT Model Specification
D7.3: Runtime Safety & Security Concept – EDDI-based Multi Agent System and Communication
Technology
D4.6: DT EDDI Generator Tools
D4.2: Technical EDDI Specification & Open Dependability Exchange Metamodel
D7.2: Runtime EDDI Generator Tools
D7.4: Open-Source Software Components for Explainable EDDIs
Industry Transferability
D5.4: EDDI Tool Tailorability
D8.3: SESAME Tool Platform

Figure 1 Relationship of D7.4 to other deliverables

2. Motivation

As robotic systems find increasing applications involving more complex human and general interactions with their environment, the need to understand their perception and decision-making process is likely to also increase in importance. Existing work is interrogating this subject from different perspectives, including explainability for robotics in human interaction [1] [2] [3] [4], explainability of artificial intelligence (XAI) for robotics [5] and XAI in relation to agents and Multi-Agent Systems [6].

Given that EDDIs are intended to be part of the developed Multi-Robot System (MRS), it is also relevant to support their explainability, which promises the following key benefits:

- Supports the development of robust models, as developers' understanding of the EDDI state can be interrogated more comprehensively. Particularly for Machine Learning (ML)-related elements, see e.g. SMILE in section 5, such an understanding is essential to efficiently test and improve upon the underlying ML model.

- Improves feedback from field operation data. Similar to development-time benefits, enhancing the explainability of field data can, potentially, significantly increase the value in terms of fault identification and insight acquisition. Both of these information instances can be then exploited to improve the MRS dependability.

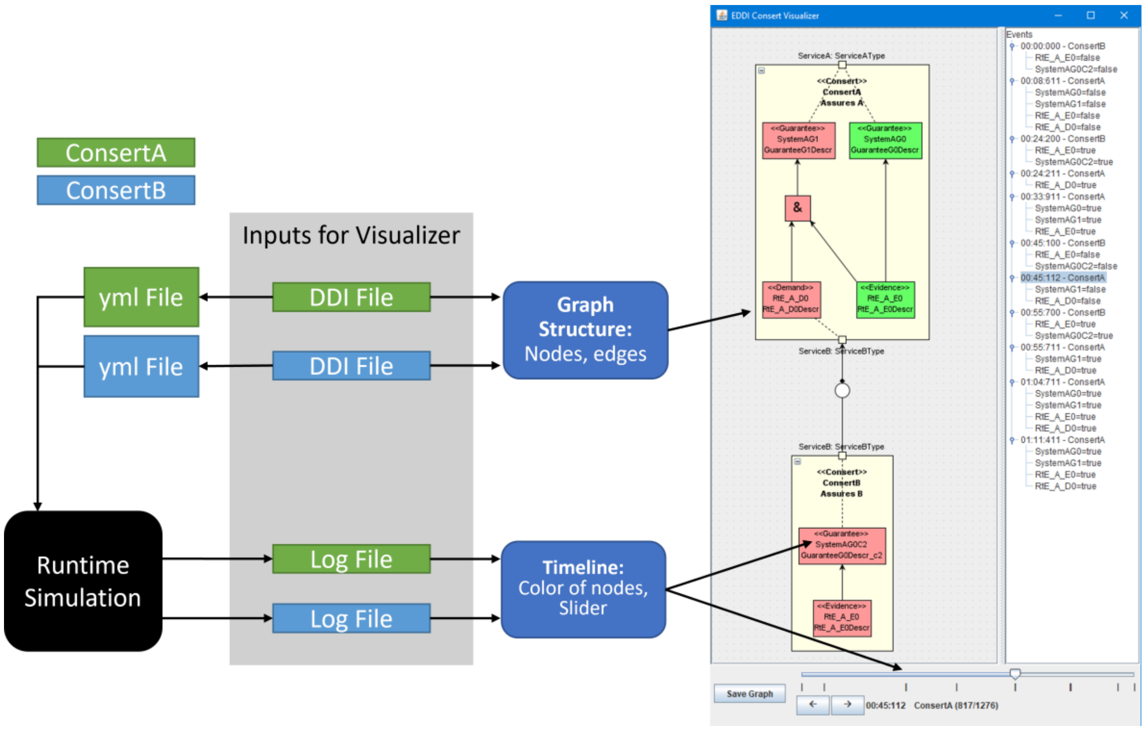
- Improves operator/end-user comprehension of MRS behaviour. MRS could potentially feature multiple highly complex and emergent robotic behaviours. Therefore, having means of associating observed behaviours with an explanation of the state of the RT EDDIs deployed within them can help reduce uncertainty and increasing MRS trustworthiness.

3. ConSerts Visualiser

Conditional Safety Certificates (ConSerts) are specified at development time based on safety requirements, as described in SESAME deliverable document D7.1. They are defined for each adaptable system or robot that can operate in a system-of-systems or Multi-Robot System (MRS) respectively to perform an overall service safely. At runtime/ simulation time, conditions in terms of demanded safety requirements (specified for required services, provided by other collaborating robots) and internal measurable safety requirements, aka Runtime Evidence (RtE), are evaluated to determine which safety requirement can be guaranteed for the provided service. ConSerts can be represented in form of a RT EDDI conforming to the ODE metamodel, described in the SESAME deliverable documents D4.1 (initial version) and D4.6 (updated version).

Consider that each robot in an MRS can have its own EDDI containing its respective ConSerts defined. Additionally, the robots in an MRS do not necessarily have to be developed by the same manufacturer. Thus, the composition of the collaborating robots' ConSerts trees is not necessarily known at development time. Therefore, to understand and review the composition of these ConSerts and how they evaluate given certain fulfilment states of internal RtEs at runtime/simulation-time, a visualisation of said aspects would be useful. The EDDI ConSert Visualiser fulfils these requirements, as it gives insights into:

1. the collaborating robots' overall ConSerts composition and
2. the execution-time-related safety requirements fulfilment based on data collected during runtime/simulation-time and the collaborating robots' EDDIs.


ConsertA
ConsertB
Inputs for Visualizer
yml File
DDI File
yml File
DDI File
Graph Structure: Nodes, edges
Runtime Simulation
Log File
Log File
Timeline: Color of nodes, Slider
EDDI Consert Visualizer
ServiceA: ServiceAType
<<Consert>> ConsertA Assures A
<<Guarantee>> SystemAG1 GuaranteeG1Descr
<<Guarantee>> SystemAG0 GuaranteeG0Descr
&
<<Demand>> RtE_A_D0 RtE_A_D0Descr
<<Evidence>> RtE_A_E0 RtE_A_E0Descr
ServiceB: ServiceBType
ServiceB: ServiceBType
<<Consert>> ConsertB Assures B
<<Guarantee>> SystemAG0C2 GuaranteeG0Descr_c2
<<Evidence>> RtE_A_E0 RtE_A_E0Descr
Save Graph
00:45:112 ConsertA (817/1276)

```
Events
00:00:000 - ConsertB
    RtE_A_E0=false
    SystemAG0C2=false
00:08:611 - ConsertA
    SystemAG0=false
    SystemAG1=false
    RtE_A_E0=false
    RtE_A_D0=false
00:24:200 - ConsertB
    RtE_A_E0=true
    SystemAG0C2=true
00:24:211 - ConsertA
    RtE_A_D0=true
00:33:911 - ConsertA
    SystemAG0=true
    SystemAG1=true
    RtE_A_E0=true
00:45:100 - ConsertB
    RtE_A_E0=false
    SystemAG0C2=false
00:45:112 - ConsertA
    SystemAG1=false
    RtE_A_D0=false
00:55:700 - ConsertB
    RtE_A_E0=true
    SystemAG0C2=true
00:55:711 - ConsertA
    SystemAG1=true
    RtE_A_D0=true
01:04:711 - ConsertA
    SystemAG0=true
    SystemAG1=true
    RtE_A_E0=true
    RtE_A_D0=true
01:11:411 - ConsertA
    SystemAG0=true
    SystemAG1=true
    RtE_A_E0=true
    RtE_A_D0=true
```

Figure 3.1 Overview of EDDI ConSert Visualiser Framework

Figure 3.1 depicts an example ConSerts composition visualised in the EDDI ConSerts Visualiser tool together with the required inputs and their origin. In order to visualise the composition and the internally-defined conditional safety requirements structure of each ConSert, the visualiser imports and analyses the EDDI files of each collaborating robot. The visualiser tool does not currently support the user to inspect the ConSerts' states live during runtime, but instead relies on the states of each ConSert logged during runtime to be imported afterwards into the tool. This way, the user is able to jump forwards and backwards through the recorded states to inspect the ones that are of interest. We intend to improve upon the component to allow it to also visualise live during runtime.

The EDDI ConSerts Visualiser is implemented in Java using the Java Swing framework[1] for the realization of the general GUI components. Additionally, the Java API (jGraphX[2]) of the mxGraph[3] diagramming library was used to allow the user to rearranging and resizing the visual components, as well as zooming and panning the diagram canvas. The customized graph appearance can be saved using the "Save Graph" button on the bottom left corner to allow the user to load the graph with the saved appearance the next time the Visualiser is started. In order to inspect the ConSerts' states for each point in time of the execution, the user has following options:

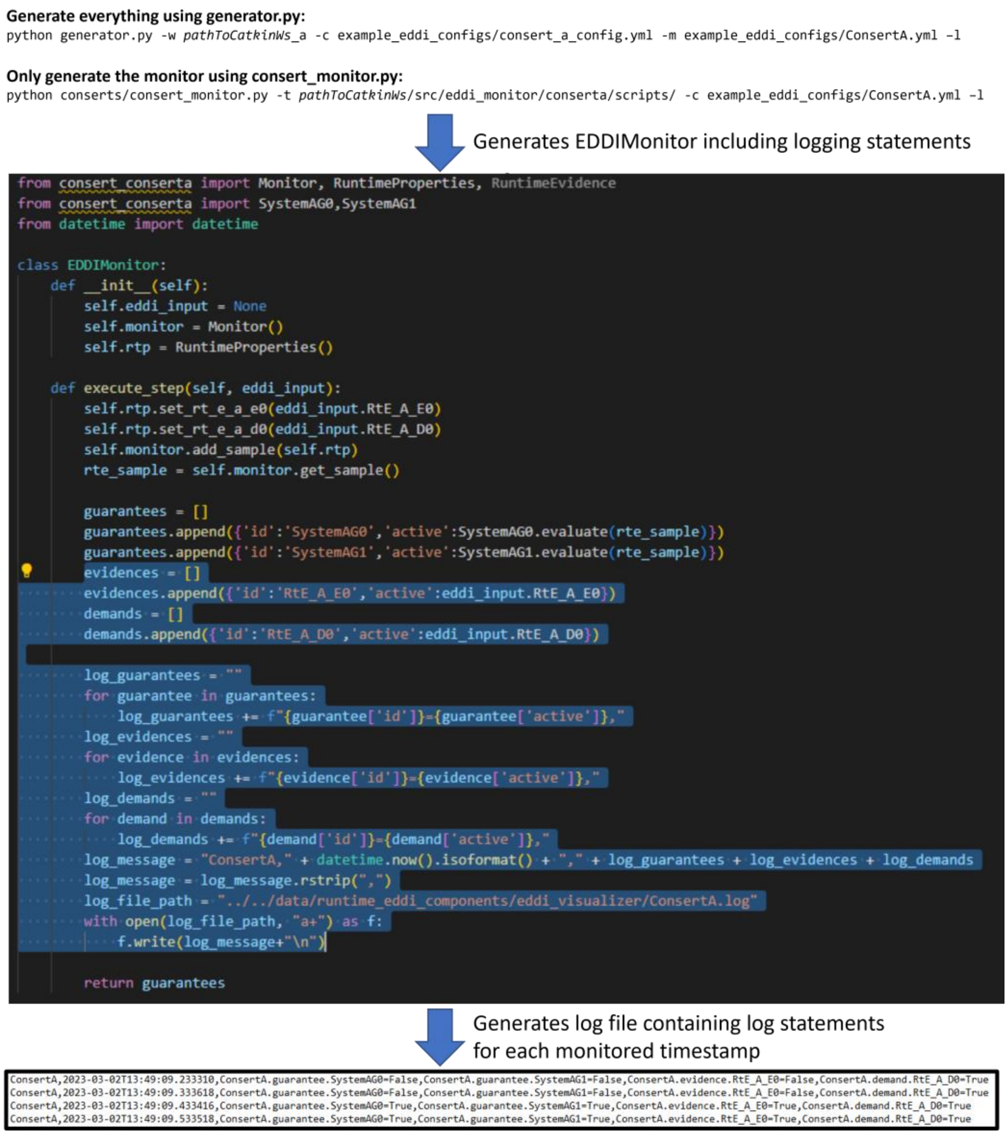
- Using the slider component at the bottom to reach each logged state.
- Using the left and right arrow buttons below the slider component to jump to those timepoints where events have occurred. Events are defined as timepoints where at least one safety requirement fulfilment state has changed.
- Using the events' listing on the right side of the visualiser provides an overview of all the relevant timepoints where states (indicating safety requirement fulfilment) have changed. The listing additionally gives insight into the changed states of the relevant safety requirements (true = fulfilled; false = not fulfilled).

When reaching an event, the appearance of the ConSerts graph will change in the sense that all nodes (safety requirements depicted as guarantees, demands and RtEs as well as logical gates) will be re-colourised. Nodes coloured in red indicate that the safety guarantees could not be provided at a given timepoint, whereas green-coloured nodes indicate that safety guarantees can be provided. Safety guarantee status is propagated upwards the ConSert trees, depending on the Boolean logic gates they connect to.

In order to provide the required logged states, the generation of ConSerts monitors, initially described in chapter 3 of the SESAME deliverable document D7.2, has been adapted. The generator can now be called using the command line with an additional parameter "-l" (lower case character "L") to instrument the ConSerts monitor such that the ConSerts state (state of safety requirements' fulfilment) is written to a log file each timestep. Figure 3.2 visualises the workflow in order to generate an extended ConSert monitor that additionally outputs a log file, where each line represents the safety

[1] https://docs.oracle.com/javase/8/docs/technotes/guides/swing/
[2] https://github.com/jgraph/jgraphx
[3] https://github.com/jgraph/mxgraph

requirements' fulfilment states at a specific point in time. The log file for each ConSert is written to the same directory the respective ConSert monitor is located in.


Generate everything using generator.py:
Only generate the monitor using consert_monitor.py:
Generates EDDIMonitor including logging statements
Generates log file containing log statements for each monitored timestamp

```
python generator.py -w pathToCatkinWs_a -c example_eddi_configs/consert_a_config.yml -m example_eddi_configs/ConsertA.yml -l
```

```
python conserts/consert_monitor.py -t pathToCatkinWs/src/eddi_monitor/conserta/scripts/ -c example_eddi_configs/ConsertA.yml -l
```

```python
from consert_conserta import Monitor, RuntimeProperties, RuntimeEvidence
from consert_conserta import SystemAG0,SystemAG1
from datetime import datetime

class EDDIMonitor:
    def __init__(self):
        self.eddi_input = None
        self.monitor = Monitor()
        self.rtp = RuntimeProperties()

    def execute_step(self, eddi_input):
        self.rtp.set_rt_e_a_e0(eddi_input.RtE_A_E0)
        self.rtp.set_rt_e_a_d0(eddi_input.RtE_A_D0)
        self.monitor.add_sample(self.rtp)
        rte_sample = self.monitor.get_sample()

        guarantees = []
        guarantees.append({'id':'SystemAG0','active':SystemAG0.evaluate(rte_sample)})
        guarantees.append({'id':'SystemAG1','active':SystemAG1.evaluate(rte_sample)})
        evidences = []
        evidences.append({'id':'RtE_A_E0','active':eddi_input.RtE_A_E0})
        demands = []
        demands.append({'id':'RtE_A_D0','active':eddi_input.RtE_A_D0})

        log_guarantees = ""
        for guarantee in guarantees:
            log_guarantees += f"{guarantee['id']}={guarantee['active']},"
        log_evidences = ""
        for evidence in evidences:
            log_evidences += f"{evidence['id']}={evidence['active']},"
        log_demands = ""
        for demand in demands:
            log_demands += f"{demand['id']}={demand['active']},"
        log_message = "ConsertA," + datetime.now().isoformat() + "," + log_guarantees + log_evidences + log_demands
        log_message = log_message.rstrip(",")
        log_file_path = "../../data/runtime_eddi_components/eddi_visualizer/ConsertA.log"
        with open(log_file_path, "a+") as f:
            f.write(log_message+"\n")

        return guarantees
```

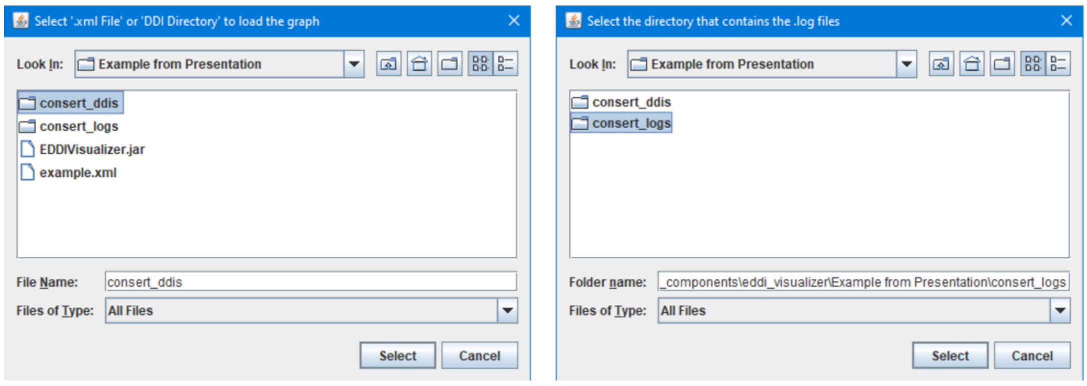
```
ConsertA,2023-03-02T13:49:09.233310,ConsertA.guarantee.SystemAG0=False,ConsertA.guarantee.SystemAG1=False,ConsertA.evidence.RtE_A_E0=False,ConsertA.demand.RtE_A_D0=True
ConsertA,2023-03-02T13:49:09.333618,ConsertA.guarantee.SystemAG0=False,ConsertA.guarantee.SystemAG1=False,ConsertA.evidence.RtE_A_E0=False,ConsertA.demand.RtE_A_D0=True
ConsertA,2023-03-02T13:49:09.433416,ConsertA.guarantee.SystemAG0=True,ConsertA.guarantee.SystemAG1=True,ConsertA.evidence.RtE_A_E0=True,ConsertA.demand.RtE_A_D0=True
ConsertA,2023-03-02T13:49:09.533518,ConsertA.guarantee.SystemAG0=True,ConsertA.guarantee.SystemAG1=True,ConsertA.evidence.RtE_A_E0=True,ConsertA.demand.RtE_A_D0=True
```

Figure 3.2 Logfile Generation Workflow

In order to start the visualiser after runtime/simulation time and having the required input files in place (EDDI files of the executed robots and log files generated during the execution), those files have to be provided within a environment. Therefore, copy all EDDI files (with the *.ddi* file extension) into a single directory and copy the log files generated by each ConSert monitor in another separate directory. In order to start the EDDI ConSert Visualiser, open a terminal of your choice (e.g. Command Prompt or PowerShell on Windows or Terminal on Linux), change current directory to the one where the EDDIVisualiser.jar is located and run following command (JRE >= 11 is required):

```
java -jar EDDIVisualizer.jar
```

When starting the visualiser, two selection dialogs are popping up for the user to interact with (see Figure 3.3). First, the user has to either choose the directory where the EDDI files are located, or choose an XML file containing a saved graph model from a previous session, as mentioned above. Afterwards, in the next selection dialog, the user has to choose the directory where the respective log files are located. Confirming the second selection, the visualiser will start and automatically layout the graph vertically. As mentioned above, the layout can be changed by the user manually anytime, but has to be saved to an XML file to load the same layouted graph upon future tool startups.


Select '.xml File' or 'DDI Directory' to load the graph

Look In: Example from Presentation

consert_ddis
consert_logs
EDDIVisualizer.jar
example.xml

File Name: consert_ddis
Files of Type: All Files

Select Cancel

Select the directory that contains the .log files

Look In: Example from Presentation

consert_ddis
consert_logs

Folder name: _components\eddi_visualizer\Example from Presentation\consert_logs
Files of Type: All Files

Select Cancel

Figure 3.3 Selection dialogs at visualiser startup

4. Attack Tree Visualisation

In an attack tree, the root is the ultimate goal of the attacker and the leaves are different ways (=attacks) to achieve that goal. Between the goal and the leaves, one can find sub-goals, which describe intermediate achievements of the attacker that bring them closer to their goal.

The best way for a human to receive all that information is to see the actual tree form of an attack tree. For that reason, we developed an attack tree visualisation tool. It utilises the Thymeleaf JAVA template engine, gets as input a series of nodes and edges, and depicts them as a tree. Listing 1 includes the JAVA method that is responsible for creating the *nodes* and *edges* ArrayLists. As it can be seen, a new *node* and *edge* instance is created for every node of a given tree (currentNode in the listing). Both instances are inserted in the corresponding ArrayLists. Finally, the for statement in the end calls the same method recursively for every child of the current tree node. The *nodes* and *edges* ArrayLists are used to visualise the attack trees.

```java
// method to create the nodes and edges of a given Tree (=CanPrecedeNode)
public void createNodesEdges(CanPrecedeNode currentNode) {

    Node node = new Node();
    node.setId(currentNode.getId());
    node.setLabel(currentNode.getData());
    node.setTitle(currentNode.getData());
    node.setExtendedDescription(currentNode.getExtendedDescription());
    node.setWidthConstraint(100);
    node.setHeightConstraint(100);
    node.setBorderWidth(0);
    node.setFont("14px arial white");
    node.setShape("box");
    nodes.add(node);

    Edge edge = new Edge();
    edge.setFrom(currentNode.getParentId());
    edge.setTo(node.getId());
    edge.setColor("#0A9396");
    edge.setWidth(4);
    edges.add(edge);

    if (currentNode.getChildren().size() > 0) {
        for (int i = 0; i < currentNode.getChildren().size(); i++) {
            CanPrecedeNode currentChild = currentNode.getChildren().get(i);
            createNodesEdges(currentChild);
        }
    } else {
              // no children
    }
}
```

Listing 1: JAVA method for visualisation of an attack tree

Attack Tree Visualization

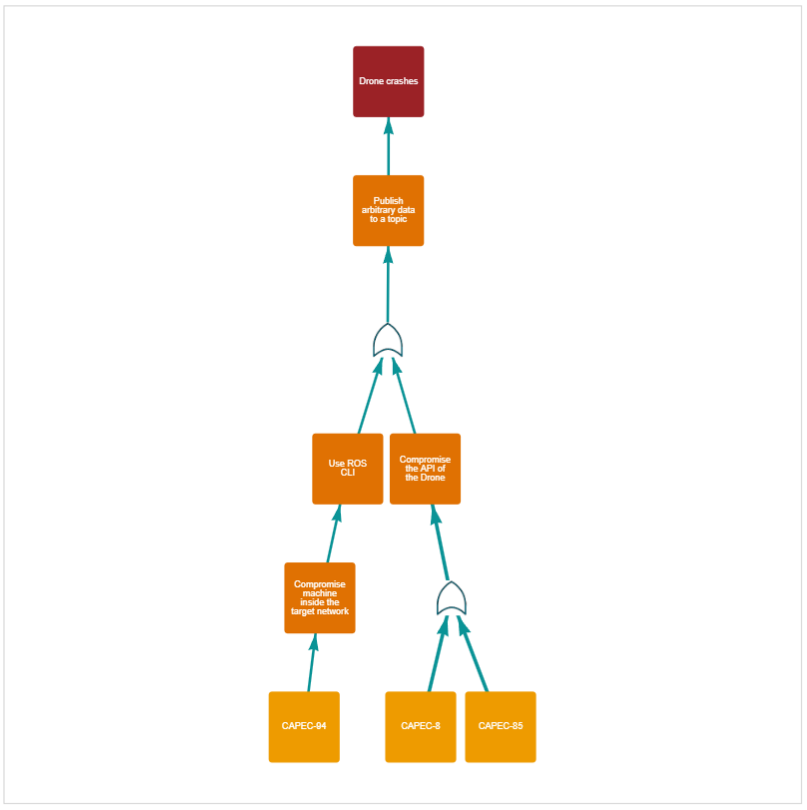
List of Attack Trees

Drone crashes
Insufficient design of mitigation actions
Communication interruption with Drone

Node Details

ID	Title	Extended description
303	OR Gate	Any single child is sufficient to lead to the parent state.

Attack Tree Visualization


Drone crashes
Publish arbitrary data to a topic
Use ROS CLI
Compromise the API of the Drone
Compromise machine inside the target network
CAPEC-94
CAPEC-8
CAPEC-85

List of Attack Trees

Drone crashes
Insufficient design of mitigation actions
Communication interruption with Drone

Node Details

ID	Title	Extended description
303	OR Gate	Any single child is sufficient to lead to the parent state.

Figure 4 Visualisation page of the attack trees (incl. combined and magnified views)

Figure 4 depicts the page for the visualisation of the attack trees. It consists of three main components: visualisation panel, list of attack trees, and node details. The vitalization panel depicts the chosen tree from the list of attack trees. The depicted tree is one of the potential Template Attack Trees related to the target system, based on its identified potential attacks. Three colors are used for the actual visualisation of an attack tree. The ultimate goal of the attacker (root of the tree) is presented as a red square. Light orange is given to known attacks, documented in the CAPEC repository with a specific CAPEC-ID. They are usually find at the leaves of the attack tree. Finally, Dark orange are all the intermediate sub-goals, security states of the target system, after a set of attacks.

The Listbox at the right side of the page, includes the whole list of the selected potential Template Attack Trees. The description of their ultimate goal is mentioned. By clicking any of these goals, the corresponding attack tree is depicted at the vitalization panel. Since a large number of attack trees can be related to a given target system, a Listbox menu allows the depiction of one tree at a time.

Moreover, the user can select a node of a tree and get additional information about that node. The Node Details component, shows additional information for any tree node that is selected. For now the title and extended description are given. More information will be added in the future, such as mitigation actions.

Attack Tree Visualization

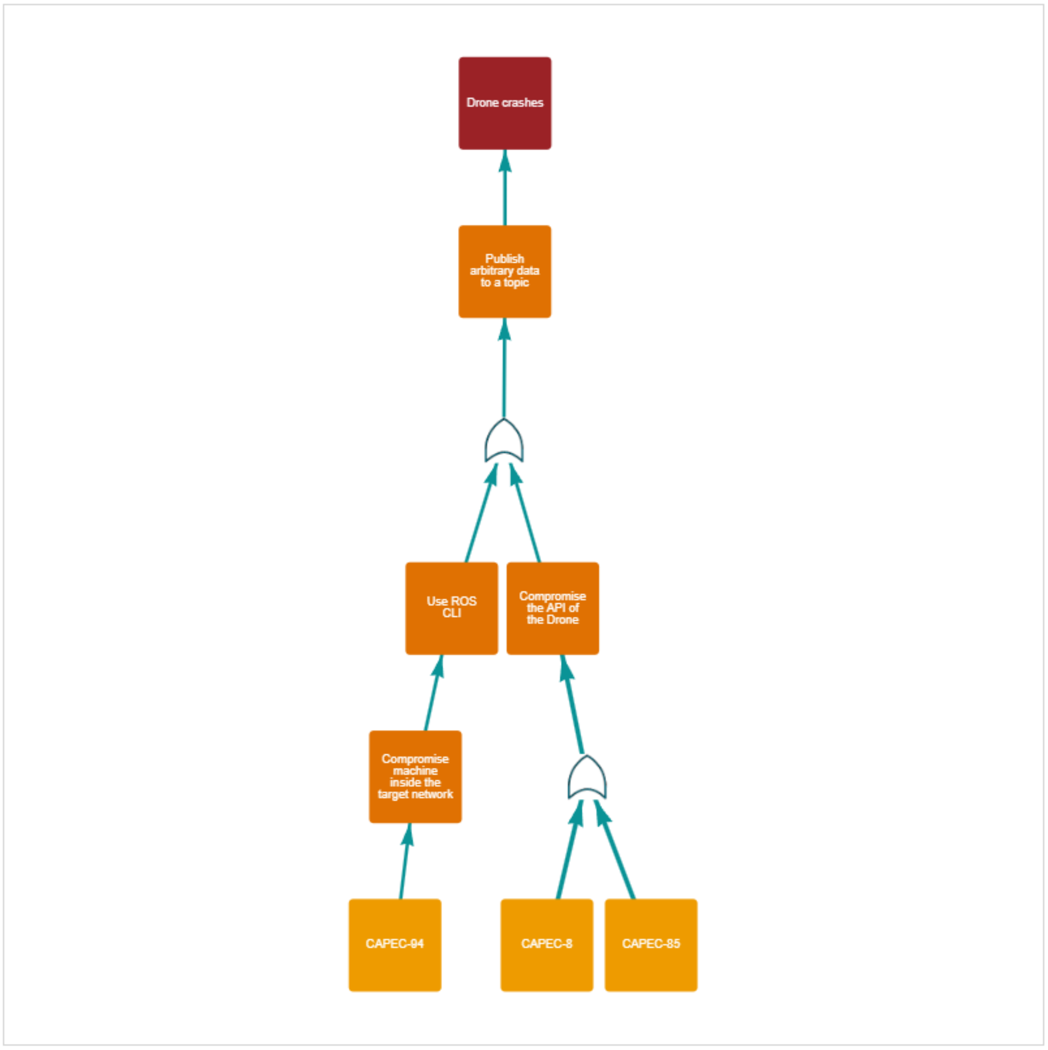

Drone crashes
Publish arbitrary data to a topic
Use ROS CLI
Compromise the API of the Drone
Compromise machine inside the target network
CAPEC-94
CAPEC-8
CAPEC-85

Figure 5 Sample of attack tree visualisation

Let's have a closer look to the tree included in the visualisation panel. The tree has three leaves, each for known attacks with ids 94, 8, and 85. Attack with CAPEC-ID 94 could allow an attacker to compromise a machine in a target network. The attacker, then, could find themselves at the state where they can use the ROS command line tool from that compromised machine. Any of the two other attacks (CAPEC-IDs 8 and 85) can lead to the security state where the API of one of the Drones is compromised. This state or the one besides (use ROS command line tool) can lead to the state where the attacker can publish arbitrary data to a ROS topic. Finally, this state lead to the ultimate goal, which is the compromised Drone to crash.

5. SMILE

As described in D4.5, SMILE stands for Statistical Model-agnostic Interpretability with Local, and it uses Empirical Cumulative Distribution Function (ECDF)-based statistical distance measures such as Wasserstein, Anderson-Darling, and Cramer-von Mises. In this section, we will discuss 1) How SMILE can be used for runtime explainability evaluation of ML components, 2) How we evaluate the performance of SMILE itself, and 3) A discussion on the capabilities and limitations of SMILE in this project. It should be noted that the idea of SMILE has been submitted to IEEE Software and it is under review.

Publication
Aslansefat, K., Hashemian, M., Walker, M., Akram, M. N., Sorokos, I., & Papadopoulos, Y. (2023). Explaining black boxes with a SMILE: Statistical Mode-agnostic Interpretability with Local Explanations. IEEE Software (Under review).

5.1 Runtime Explainability Evaluation of ML Components

The SMILE approach can be used for runtime explainability evaluation of the ML components. Focusing on KIOS use case, one can select a public dataset on Kaggle for Aerial Semantic Segmentation Drone Dataset . The main task is to detect person(s) in the image and initiate further actions based on the detection to make sure the UAV operates in a safe manner. In the design time different approaches can be used to evaluate the performance of ML component and with regards to semantic segmentation algorithms, mean Intersection over Union (mIoU) can be a method to assess the algorithm. The following figure shows A) original image sample, B) its ground truth and C) the UNET segmentation with mIoU measure.

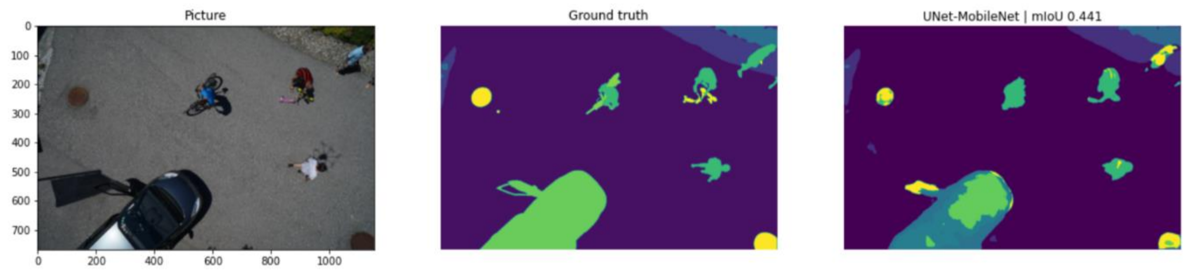


Figure 6 An example of measuring mIoU for an image segmentation task in design time (test sample is from Aerial Semantic Segmentation Drone Dataset)

In runtime evaluation, there's no established ground truth for gauging performance. We propose using the high-scoring super pixels from SMILE as a provisional ground truth and computing the mIoU between the SMILE mask and the machine learning segmentation outcomes for real-time assessment. It's important to note that generating SMILE explainability for an image may take several minutes, so it currently offers sporadic real-time explainability measurements. Additionally, SMILE is not equipped to directly support image segmentation algorithms. For instance, in our example, we adapted the machine learning model output to produce a binary classification - either a person is detected or not. The following figure shows A) a test sample is from Aerial Semantic Segmentation Drone Dataset, B) Detection Super pixel as stated in SMILE procedure in D4.5 and C) Creating an explainability mask.

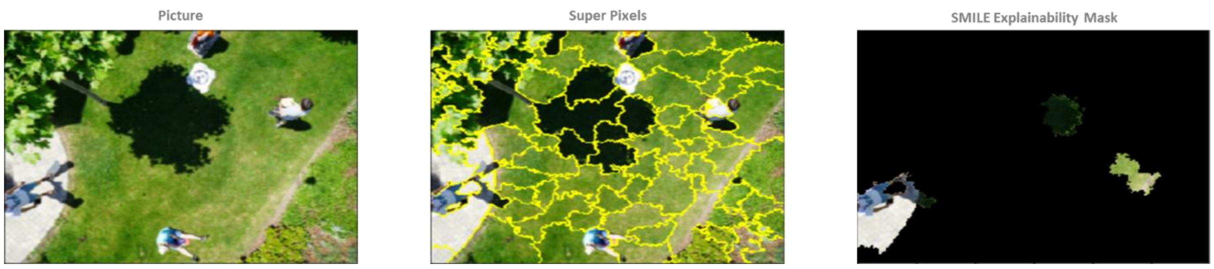


Figure 7 A) a test sample is from Aerial Semantic Segmentation Drone Dataset, B) Detection Super pixel as stated in SMILE procedure in D4.5 and C) Creating an explainability mask.

5.2 PERFORMANCE EVALUATION OF SMILE

We have performed a number of experiments to evaluate the performance of SMILE compared to common alternatives (LIME and SHAP and, for images, BayLIME). One way to evaluate an explanation is to compare the outcome to human intuition: a trustworthy explanation should be consistent with the explanations of humans who have proper expert knowledge and an understanding of the model. For example, if both an ML model and a doctor assess the same MRI scan image, both should highlight the same features of the image as the reason for their diagnosis.

In order to test this, we have reproduced the experiment used by [7]. Figure 8-(A) shows the comparison of human feature impact estimates versus those provided by LIME, SHAP, and SMILE. The experiment is based on a model that assigns a sickness score of 2 for fever and cough, 5 when only one symptom is present, or 0 otherwise; a third unrelated symptom, congestion, is also included. As can be seen in the figure, both SMILE and SHAP are consistent with the human explanations, providing positive values for fever and cough and zero for congestion. However, LIME's explanations were not consistent: LIME not only provided negative values for fever and cough but also assigned small negative values for congestion, which is an unrelated symptom.

Figure 8-(B) show the results of LIME, SHAP and SMILE for a XgBoost model trained on the COMPAS dataset. Although all models have the Fidelity score of 1, there is a disagreement between them which cannot be justified without having the ground truth. One way to obtain the ground truth is to make a model intentionally biased toward a specific feature and see how these methods respond. For example, in Figure 8-(C), a racist model is provided, and as can be seen, all three explainers have shown the race as the main feature. In SMILE, other features have very small values and that is because of the effect of local perturbations (the two circles in Figure 8-(B)). In practice, there is a trade-off between being robust to manipulations like adversarial attacks and focusing on specific feature explanations. The trade-off can be tuned using number of local perturbations in the algorithm.

Last but not least is Figure 8-(D), which shows an adversarial attack on explainability. As discussed in [8], specific models are designed to fool LIME and SHAP explanations towards an unrelated feature. The same approach is used to test SMILE as well. As illustrated in this figure, the LIME approach has been fooled by the attack while SHAP and SMILE both show a degree of robustness to the attack.

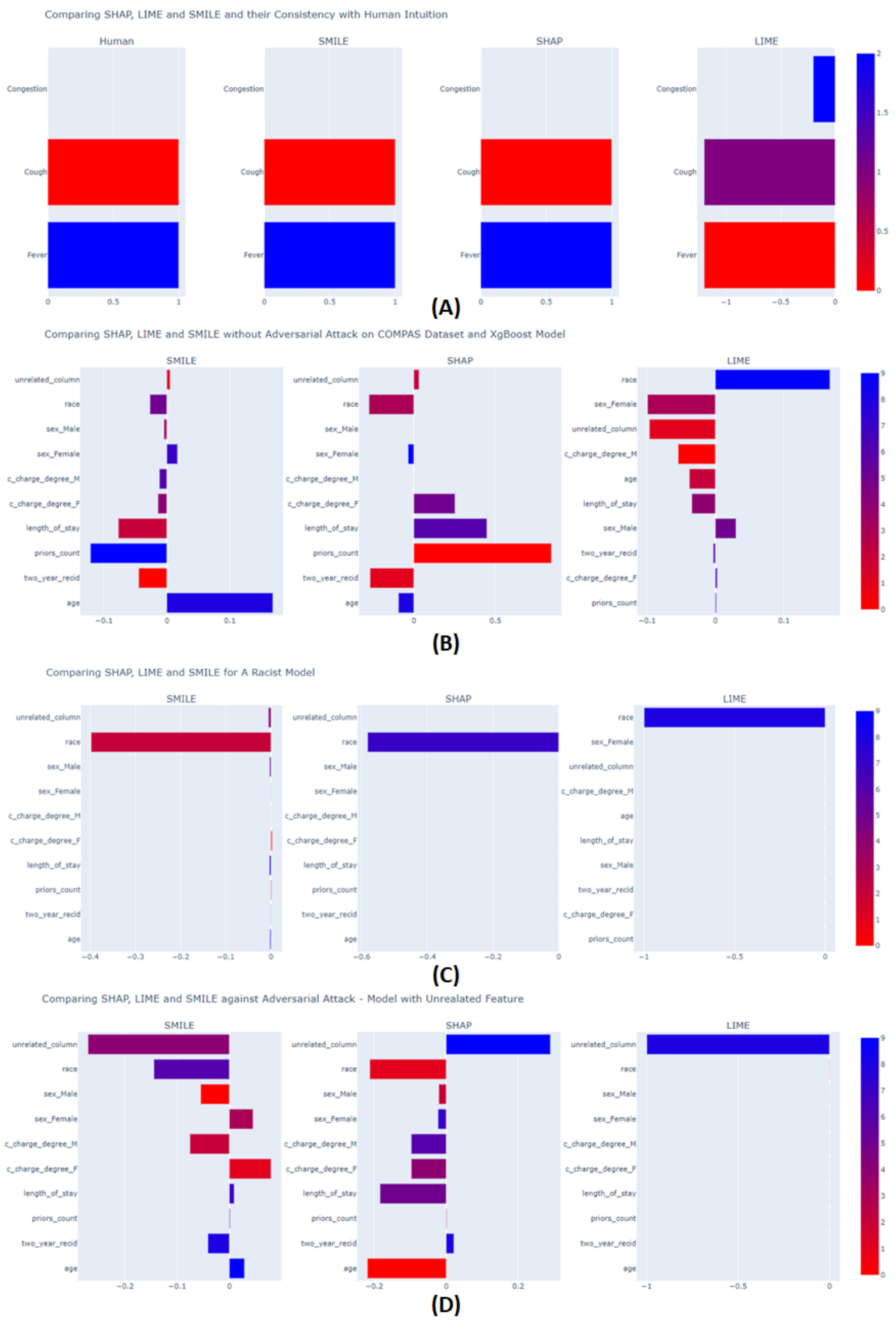


Figure 8 Comparing SHAP, LIME and SMILE (A) and their Consistency with Human Intuition, (B) COMPAS Dataset and XgBoost Model without Adversarial Attack, (C) against Adversarial Attack and a Racist Model, (D) against Adversarial Attack and a Model with Unrelated Feature

Explainability in images is particularly challenging compared to simple tabular data, because the features are all part of the image rather than numbers conveniently separated into discrete sets. Explaining the diagnosis of an illness from a scan image means breaking down the image into those parts that either support or contradict the given diagnosis, whereas the task of explaining a diagnosis made on the basis of variables representing discrete symptoms is much smaller in scope. Fortunately, because SMILE widens the range of values being inspected as part of the explanation, it frequently produces a more accurate, holistic explanation of a prediction about an image, e.g. by encapsulating more of those parts of the image that support the prediction and not including those parts of the image that are irrelevant or misleading.

Moreover, the current version of LIME uses a perturbation function that simply turns super-pixels on and off and then compares their corresponding perturbation vectors using cosine metrics. On the other hand, because SMILE compares the original and perturbed images, it not only makes it more accurate but also allows the algorithm to use different types of perturbations, like adding noise, etc.

Space constraints mean we cannot provide multiple examples here, but to demonstrate the capabilities of SMILE, we present here the results of a dog classification using the Inception_V3 model. Figure 9-(A) delineates a comparison between LIME, BayLIME and SMILE. BayLIME is another extension of LIME that focuses on improving the surrogate model by replacing its weighted linear regressor model with a Bayesian regressor model. The top-left image shown in this Figure is the original, which is given to the model. This example is that used in [9], in which four different settings -- non_Bay, which is similar to the original LIME approach; Bay_non_info_prior, Bay_info_prior, and BayesianRidge_inf_prior_fit_alpha -- are all considered. The results of these settings are shown in the central four images, and on the right the SMILE result is shown. As can be seen, SMILE largely outperforms both the original LIME and BayLIME, successfully identifying the dog as the most important feature while not identifying any irrelevant sections of the image as being important.

Figure 9-(B) shows the heat-map for SMILE explainability for the dog image classification considering cosine, Kuiper, Kolmogorov-Smirnov, Wasserstein, Anderson Darling, and Cramer-Von Mises distances respectively. The provided heat-map shows that ECD-based explainability can provide more accurate results.

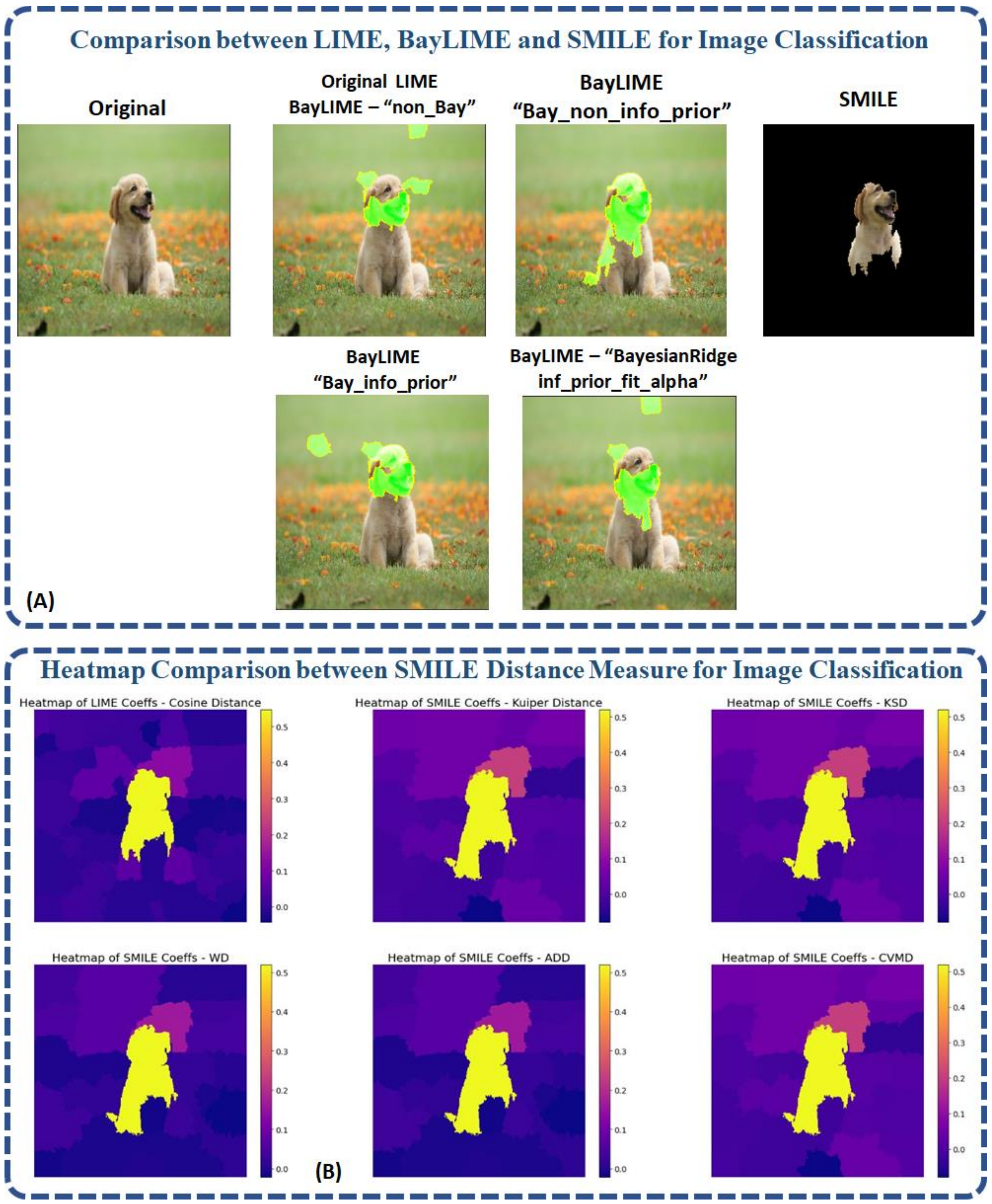


Figure 9 (A) Comparing SMILE with LIME and BayLIME for dog classification and its explainability for inceptionV3 model. (B) Heatmap Comparison between SMILE Distance Measure for Image Classification.}

5.3 DISCUSSION ON SMILE FOR EXPLAINABILITY

As SESAME forges ahead in revolutionizing multi-robot systems, putting emphasis on aspects like safety, security, efficiency, and explainability, integrating SMILE into its framework could prove to be invaluable. SMILE's model-agnostic nature and capability to analyse various forms of data make it an ideal candidate for enhancing the explainability aspect of the multi-robot systems developed through SESAME's innovations, particularly in conjunction with the Executable Digital Dependability Identities (EDDI).

EDDI, in SESAME, focuses on ensuring the dependability of multi-robot systems by carrying verifiable models that encompass safety and security hazards, their causes, effects, and possible corrective actions. SMILE's capabilities to produce reliable and robust local explanations for diverse data types, including images and text, could be seamlessly integrated into EDDI's framework. Through this integration, stakeholders could obtain rich insights into how the multi-robot systems are making decisions, the underlying factors that contribute to the systems' behaviour, and the interdependencies among various elements within the system. Moreover, the safety and security analyses within EDDI could be supplemented with explanations generated by SMILE, facilitating a deeper understanding of the root causes of potential hazards and the effectiveness of the mitigation strategies deployed by the robots.

A key area where SMILE's integration can further enhance SESAME is robustness. SMILE's ability to analyse a larger portion of input data can potentially provide SESAME with a more comprehensive understanding of ML components and their behaviour in multi-robot systems.

Given that SESAME places emphasis on efficiency and productivity, it is crucial to also address SMILE's computational complexity. Strategies like optimizing calculations by offloading them to GPUs or employing approximations such as Sinkhorn distance, a faster approximation of the Wasserstein distance that SMILE uses, can be considered. This will align SMILE with SESAME's real-time requirements and ensure that the added computational complexity does not become a bottleneck in the system.

In SESAME's quest for advancements in multi-robot systems, the need for explainability is paramount to foster trust in the deployed systems. Incorporating SMILE as an explainability tool into the SESAME framework can be a significant leap towards achieving transparency, safety, and security in the deployment of multi-robot systems. Through rigorous testing and validation across SESAME's various use cases, such as those focusing on robustness, productivity, and security, the benefits of integrating SMILE can be empirically assessed, and its contribution to the trustworthiness and dependability of multi-robot systems can be quantified.

5.4 SMILE Code Availability

Regarding the research reproducibility, codes and functions supporting this paper are published online at GitHub: https://github.com/Dependable-Intelligent-Systems-Lab/xwhy.

6. Summary

In this deliverable, we present tools developed or improved upon in SESAME, with which EDDI state or behaviour can be explained during development or runtime. The tools discussed are going to be available as open-source from the SESAME project GitHub repository at https://github.com/sesame-project/explainable_eddis, alongside documentation and examples.

7. REFERENCES

[1] S. Thellman and T. Ziemke, “The perceptual belief problem: Why explainability is a tough challenge in social robotics,” *ACM Transactions on Human-Robot Interaction (THRI),* vol. 10, no. 3, pp. 1-5, 2021.

[2] R. Setchi, M. Dehkordi and J. Khan, “Explainable robotics in human-robot interactions,” *Procedia Computer Science,* vol. 176, pp. 3057-3066, 2020.

[3] G. Papagni and S. Koeszegi, “Understandable and trustworthy explainable robots: a sensemaking perspective,” *Paladyn, Journal of Behavioural Robotics,* vol. 12, no. 1, pp. 13-30, 2020.

[4] G. Papagni and S. Koeszegi, “Challenges and solutions for trustworthy explainable robots,” *Trust in Robots,* vol. 15, no. 57, 2022.

[5] S. Wachter, B. Mittelstadt and L. Floridi, “Transparent, explainable, and accountable robotics,” *Science Robotics,* vol. 2, no. 6, 2017.

[6] S. Anjomshoae, A. Najja, D. Calvaresi and K. Framling, “Explainable agents and robots: Results from a systematic literature review,” in *18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019),* Montreal, Canada, 2019.

[7] S. Lundberg and S. Lee, “A unified approach to interpreting model predictions,” *Advances in neural information processing systems.*

[8] D. Slack, S. Hilgrad, E. Jia, S. Singh and H. Lakkaraju, “Fooling LIME and SHAP: Adversarial attacks on post-hoc explanation methods,” *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society,* pp. 180-186, 2020.

[9] X. Zhao, W. Huang, X. Huang, B. Robu and D. Flynn, “Baylime: Bayesian local interpretable model-agnostic explanations,” *Uncertainty in artificial intelligence,* pp. 887-896, 2021.

[10] K. Aslansefat, I. Sorokos, D. Whiting, R. Kolagari and Y. Papadopoulos, “SafeML: Safety Monitoring of Machine Learning Classifiers through Statistical Difference Measure.,” in *International Symposium on Model-Based Safety and Assurance (IMBSA) 2020,* Lisbon, Portugal, 2020.

[11] M. De Graaf, A. Dragan, B. Malle and T. Ziemke, “Introduction to the special issue on explainable robotic systems,” *ACM Transactions on Human-Robot Interaction (THRI),* vol. 10, no. 3, pp. 1-4, 2021.

[12] F. Sado, C. Loo, W. Liew, M. Kerzel and S. Wermter, “Explainable Goal-driven Agents and Robots - A Comprehensive Review,” *ACM Computing Surveys,* vol. 55, no. 10, pp. 1-41, 2023.