# Joint analysis of safety and security

## Identifying interactions between safety and security, from model level to language level

Lada Egorova
CTSYS team, LCIS

**Supervised by:** Oum-El-Kheir Aktouf
**School Tutor:** Om Essaad Slama
**The chairman of the jury:** Yann Kieffer

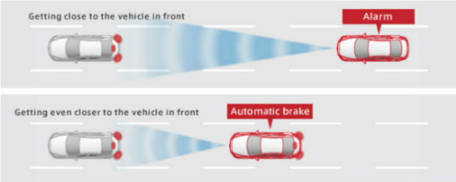
24 June 2025

# Table of Contents

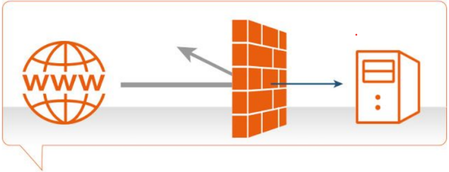
# Table of Contents

# Problem context

**Auto-braking** - Safety feature
In critical situation, system tries to send a message to stop the vehicle




**Firewall** - Security feature
Firewall blocks the unexpected messages, which can lead to a danger



# Problem context

Finding interactions can be difficult:

- Requires manual analysis
- Developers of safety and security features usually don't interact with each other
- The interactions are hard to predict

# Existing methods

| Approach Type | Method / Tool | Application / Notes |
|---|---|---|
| Game-Theoretical | Game theory | Used for co-verification and cyberspace conflict modeling |
| Tree-Level Models | Fear Event Trees, Multi-Level Models, Fault Trees | Analyze origins of safety-security interactions |
| Knowledge Graphs | Dual-security analysis | Applied to ICS for enhanced threat modeling and traceability |
| Formal Methods | Event-B framework | Formal guarantees; limited scalability |
| Matrix-Based | Inspired by Analytical Network Process (ANP) | Combined risk assessment in cyber-physical systems |
| AI / ML Based | Large Language Models (LLMs) | Has potential for safety-security management |
| Bowtie Analysis | Bowtie diagrams | Visual risk assessments in Industrial Control Systems |
| Other Tools | Semi-Automatic Tools, Standards & Co-Engineering, Bowtie Analysis | Used for specific cases |

# LCIS lab

LCIS lab was founded in October 1996 and now brings together:

- 26 teacher-researchers
- 20 to 30 Ph.D. students
- postdoctoral researchers and research collaborators
- around 20 interns

ORSYS

CTSYS

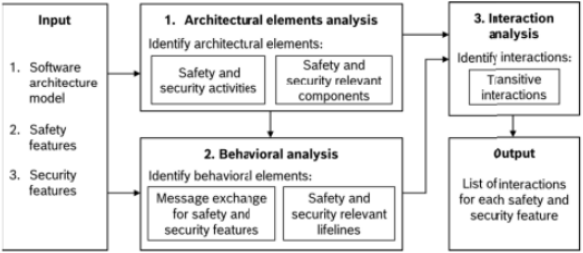
CO4SYS

LCIS
Laboratoire de Conception et d'Intégration des Systèmes

# Problem context: FIISS

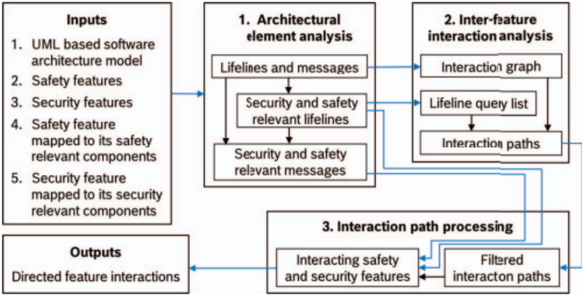
In LCIS lab, FIISS (and its improved version, I-FASST), were developed:

- The tool takes a UML sequence diagram as an XML file as an input
- Returns a list of existing interactions between safety and security features
- Works iteratively for each feature



# Problem context: I-FASST

- Has more inputs
- Filters paths with interaction chains
- Runs in one go



# Problem context: possible contributions

| **Solution based on more general model (e.g., knowledge graphs)** | **Runtime verification research** |
|---|---|
| • XML files are not easy to edit in runtime<br>• The general model solution does not depend on concrete realization<br>• A general model may allow us to store different types of information | • To let engineers see the consequences of their actions directly on the spot<br>• To enrich our knowledge of possible threats |

# Table of Contents

# Specifications: expected results

- A tool for converting an UML diagram to a general model
- A tool for performing an I-FASST analysis on a general model
- I-FASST analysis on a general model should be faster than on an XML diagram
- The possible existing research into runtime verification has been identified

# Table of Contents

# Comparison of potential general models

| Approach | Key Characteristics | Notes / Applications |
|---|---|---|
| Knowledge Graphs | Graph-based structure; uses link prediction; measurable with metrics like Mean Rank, Precision, Recall | Applied in cybersecurity for threat modeling and prediction; interpretable; useful for structured knowledge representation |
| Vector Databases | Store data as numerical vectors for fast similarity search | Faster but may lose relationship semantics; not ideal when link fidelity is crucial |
| Large Language Models (LLMs) | Neural networks trained on vast textual data; excels at NLP tasks | Still less reliable than KGs for factual accuracy and long-tail entities; promising for unstructured data extraction |
| Other Approaches | Concept maps, conceptual graphs, semantic networks | Often based on or derived from knowledge graphs; more task-specific and less generalizable |

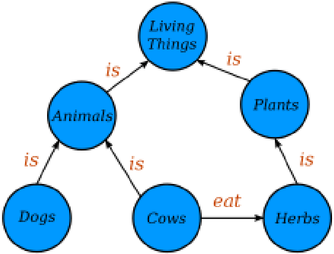
LCIS Laboratoire de Conception et d'Intégration des Systèmes

# Choosing a database

| DB name | Apache2 license | Programming language | Description |
|---|---|---|---|
| ArangoDB | + | C++, JavaScript, .NET, Java, Python, Node.js, PHP, Scala, Go, Ruby, Elixir | NoSQL DB, ArangoDB Query Language, high scalability |
| Amazon Neptune | + | Not disclosed, used through queries | Easy to migrate data from AWS |
| Azure Cosmos DB | - | Not disclosed, used through queries | Multi-modal DB, Apache Gremlin language |
| JanusGraph | + | Java | High scalability |
| NebulaGraph | + | C++, Go, Java, Python | High scalability |
| Neo4j | + | Java, .NET, JavaScript, Python, Go, Ruby, PHP, R, Erlang/Elixir, C/C++, Clojure, Perl, Haskell | Web and desktop versions, Cypher query language, the biggest community |
| Microsoft SQL Server 2017 | - | SQL/T-SQL, R, Python | SQL support |
| Oracle | - | PGQL, Java, Python | PGQL query language |
| OrientDB | + | Java | Both a graph DB and a NoSQL DB |
| TerminusDB | + | Prolog, Rust, Python, JSON-LD | Document-oriented KG |
| TigerGraph | - | C++ | Parallel, super-fast |

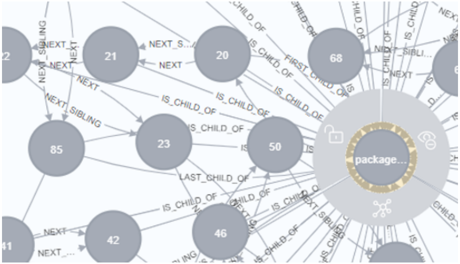
# Knowledge graph

- A structure to store the data
- Has nodes and relationships between them
- Can store any type of information
- Human-readable



# Choosing an approach

**APOC tool**




**LLMGraphTransformer tool**

- Shows good results
- However, it is not deterministic

# Table of Contents

# Diagram conversion

## Remark

We call objects in XML file as *"entities"* with *"attributes"*, and their representation in a knowledge graph as *"nodes"* with *"properties"*

- Iteratively go through the XML diagram and transform each entity and relationship
- Use a predefined set of rules

## Examples

The node's label in the KG is equal to the entity's attribute "xmi:type"

## Examples

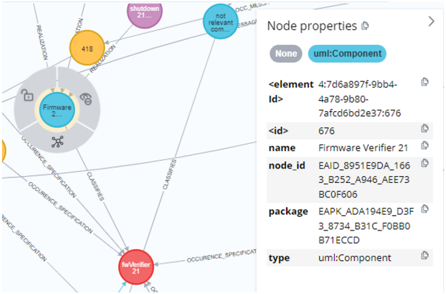
A relationship has type "OCC_MESSAGE", if it is represented in the diagram as an entity with type "uml:Message". It is an edge in the KG between two nodes of type "uml:OccurenceSpecification"

# Diagram conversion: example

From an XML file

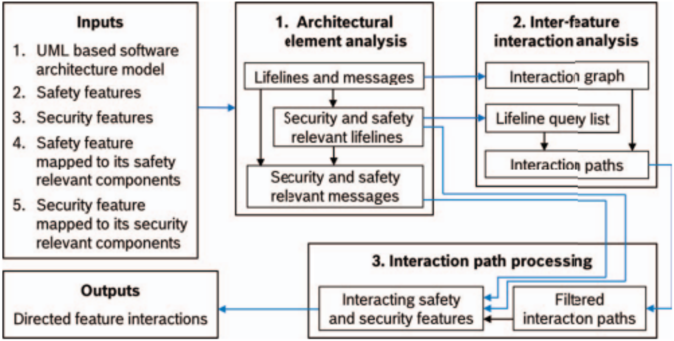
```
<packagedElement xmi:type="uml:Component"
xmi:id="EAID_8951E9DA_1663_B252_A946_AEE73BC0F606"
name="Firmware Verifier 21" visibility="public"
isIndirectlyInstantiated="true"/>
```

To a knowledge graph




LCIS Laboratoire de Conception et d'Intégration des Systèmes

# I-FASST method




## Lifeline

A structural element in a sequence diagram that classifies a component. In our case, 1 feature has 1 component and 1 lifeline

# Remarks

**1.**

We say that two lifelines have the same relevance, if both of them are safety relevant, or both of them are security relevant.

**2.**

We say that two lifelines have different relevance, if the first lifeline is safety relevant and the second lifeline is security relevant, and vice versa.

**3.**

We say that a lifeline is not relevant, if it is neither safety nor security relevant.

# A. Architectural element analysis — 1. Safety and security relevant activities and components

We can skip this step because we have it as input, however, it is possible to find them manually:

| | |
|---|---|
| **1.** Find all nodes in the graph that have type `"uml:Activity"` | **2.** For each node, take its property `"package"` and define if it is a safety or security package |
| **3.** For each activity, find connected nodes and check if they have type `"uml:Component"` | **4.** Map each feature to its safety or security relevant components |

# A. Architectural element analysis — 2. Message exchanges

## Cypher language

Cypher is a declarative graph query language that is similar to SQL in terms of graphs

## Query

MATCH (n) - [r] $\rightarrow$ (m) WHERE r:MESSAGE RETURN n, r, m

(n), (m) is a template for nodes, [r] for a relationship
This query returns a list of tuples (n, r, m) where there is an edge of type MESSAGE between n and m

# B. Inter-feature interaction analysis — 1. Interaction graph

| **Custom query** | **Integrated libraries** | **Separate graph** |
| --- | --- | --- |
| Can skip this step<br>because we already have a graph | Need to integrate | Need to build |
| 50 seconds | 21 seconds | 2 seconds |
| Custom query usage | Dijkstra's source-target shortest path | `NetworkX` library<br>(the one that was used in I-FASST) |

Here we form a list of queries (srcLifeline, dstLifeline) with different relevance

### List of queries

(fwVerifier, fwLoader),
(fwLoader, fwVerifier),
(fwVerifier 2, fwLoader),
(fwLoader, fwVerifier 2),
...

Applying queries to the graph to find paths:

**Query**

```
MATCH p = (n) -[*..max_length]→ (m) // to configure max path length
WHERE n.node_id = $start_value AND m.node_id = $end_value
AND ALL(r IN relationships(p) WHERE r.diagram_id =
head(relationships(p)).diagram_id) // to search in one diagram
WITH p, nodes(p) AS all_nodes
RETURN p, all_nodes
LIMIT 1 // can have multiple paths
```

# C. Interaction path processing

1. Filtering interaction paths

**Remark**

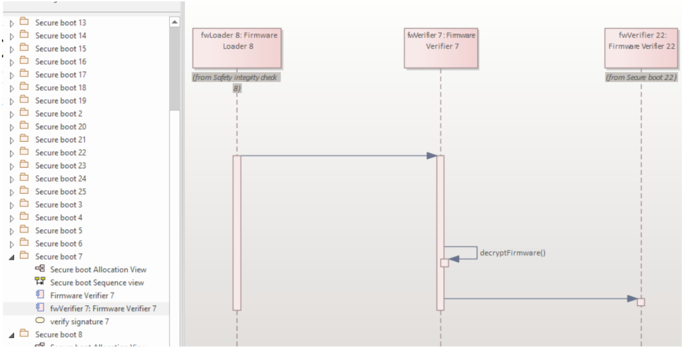
Interaction chains — interaction paths containing safety or/and security intermediate nodes

We need to avoid these paths, so we filter the set and for each path check if it has interaction chains

2. Interacting features

The rest "good" paths are returned as a function output

# The unit-under-test: a diagram created in Enterprise Architect




The final version contains around 1,300 nodes and 1,400 relationships, 50 safety and 25 security features

# Comparison metrics: execution time and the information obtained

## I-FASST on KG

| path |
|---|
| Secure boot 21 → Safety integrity check 44 |
| Secure boot 10 → Safety integrity check 49 |
| Secure boot 11 → Safety integrity check 40 |
| Secure boot 12 → Safety integrity check 26 |
| Secure boot 18 → NRF 6 → NRF 4 → Safety i. c. 16 |
| Secure boot 20 → Safety integrity check 51 |
| Secure boot 3 → NRF 2 → Safety integrity check 27 |
| Secure boot 6 → Safety i. c. 47 |
| Secure boot of automotive firmware → Safety i. c. |
| Safety integrity check 32 → Secure boot 24 |
| Safety integrity check 15 → Secure boot 22 |
| Safety integrity check 6 → Secure boot 23 |
| Safety integrity check 8 → Secure boot 7 |
| Safety integrity check 44 → Secure boot 21 |
| Safety integrity check 50 → Secure boot 9 |
| Secure boot 4 → Safety integrity check 2 |

Time: 1.87 seconds

## I-FASST on XML

| Feature 1 | Feature 2 |
|---|---|
| Secure boot 21 | Safety integrity check 44 |
| Secure boot 10 | Safety integrity check 49 |
| Secure boot 11 | Safety integrity check 40 |
| Secure boot 12 | Safety integrity check 26 |
| Secure boot 18 | Safety integrity check 16 |
| Secure boot 20 | Safety integrity check 51 |
| Secure boot 3 | Safety integrity check 27 |
| Secure boot 6 | Safety integrity check 47 |
| Secure boot 1 | Safety integrity check 1 |
| Safety integrity check 32 | Secure boot 24 |
| Safety integrity check 15 | Secure boot 22 |
| Safety integrity check 6 | Secure boot 23 |
| Safety integrity check 8 | Secure boot 7 |
| Safety integrity check 44 | Secure boot 21 |
| Safety integrity check 50 | Secure boot 9 |

Time: 2.09 seconds

# Comparison metrics: user experience

| | **I-FASST on KG** | **I-FASST on XML** |
|---|---|---|
| Visual representation | + | - |
| Can be stored in a cloud | + | - |
| Changing in runtime | Easier | Harder |
| Complex queries | Using Cypher | Should be resolved manually |
| Runtime verification | Bigger potential because of the above | Is possible, but harder |

LCIS
Laboratoire de Conception et d'Intégration des Systèmes

# Table of Contents

# Runtime verification: Introduction

**Definition**

**Runtime verification** is a computing system analysis and execution approach based on extracting information from a running system and using it to detect and possibly react to observed behaviors (Wikipedia).

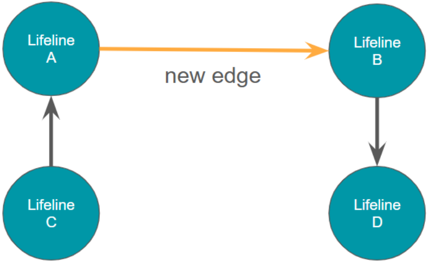
Criterias under analysis depend on current system

- E.g., out-of-bounds sensor values for safety components
- E.g., confidentiality for security components

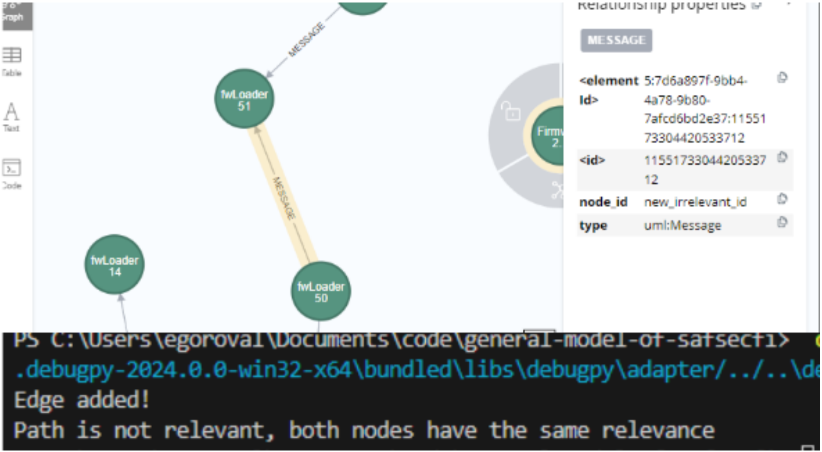
LCIS Laboratoire de Conception et d'Intégration des Systèmes

# Runtime verification, scenario 1 — a new edge is added

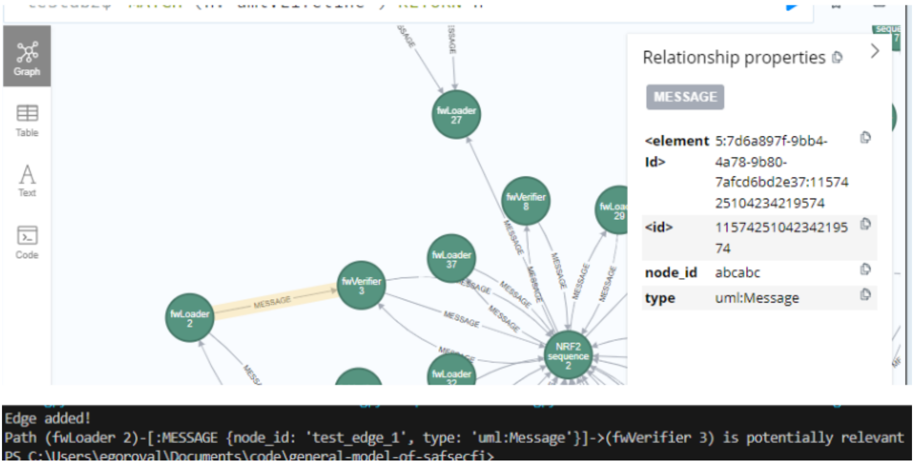
| **Case 1** | **Case 2** | **Case 3** |
|---|---|---|
| The source and the target nodes have the same relevance | The nodes have different relevance | At least one of the nodes is not relevant:<br>– The source node<br>– The target node<br>– Both nodes |



# Runtime verification, scenario 1 — the same relevance



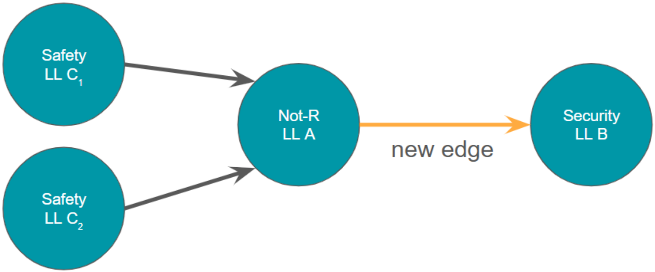
# Runtime verification, scenario 1 — different relevance




LCIS Laboratoire de Conception et d'Intégration des Systèmes

# Runtime verification, scenario 1 — at least one lifeline is not relevant

For example, the source lifeline is not relevant, and the destination lifeline is **security relevant**

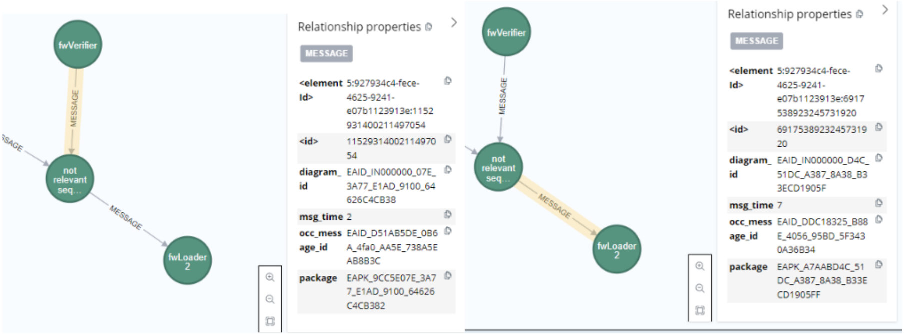



- Path 1: $C_1 \to A \to B$
- Path 2: $C_2 \to A \to B$

# Runtime verification, scenario 2 — timing aspect


- Lifeline B has to be non-relevant to avoid interaction chains
- Search for paths that begin and/or end with a non-relevant lifeline
- Try to form the paths (A) $\rightarrow$ (B) $\rightarrow$ (C) with relation to relevance of A and C
- Timing of $t_1$ should be less than timing of $t_2$

# Runtime verification, scenario 2 — example




LCIS Laboratoire de Conception et d'Intégration des Systèmes

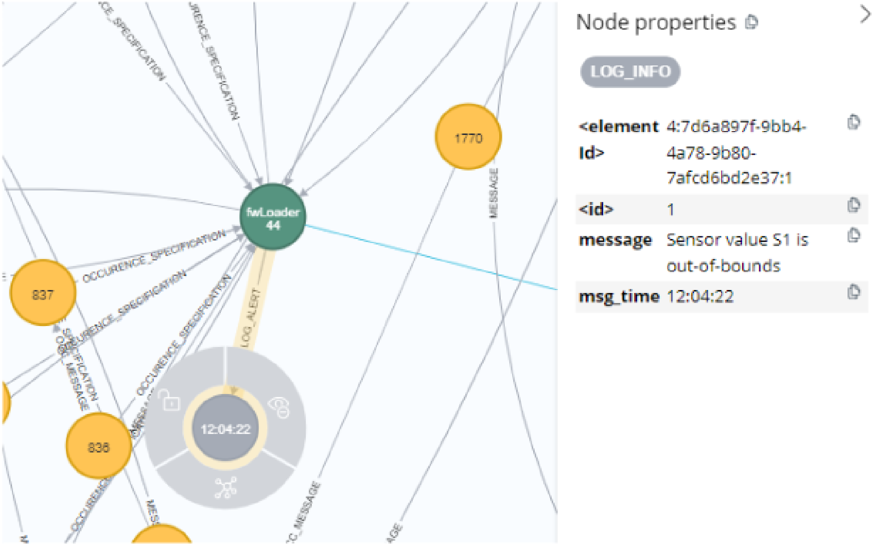
# Logs aspect, scenario 1 — from KG to logs

- For example, error message from a security feature to itself
- The message that is marked as unexpected

**Logs output**

Alert! Anomaly Sensor value S1 is out-of-bounds detected in EAID_LL000000_8428_2537_A50F_802085E313F8

# Logs aspect, scenario 2 — from logs to KG



# Table of Contents

# Results obtained with relation to specifications

| Specification | Status | Comment |
|---|---|---|
| A tool for converting a UML diagram to a KG | Met | Developed |
| A tool for performing an I-FASST analysis on KG | Met | Developed |
| I-FASST analysis on KG<br>is faster than on the XML diagram | Partially Met | The speeds are relatively equal |
| The possible existing research into<br>runtime verification has been identified | Met | 2 general + 2 logs scenarios |

# Future work

The topic still offers a vast field of research, and here are many ideas to be inspired by, for example:

- A study of the use of test case generation and its application during runtime to improve the reliability of a model
- User interface to improve the user experience
- Improvement of the I-FASST method itself
- etc...

The code is available on GitLab:

LCIS
Laboratoire de Conception et d'Intégration des Systèmes

# Thank you for your attention and to LCIS for this opportunity!


LCIS
Laboratoire de Conception et d'Intégration des Systèmes