Graduation project report

Grenoble INP - Esisar 2024/2025

Project title
Joint analysis of safety and security: identifying interactions between safety and security, from model level to language level

Name and address of company
LCIS laboratory **50 rue Barthélémy de Laffemas CS 10054** **26902 Valence Cedex 09 - France**

Name and address of student
Lada EGOROVA **6 rue Chauffour,** **26000 Valence**

Dates of internship	**27 January 2025 — 27 June 2025**
Speciality	**IMESS**
Company tutor	**Oum-El-Kheir Aktouf**
ESISAR tutor	**Om Essaad Slama**

Graduation project report

Grenoble INP - Esisar 2024/2025

Key words: safety, security, cybersecurity, joint analysis, knowledge graphs, automotive systems

Mots-clés: sûreté, sécurité, cybersécurité, analyse conjointe, graphes de connaissances, systèmes automobiles

Summary: The goal of this work is to continue previous research done in the LCIS laboratory, which was dedicated to finding a method for detecting potential safety and security interaction paths in automotive systems using UML diagrams. We aim to improve this method by answering whether it can be implemented on a more general model such as knowledge graphs and comparing the paths obtained from both methods.

The second part of the project was a study of runtime verification opportunities and their applicability to the current I-FASST system. Potential interaction scenarios were identified: the first scenario involves adding a new message exchange to the system and tracking its effect on the system; the second scenario involves searching for potential interactions between sequence diagrams.

The forensic aspect was also taken into account, and scenarios were discovered for using log data together with knowledge graphs. For example, paths obtained in I-FASST can be filtered based on log data and a resulting table can be printed with alert messages, or those paths can simply be removed from the results.

Contents

1 Introduction

The amount of automotive systems is increasing every year with enormous growth. They frequently interact with the environment, making them vulnerable to safety risks. On the other hand, security is also crucial for these systems, especially when it comes to health and life. Therefore, automotive systems rely on both safety constraints and security measures in many cases.

Consider an example of a vehicle with two features: an auto-braking system and a firewall. The former is safety-relevant, because it helps prevent car accidents, while the latter is security-relevant, as it provides cybersecurity protection against hacker attacks. If there is a danger, the auto-braking system sends an extra signal to the brakes, causing the vehicle to stop. However, if this signal is mistaken for a hacker attack by the firewall, it may be blocked, leading to a critical situation that cannot be prevented. This is why it is important to detect and resolve such interactions.

Scientists from the LCIS laboratory in Grenoble INP developed a tool called FIISS[1] and its improved version, I-FASST[2], to detect these types of interaction based on the project architecture in a UML diagram. This tool showed good results during testing and opened a field for further research.

Our goal in this project is to find ways to generalize tool applications and to apply them to possible scenarios.

1.1 Context and strategic issues of the project within the company

The work is done at the LCIS (Laboratory of Systems and Communication Engineering) in Valence, France. The lab was founded in October 1996 and now brings together 26 teacher-researchers, 20 to 30 Ph.D. students, postdoctoral researchers, and research collaborators. Every year, around 20 international students come to the laboratory for their internships.

The lab focuses on embedded systems and communication, bringing together researchers from all over the world. There are three main departments: ORSYS (Communicating Wireless RF Systems), CO4SYS (Coordination, Cooperation, and Control of Complex Systems) and CTSYS (Safety and Security of Embedded and Distributed Systems). This work is done in the CTSYS department.

The LCIS laboratory organizes an annual Scientific Day and a PhD Day to share research and look towards the future. Every year, the goals of the laboratory are announced.

In 2024, the focus was on sustainable development – the concept of developing in a way that meets the current needs without harming future generations. The current project aims to enhance the analysis of safety and security interactions, which will assist companies in detecting potential problems more quickly. This will reduce the need for resources in the future compared to XML implementation.

1.2 Specifications

There is a list of expected results from the solution:

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

2 Framing and planning, methodology

2.1 List of tasks

Our plan is as follows:

- Study existing approaches and formulate the problem;
- Find a general model that meets our requirements;
- Convert a UML diagram to the general model;
- Perform an I-FASST analysis;
- Find comparison metrics and compare the results of UML analysis with the one on the general model;
- Study existing researches about runtime verification;
- Try to apply some interesting approaches and see the results.

Work part	Estimated duration	Time period
State-of-the-art research	1 month	27th of Jan - 26th of Feb
Developing a solution	3 months	27th of Feb - 26th of May
Writing a final report	1 month	27th of May - 13th of June

3 Problem formulation

Safety-security interactions in automotive systems can lead to conflicts that are crucial to the general functioning of the system and to human health and safety. This section provides a brief overview of current research on safety and security issues and their perspectives.

3.1 Related work

The problem of solving safety-security interactions is well-known. Existing solutions can be categorized in various ways, such as by the type of approach to solving a problem, the application platform, the problem itself etc.

Different approaches

If we start talking about different approaches to solving a problem, one of them is game theory. The **game theoretical approach** can be applied to co-verification[3] and cyberspace conflicts modeling[4].

Some research is dedicated to the use of **tree-level models** for studying safety and security[5][6]. Megha Cuamara and her team use Fear Events Trees and a Multi-Level Model to analyze interactions related to safety and security[7].

There also exists the **Event-B framework**, which is a formal method for analyzing systems[8][9]. Formal methods provide a guarantee that a system will work correctly, but they have problems with scalability.

A matrix-based approach[10], inspired by Analytical Network Process (ANP), was applied to cyber-physical systems for combined risk assessments for safety and security purposes.

With growing popularity of machine learning and **LLMs in particular**, they were considered as a measure to safety and security[11].

Bowtie analysis is a method of visualizing risk assessments. An approach based on this was applied to industrial control systems in

the context of both safety and security, it means that not only safety is considered, but also security[12].

Type of applications

Another kind of classification may rely on the type of applications. In case of **cyber-physical systems**, the joint safety and security modeling was performed by Siwar Kriaa[13]. For CPS, there is also a research about safety-security risk assessment[14]. In **Industrial Control Systems**, the importance of connections leads to security vulnerabilities. They are the object of analysis within the context of conflicts between security and safety boundary conditions[15].

Type of model under analysis

The approaches and final evaluation of the method highly depend on the model being studied and how safety-security dependencies are presented.

The I-FASST method uses a list of UML diagrams, especially sequence and activity diagrams, as input.

Knowledge graphs, a popular tool for searching engines, were applied to ICS to ensure dual-security[16].

The researchers from Fraunhofer Institute for Mechatronic Systems Design[17] developed their own semi-automatic analysis tool based on **Model Sequence Diagrams**.

In case of automotive systems, the approach based on **relevant standards and process development**[18] was applied.

We summarized these approaches in the Table 1:

Table 1: Different approaches to safety-security interactions analysis

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
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
Knowledge Graphs	Dual-security analysis	Applied to ICS for enhanced threat modeling and traceability
Semi-Automatic Tools	Based on Model Sequence Diagrams	Developed by Fraunhofer Institute for semi-automated analysis
Standards & Co-Engineering	Based on process and regulatory standards	Focuses on integration of safety and security processes (especially in automotive)

Worth to mention

There is also a research about **automatically deriving test cases** from safety-security dependencies[19]. These results can be used as a potential by us for future work, for example, automatically generate tests during runtime verification.

3.2 Problem formulation

The I-FASST tool is based on UML diagrams, but they are specific to the system under analysis. If we analyze different systems, this can be challenging because we need to consider the concrete implementation and adapt our approach accordingly. Additionally, using UML diagrams can be slower than necessary. Moreover, it can be difficult to store different kinds of information in UML diagrams and, if there is a need of adding some new data in runtime, editing an XML file can lead to errors. Therefore, a more general approach would be beneficial.

Most studies do not focus on the use of generalized UML models for safety and security in automotive systems. In this paper, we will attempt to develop and implement a model of this type and compare its safety and security evaluation methods with those of classic UML representation.

3.3 Possible approaches

Vector databases

Vector databases are used for similar tasks to knowledge graphs, but differ in their underlying abstractions[20]. They store data as numerical vectors to improve the speed and quality of search. Sometimes this can lead to the loss of some important information, such as important relationships. This is critical for our goal of effectively finding safety-security interactions. We cannot afford to miss any links on a diagram.

Large Language Models

Large Language Models (LLMs) are special types of machine learning models used for natural language processing[21]. They are able to work with millions of parameters and vast amount of information. Kai Sun et al. tried to find an answer to a question, if one day LLMs will replace knowledge graphs[22]. The results showed that LLMs "are still far from being perfect in terms of their grasp of factual knowledge", especially for torso-to-tail entities (non-popular requests and long-represented data). The discussion from the popular internet blog[23] mentions accuracy, interpretability and human explainability as advantages of knowledge graphs over neural networks in general.

Knowledge graphs

Knowledge graphs are used in various fields, from searching services like Google to security tools. They help to aggregate data from pop-

ular databases to find common patterns and predict future threats, so-called **link prediction**[24]. They also use rank-based metrics (the Mean Rank, the Mean Reciprocal Rank, and Hits@N scores) to measure the quality of the embedding. For measuring link prediction quality they use specific metrics such as precision, recall, and F1-score (the useful model predictions).

Another approach is to model security threats on a software level[25]. Romy Fieblinger et al. combined it with Large Language Models(LLM) to extract data from unstructured Cyber Threat Intelligence (CTI)[26]. In the article "A review of knowledge graph application scenarios in cybersecurity" the authors provide an overview of current methods to concepts of building a cybersecurity knowledge graph[27]. To sum up, knowledge graphs show good results in this context.

Other approaches

There are also a lot of related methods, for example, concept maps[28], conceptual graphs[29], semantic networks, etc. However, these methods are relatively specific and vulnerable to a concrete task, and sometimes are based on knowledge graphs (concept maps, conceptual graphs).

Table 2: 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

Conclusion

Based on the Table 2, we have decided to choose knowledge graphs as a tool due to their ability to represent entities and relationships. They also have the possibility to apply link prediction which may help us find new interactions. Additionally, they are easy to implement.

4 Conversion of a UML diagram into general model

4.1 Introduction to UML diagrams

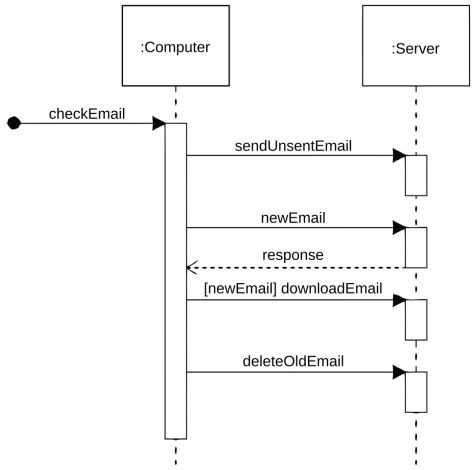

Figure 1:
Sequence diagram[30]

UML (Unified Modeling Language) diagrams have become a standard in the industry for modeling complex systems. Various types of UML diagrams exist[31], and the choice depends on the type of system being modeled and the information to be represented. They are categorized into structure diagrams (e.g., class, component, object), behavior diagrams (e.g., activity, state machine) and interaction diagrams (e.g., sequence, communication).

In this study we focus on sequence (mostly) and activity diagrams because FIISS and I-FASST methods are based on diagrams of such type.

Sequence diagrams show interactions between components arranged in time. A sequence diagram usually includes actors and message exchanges between them in timeline. In the example Figure 1) the actor Computer sends a message *sendUnsentEmail* to the actor Server.

4.2 Choosing a KG database

Knowledge graphs need a storage to keep and access data. For these purposes, databases are used. With growing popularity of knowledge graphs, there is variety of databases to choose: Neo4j, AWS Neptune,

OrientDB etc[32]. We will make an overview of each of them and then choose the one that fits us best.

In our situation, we have the following criteria: free license, Python language support and community support (nice to have).

Based on them, we built the following table to compare the most used databases[33]:

Table 3: Databases comparison

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

Only a few databases fit our requirements: ArangoDB, NebulaGraph, Neo4j and TerminusDB.

Azure Cosmos DB, TigerGraph, Oracle and Microsoft SQL Server do not have a free version, and OrientDB and JanusGraph do not support Python language.

TerminusDB

TerminusDB is a document-oriented knowledge graph, that is close to Git architecture. It uses GraphQL query language and WOQL (Web Object Query Language). It is focused mostly on documents,

that is not the content of our UML diagrams, so, probably, there is a better choice for us.

ArangoDB

ArangoDB is a native multi-model database that integrates graph, document, and key/value data models and uses AQL (ArangoDB Query Language), a SQL-like query language for complex queries between different data models. However, the free version is limited with 30-day subscription.

NebulaGraph

NebulaGraph is an open-source graph database with a focus on speed, which is used to maintain graphs with billions of vertices and trillions of edges with milliseconds of latencies.

Neo4j

Neo4j is an open-source database, that also supports an enterprise version. It remains a popular choice for building knowledge graphs[34]. It has great community support and both web and desktop versions and is available from a wide range of programming languages through Bolt interface.

Conclusion

Comparing Arango, NebulaGraph and Neo4j, we see that all of these options fulfill our needs so we are free to choose any of them. We will stop with Neo4j as it is most rated graph database among others [35].

4.3 APOC tool

Neo4j library has its own tool called APOC to convert an XML diagram into a knowledge graph[36]. This tool transforms the XML

elements into nodes and creates relationships based on their hierarchy in XML file (like "IS_CHILD_OF, NEXT, NEXT_SIBLING"). It is fast, reliable, and maintained by a team of Neo4j developers. However, it is not convenient in our case. The resulting graph is overwhelmed, the relationships are not informative enough, and working with it has almost no difference from working with an XML file (Figure 2).

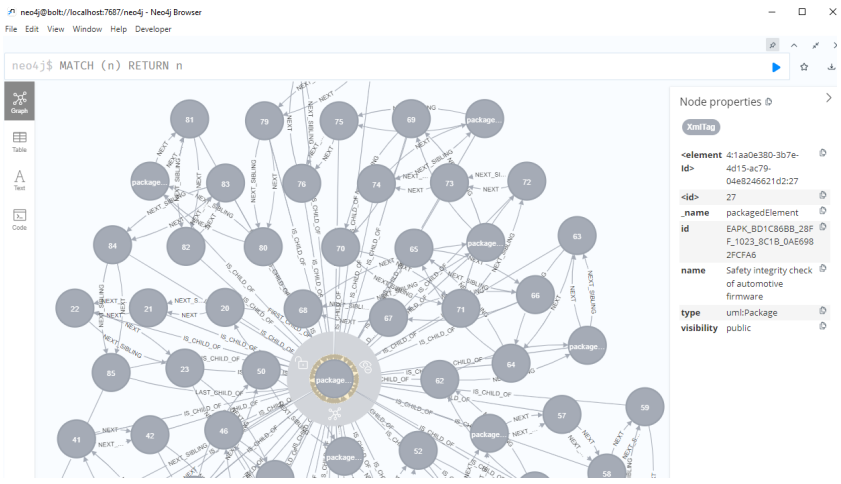

Figure 2: Knowledge graph built with the APOC tool

4.4 LLM transformation

There is tool called LLMGraphTransformer[37] to build knowledge graph using LLMs. It is quite fast and easy-to-perform, however, the results are non-deterministic since it is a neural network and there is a risk of data loss.

4.5 Custom realization

We will refer to the articles[38][39][40] to create a knowledge graph from a UML diagram.

We get inspiration from the research dedicated to enhancing UML class diagrams conversion to knowledge graph to define a set of rules[41].

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

Part 1. Rules for mapping nodes:

- We iteratively go through the whole graph and take all entities, except a few types (part 2) to map them to nodes.
- The node's label in the KG is equal to the entity's attribute "xmi:type", for example, "uml:Component". It is allowed to have multiple nodes of the same label.
- An entity's attribute "xmi:id" is mapped to the resulting node's attribute "node_id".
- Attributes of types "name", "covered", "client", "supplier", etc. are mapped to the resulting node without changes. It is not allowed to have multiple nodes with the same "node_id".
- Some attributes ("visibility", "isReadOnly", "isSingleExecution", "isIndirectlyInstantiated") are not mapped to a new diagram, as they are not needed for the analysis. However, it is possible to include them in future works to have extra information.

Part 2. Rules for mapping relationships:

- 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"
- A relationship with type "MESSAGE" occurs between nodes of type "uml:Lifeline". If there is an edge of type "OCC_MESSAGE" between two nodes, we build an edge of type "MESSAGE" between nodes, that represent their corresponding lifelines.

- A relationship has type "REALIZATION", if it is represented in the diagram as an entity with type "uml:Realization". It is an edge in the KG between a node of type "uml:Activity" and a node of type "uml:Component"

- A relationship has type "CONTROL_FLOW", if it is represented in the diagram as an entity with type "uml:ControlFlow". It is an edge in the KG between nodes representing entities of an activity diagram in the same way the diagram is built.

- A relationship with type "CLASSIFIES" occurs between nodes of types "uml:Component" and "uml:InstanceSpecification". To create it, firstly we check if an entity E has type "uml:InstanceSpecification" and, secondly, search for a node with node_id that equals to the E's property "classifier".

- A relationship with type "OCCURENCE_SPECIFICATION" occurs between nodes of types "uml:OccurenceSpecification" and "uml:Lifeline". To create it, we check if an entity E has type "uml:OccurenceSpecification" and search for a node with node_id that equals to the E's property "covered".

- A relationship with type "REPRESENTS" occurs between nodes of types "uml:Lifeline" and "uml:InstanceSpecification". To create it, we check if an entity A has a child of tag "ownedAttribute" and search for an entity B with type "uml:InstanceSpecification" so the A's attribute "idref" is equal to the B's attribute "represents".

- A relationship with type "BEHAVIOR_OF" occurs between nodes of types "uml:CallBehaviorAction" and "uml:Activity". To create it, we check if an entity A has type "uml:CallBehaviorAction" and search for an entity B with type "uml:Activity" so the A's attribute "idref" is equal to the B's attribute "node_id".

Let's consider an example, a UML diagram represented in XML format and one of its components, 'Firmware Verifier 21'. An XML entity with the attribute "xmi:type" = "uml:Component" represents a component from the source diagram.

In the XML file it is defined as follows:

```
<packagedElement xmi:type="uml:Component" xmi:id="EAID_-
8951E9DA_1663_B252_A946_AEE73BC0F606" name="Firmware Verifier
21" visibility="public" isIndirectlyInstantiated="true"/>
```

After conversion, we can see a node in the graph with type "uml:Component" (Figure 3), marked with blue color. It is connected to the node that represents the component's instance specification 'fwVerifier 21', marked with red color, with the edge of type 'CLASSIFIES' and then this instance specification is connected to the node that represents the corresponding lifeline 'fwVerifier 21', marked with green color.

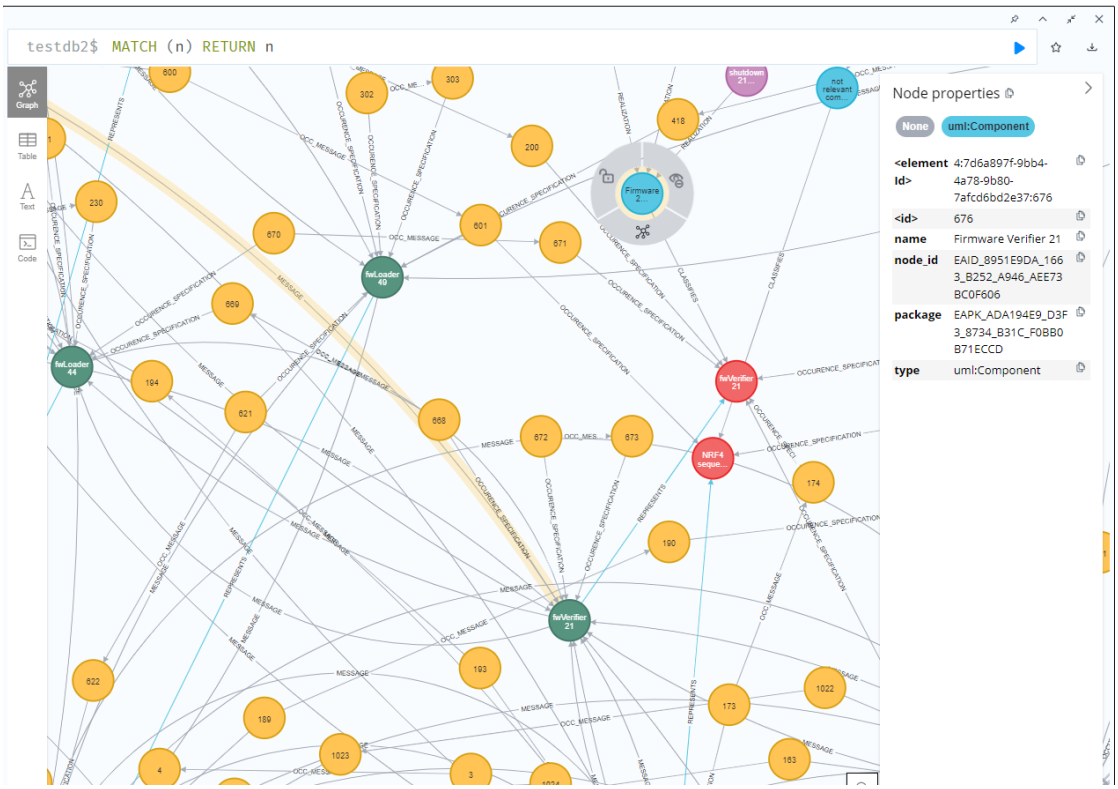

Figure 3: Firmware Verifier 21 on the graph

5 Performing the I-FASST analysis

5.1 FIISS code adaptation

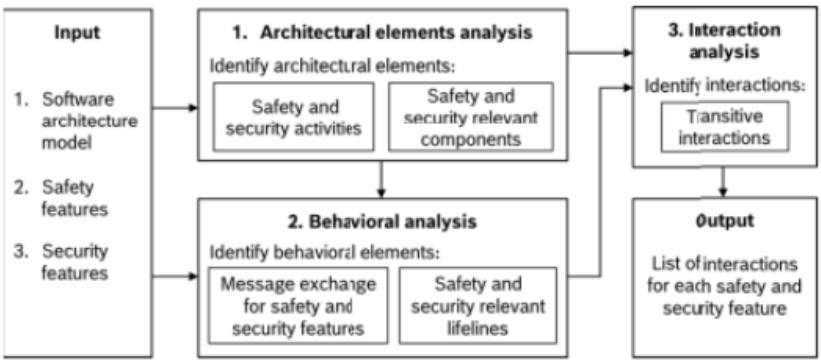

Figure 4: Overview of the FIISS method[1]

The FIISS analysis has three inputs: the diagram in the XML file itself, a list of safety features, and a list of security features. If we are interested in all the features in the diagram, we can deduce them by searching for the nodes with the type "uml:Package".

The algorithm consists of three steps (Figure 4) that we will recreate with relation to knowledge graphs:

A. Architectural elements analysis

The goal of this step is finding safety and security relevant activities and components. We will find all nodes in the graph that have type "uml:Activity" and for each node take its property "package", then define if it is safety or security package. Then for each activity find connected nodes and check if they have type "uml:Component". And so we are able to map each feature to its safety and security relevant components.

B. Behavioral analysis

In the second step we search for message exchanges between lifelines and define if these lifelines are safety or security relevant.

Because of the specific rule of building sequence diagrams, a message exchange can happen between two nodes of type

"uml:OccurenceSpecification", that refer to their lifelines using property "covered". To simplify the search, we added extra edges between their corresponding lifelines, so we could directly refer to them. We form a query using Cypher language

```
MATCH (n) -[r]-> (m) WHERE r:MESSAGE RETURN n, r, m
```

And then we have all possible direct message exchanges in the graph between lifelines.

We can also find a lifelines package by taking the property "package" of it. And we can also define whether a lifeline is safe or secure, because its nodes are connected to nodes of type "uml:InstanceSpecification" with edge of type "REPRESENTS", and they are connected to the components with an edge of type "CLASSIFIES". Then we map each lifeline to safety, security, or both safety and security lifelines list.

C. Interaction analysis

The last step is about analyzing the information obtained in order to find interaction paths. For each feature, we take its lifelines and search for paths from one to another using different combinations. In the XML version of FIISS a separate graph is built for paths searching, but it is not necessary in our case because we already have a graph. To find paths, we use the query:

```
MATCH p = (n) -[*]-> (m) WHERE n.covered = $start_value
and m.covered = $end_value RETURN p
```

This query can be used to configure the maximum path length:

```
f"MATCH p = (n) -[*..max_length]-> (m) WHERE n.covered =
$start_value and m.covered = $end_value RETURN p"
```

Now we classify the paths we have, checking if the first node represents a safety relevant lifeline and the second is a security relevant lifeline etc.

The output of the method is a set of features mapped to their interactions that consist of path, interaction_type (safety $\rightarrow$ security, safety-security $\rightarrow$ safety etc.), and both source and destination lifelines.

5.2 Propagation into I-FASST

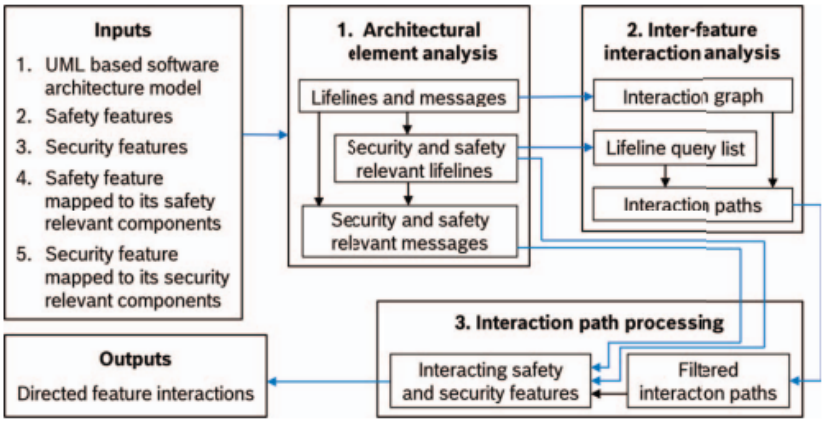

Figure 5: Overview of the I-FASST method[2]

The I-FASST method requires five inputs: the diagram itself, lists of safety and security features and for each feature its mapping to its safety and security relevant components.

Similarly to FIISS, the I-FASST analysis has three steps (Figure 5):

A. Architectural element analysis

1) For each feature we search for its safety and security relevant lifelines, using the same method as in FIISS, by a package property.

2) Then also repeat the procedure from FIISS to find message exchanges.

B. Inter-feature interaction analysis

The step has three parts:

1. Building an interaction graph based on message exchanges

This step was necessary for the XML version of I_FASST analysis, but we do not need it because we already have a graph.

2. Lifeline query list

Building a list of queries, where each query is a couple of source and destination lifeline from A.1) and the first in security relevant and second is safety relevant and vice versa.

3. Interaction paths

Applying queries to the graph to find paths (as in the FIISS). We also can configure their maximum length p as was shown above.

C. Interaction path processing

1. Filtering interaction paths

We need to filter out paths that contain chains of interactions. If there are indirect interaction paths (containing intermediate links), these links may also be relevant to safety or security, leading to a chain of interactions that we want to avoid.

We can achieve this by querying and filtering the results. The path obtained in the search consists of a set of nodes. In order to find intermediate nodes, we remove the first and last node and then check each node to see if it represents a safety or security-relevant lifeline. If so, we ignore that path and do not include it in the final set of interactions.

2. Interacting safety and security features

Then "good" paths form a list of interactions, where each interaction is a pair of interacting features.

5.3 The unit-under-test

Another important question is the evaluation of our method. We will compare it with the existing realization of I-FASST method, and for this we need a sample project and a list of metrics. Our sample project has to be large enough and have different test cases to check. The FIISS and I-FASST environments were tested on the architectural model of an ECU containing 22 security features and 48 safety features. The diagram was provided by a company, and, as it was a real-world project, it is not possible to have it due to confidentiality. So we will build our own project that is close to this according to the information we have. Our requirements are as follows. The diagram has to have:

- Relevant direct message exchanges;
- Relevant indirect message exchanges;

- Irrelevant indirect message exchanges (chain interaction paths);
- Message exchanges to itself.

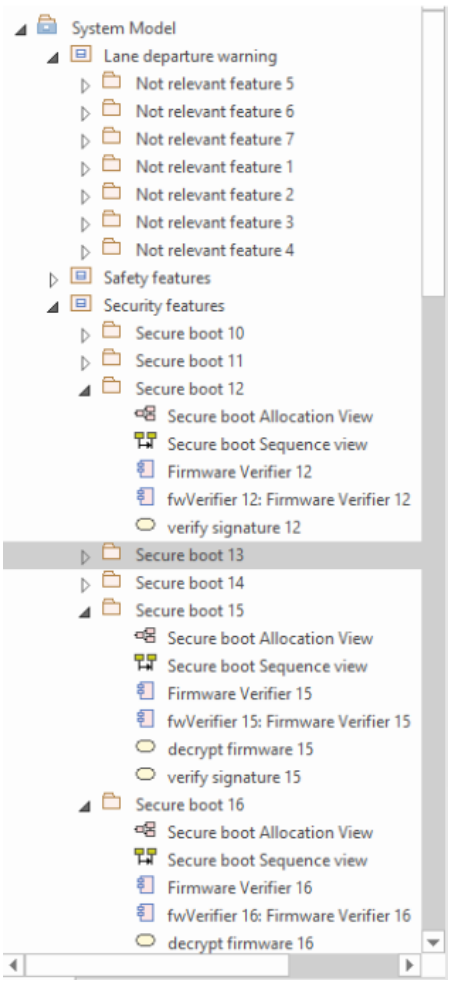

Figure 6:
Diagram root

We will use examples from Priya's work[1][2] to build something similar. Then, we will extend it by adding additional safety and security features and corresponding components. We will also add lifelines and messages between them. It is not essential to have real meaningful names for the safety and security components in our case. We just need to check if the tools can detect and filter the paths.

Our diagram (Figure 6) contains a set of safety features, a set of security features, and a set of irrelevant features for testing pathways with intermediate nodes.

Direct message exchange happens between two relevant lifelines. We are interested in cases where the first line of defense is safety relevant and the second line is security relevant, and vice versa.

The example (Figure 7) shows the direct path between a safety relevant lifeline fwLoader 50 and a security relevant lifeline fwVerifier 9.

Undirect message exchange occurs between two relevant lifelines but there are some intermediate nodes, that can be safety or security relevant or not.

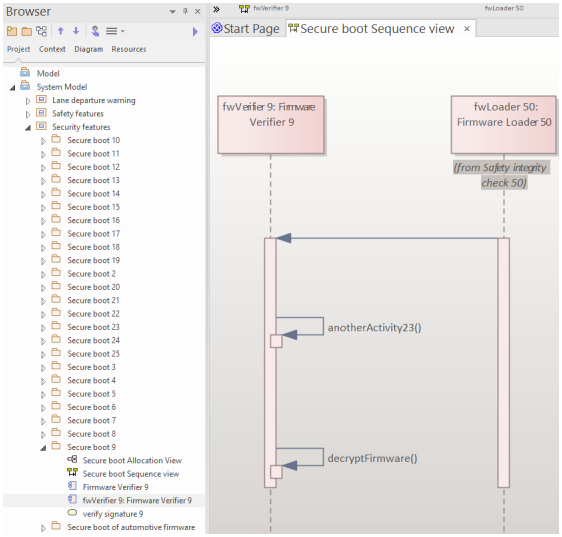

Figure 7:
Direct path

The message exchange between security feature fwVerifier 18 and safety feature fwLoader 16 is relevant (Figure 8), because the intermediate nodes are neither safety nor security relevant.

However, the path between safety relevant lifeline fwLoader 8 and security relevant lifeline fwVerifier 22 has interaction chains (Figure 9), because the intermediate lifeline fwVerifier 7 is security relevant.

There are also messages sent from a lifeline back to itself (for example, the message decryptFirmware() in Figure 7), but we do not think they are relevant at the moment. However, there is potential for future research, because these messages may provide valuable information at some point in the future.

After conversion to a knowledge graph, our final version of the diagram contained around 1,300 nodes and 1,400 relationships.

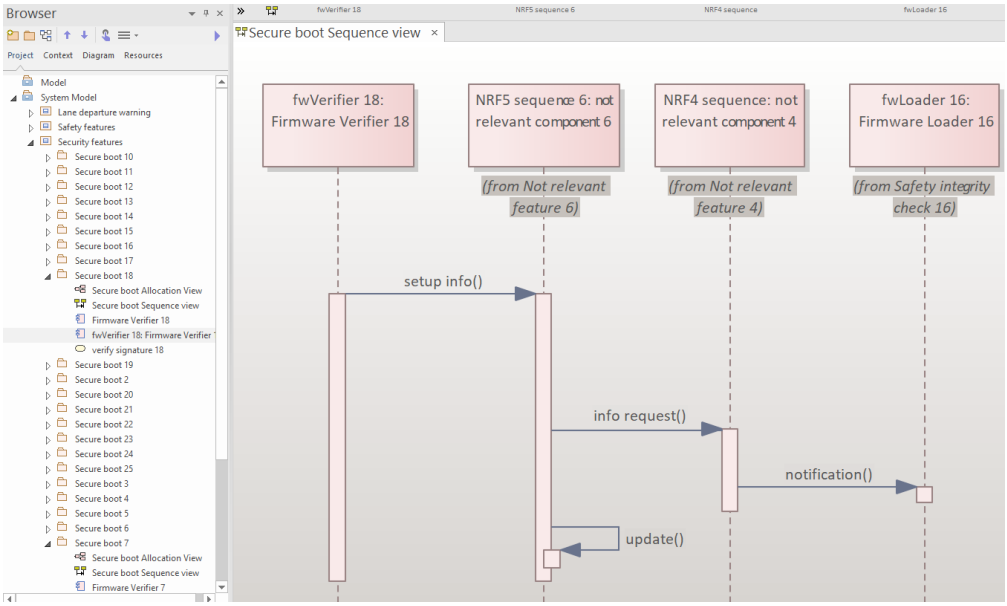

Figure 8: A relevant example

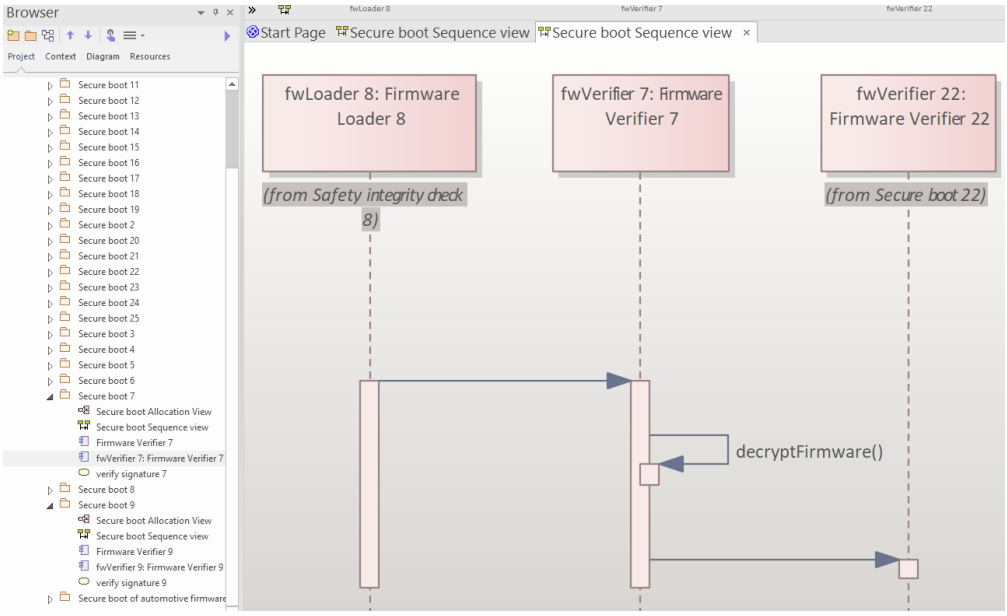

Figure 9: An irrelevant example

6 Results comparison

6.1 Execution times

The search in I-FASST method takes around 2 seconds.

In our case, we want to find all possible paths between every pair of nodes in a set of queries. To accomplish this, we need to query the database every time. If we used a Cypher query to find all paths at once, it would take about 30 seconds on our test dataset. This isn't a major problem, but it's relatively slower than the I-FASST search.

Neo4j has its own algorithms, and we tried two: Dijkstra's source-target shortest path search to find a path between two nodes, which took 21 seconds in the test. All pairs shortest path searching for all paths in the graph at once was fast, but only outputted source and destination information, so we wanted to exclude intermediate nodes.

It is possible to convert a Neo4j graph into a NetworkX graph, which is used in I-FASST. We applied this method and got a result in about 2 seconds.

6.2 The information obtained

We can count the number of true positives in relation to the total number of interactions found, as was done in Priya's study: FIISS analysis. However, in that case, the check was carried out by professional engineers employed by a company that we do not have access to. Therefore, we created the scenarios mentioned above in the test diagram and tested how well both methods handled them.

We can see that I-FASST for XML diagrams finds 15 paths, and our method is able to detect all of them as well (Figure 10).

6.3 Non-discrete metrics

Working with knowledge graphs has some advantages compared to working with XML using `lxml` library. Of course, the library is

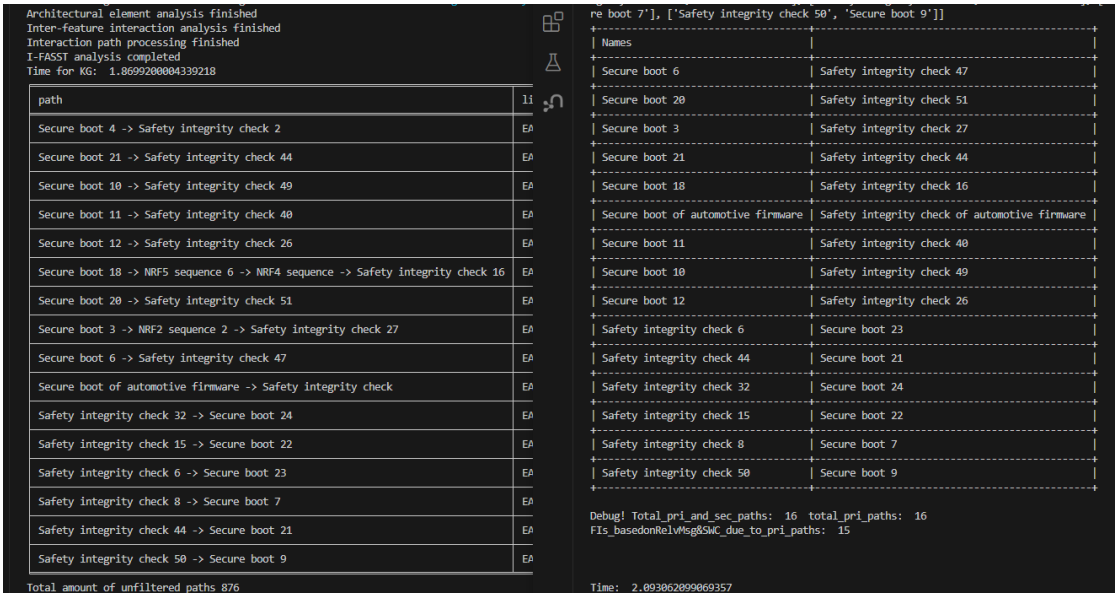

Figure 10: Results comparison

powerful, and we can use searching paths, but it is hard to debug.

1. It is easier to work with graphs in complex cases, because we can use the Cypher query language to build a complex query and analyze the data.

2. We do not need to create a separate graph in order to find potential safety and security interactions, because the data is already presented as a graph. However, if the speed is important, building a separate graph using NetworkX is needed.

3. The graph representation of data is more user-friendly. It makes it easier to detect potential errors and allows to search for information using the Cypher query language.

4. It is possible to modify a graph that has already been built: not only adding, but also removing and changing nodes and edges. This is more difficult in the case of UML diagrams that are represented as an XML file.

5. All the data is stored in a database that can be hosted in the cloud, so engineers can work with large datasets using a regular computer, and if the file is an XML file, all the necessary data will be stored in RAM.

7 Runtime verification: state-of-the-art

The fast-evolving technologies and growing speeds of information exchange require means to perform verification during runtime. The potential of resolving the dangerous interactions in real time may simplify the engineers routine. Moreover, runtime verification may reveal the information that can not be obtained during offline analysis. For example, diagram designers will be able to instantly see the impact of adding new elements to the architecture. The system behavior may also find possible safety-security interactions based on various metrics, for example, timings. Also, the link prediction feature of knowledge graphs can be used to predict possible interactions using existing graph state.

Runtime verification is crucial for various fields of life, from drones cybersecurity testing[42] and smart agents monitoring[43] to hacker attacks on pacemakers[44]. In SIES 2018, researchers from University of Basque Country presented Cetratus[45]: solution to update software without system shutdown in the field of Industrial Internet of Things.

The Control-Flow Integrity technique is widely used. However, in some cases, it can cause performance overhead. For example, in mixed-critical systems[46].

7.1 Existing approaches

Formal approach

Amongst the existing approaches it is worth to mention formal approach[47][44], especially Frama-C and tools related to it[48] as a huge part of this study and UML-related tools[49]. Solutions based on formal approach are not applicable to us because of scalability problems, however, we may get some inspirations while studying them.

Machine learning approach

Machine learning methods are also applied to the runtime verification problem. The multi-agent reinforcement learning method for safety runtime verification for intelligent warehouses[50] was proposed and tested in 2024.

Control-Flow integrity approach

Control-flow integrity approach is quite popular, and there are a lot of solutions based on it. The comparison of existing control-flow techniques was researched in 2017 and 2019[51][52].

KCoFI[53] is a tool to protect commodity operating systems from various kinds of attacks through control-flow integrity without using heavyweight complete memory safety. Another example is DExIE[54], a hardware monitor that catches illegal control flows in time to prevent any illegal instructions from touching memory. TitanCFI[55] is a solution that was made to enforce CFI in the Root-of-Trust feature of RISC-V platforms.

7.2 Criterias for runtime verification

It is interesting to take a look on criterias that are considered in different solutions to define some patterns and ideas what we could use in our case.

In formal verification criterias are often defined rules that express desired program properties, for example, if we speak about hyperproperties[44], data minimality, non-interference, integrity, and software doping.

If the entities under study actively act with the environment, for example, multi-agent systems, criterias include environmental constraints[56]. Also for multi-agents the constraints include social interactions because they act in a group.

Runtime verification in complex system may consider the situation when the system state changes, like when a new component is

added or removed. And the properties checked may include communication latency between components, average amount of messages is below a specific amount, total amount of the allowed/authorized connections, the ability of connection or movement[57].

If the program properties include cybersecurity, we can check main characteristics of cybersecurity, such as confidentiality, integrity, availability, authentication, authorization and non-repudiation[42].

For systems that involve network traffic usage and latencies are critical, the measurement of timestamp and frequency for every device can help to prevent them[58].

Is is observable that criterias under study depend on current problem and may vary strongly to satisfy the requirements. To create a set of criterias for the system, we look at possible scenarios that may happen and lead to undesired behavior.

7.3 Conclusion and problem formulation

Looking at the research, we can see that most of it focuses only on the safety (or security) aspect, without considering the other aspects.

The solutions are quite specific, and it is difficult to find anything relevant to UML sequence diagrams. However, there are solutions that relate to UML activity diagrams. Nobody has previously attempted to apply runtime verification to the I-FASST method.

However, we could use some of the ideas from these research and apply them to knowledge graphs.

8 Runtime verification: possible scenarios

Based on the state-of-the-art, we can say that this is a good approach for defining possible test scenarios for our system and for formulating criteria that may indicate undesirable consequences.

In the case of knowledge graphs, we formulated some inspirations to consider:

- Monitor a state of the system when a new element (in case of graphs, node or relationship) is added or removed;
- Look on different diagrams and find some possible paths with relation to timings.

Starting from this, we will consider our graph as dynamic graph[59], that means it is changing over time.

8.1 Timing aspect

Scenario 1

Firstly we model the example when a new element is added to the system. Consider our knowledge graph and, when a new edge or component is added, we check the possible impact on system.

The test case would be about adding a new relationship to the graph (a new node itself will not affect the possible paths).

We search for paths based on messages between lifelines, so let's consider a new edge of type MESSAGE between nodes of type "uml:Lifeline". Our solution adds the new edge to the graph and checks its relevance. The first criteria here is relevance of source and destination.

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

lifeline is safety relevant and the second lifeline is security relevant, and vice versa. We say that a lifeline is not relevant, if it is neither safety nor security relevant.

The possible situations are as follows:

- The source and destination lifelines have the same relevance;
- The lifelines have different relevance;
- At least one of the lifelines is neither safety nor security relevant.

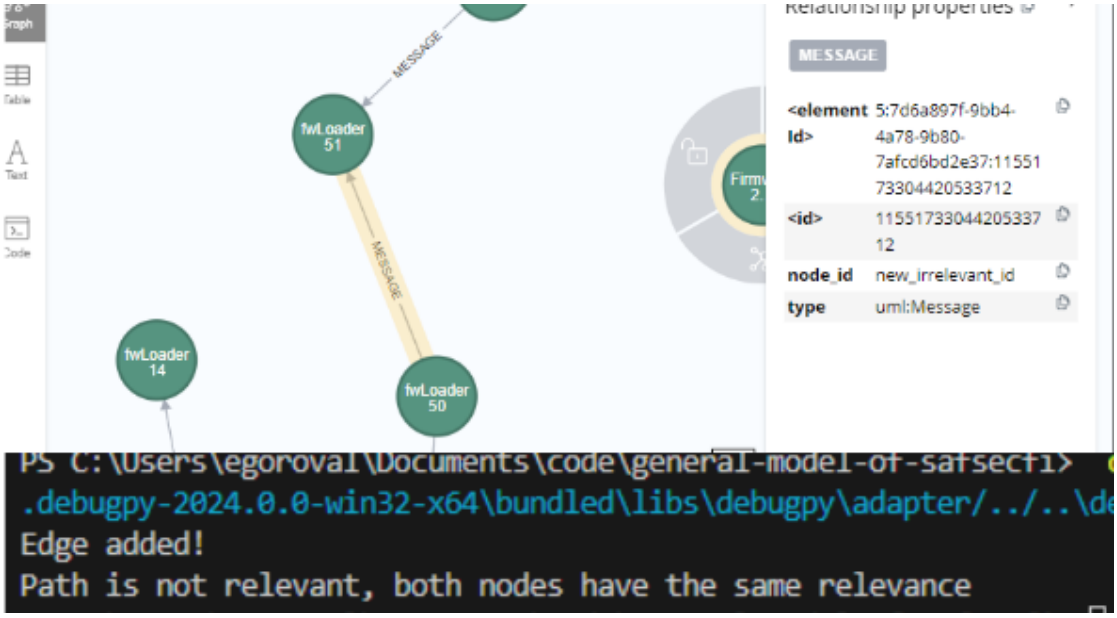

Figure 11: The same relevance

1. If the source and destination lines have the same relevance (Figure 11), then the new path is not of interest to us.

2. If the lifelines are of different relevance (Figure 12), there may be a new, possibly relevant path - from the source lifeline to the destination.

3. a) Consider a situation where the source lifeline has safety or security relevance, but the destination lifeline does not. We then try

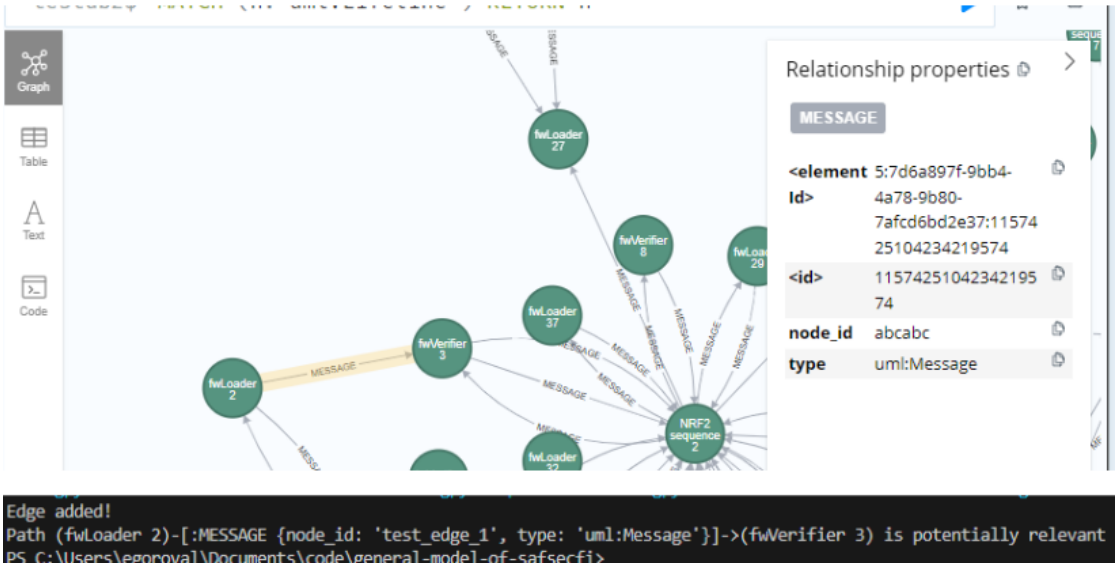

Figure 12: The different relevance

to find routes from the destination lifeline towards relevant lifelines, and form a collection of routes from the source lifeline leading to these relevant ones. Of course, only lifelines with relevance different from that of the source are of interest to us.

b) The similar situation occurs when the source lifeline is not relevant, but the destination lifeline is relevant. We look for paths that lead to the source lifeline and start from the relevant lifeline. Then, we form a collection of paths by combining the paths from those relevant lines to our target node.

c) If both lifelines are not relevant, we perform a combination of steps 3a) and 3b). We look for paths that lead to the source lifeline and paths that come from the destination lifeline. For the first set, we create two groups based on the relevance of the lifeline line and get a group of safe inputs and a group of secure inputs. For the other group, we make two more groups of outputs in the same manner, based on how relevant the end lifeline is.

Then we take a set of safe input paths and a set of secure output paths, and get their Cartesian product, because all possible combinations of these paths might be interesting to us. Then we form

the final paths as follows: (safety input path) - [the added edge] - (security output path).

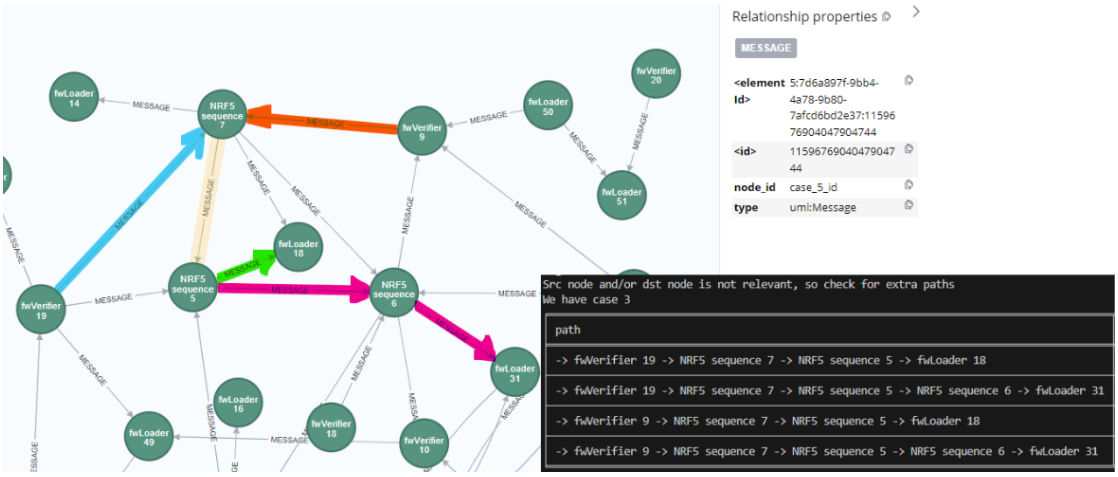

Figure 13: Both lifelines are not relevant

Then we do the same process using a set of secure inputs and a set of outputs.

The output of the check function is an array of possible interactions that may occur when a new edge is added.

The example (Figure 13) demonstrates the connection NRF5 (Not Relevant Feature 5) sequence 7 → NRF5 sequence 5, the both lifelines are not relevant. Then we checked that there are two (security) relevant lifelines that have messages to the NRF5 sequence 7, and two (safety) relevance lifelines that receive messages from the NRF5 sequence 5. So we count the Cartesian product and obtain 4 possible relevant paths.

Scenario 2

Now we consider the case of finding possible paths with respect to timing. We will search for paths between various sequences of diagrams.

The diagrams run in real time, and messages are transferred from one line to another. Therefore, if there was an exchange of messages

between lines A and B in diagram D1, it is possible that there will be an interaction between lines B and C in another diagram D2 (Figure 14).

Figure 14: Scenario 2

We consider the relevant paths that belong to a diagram and try to combine them, as in the I-FASST. A path must begin and/or end on a non-relevant lifeline, so we examine other diagrams to find paths where the starting lifeline equals the ending lifeline.

There are three potential cases as in the scenario 1, situation 3 and we form the paths in the same way.

When potential paths are found, we consider timings – in our case, it is the integer message property msg_time. For the next message, the timing value must be greater than for the previous one.

The example (Figure 15) demonstrates a simple path with a length of 2. Messages in the path are from different diagrams and are identified by their diagram_id property. The msg_time property of the first message has a lower value than that of the second, which may suggest that this path could be relevant at runtime.

8.2 Forensics checking

Runtime verification is often tied to log analysis. Safety and security components of a system interact with the environment and collect data from the system. This data is recorded in logs. In this section

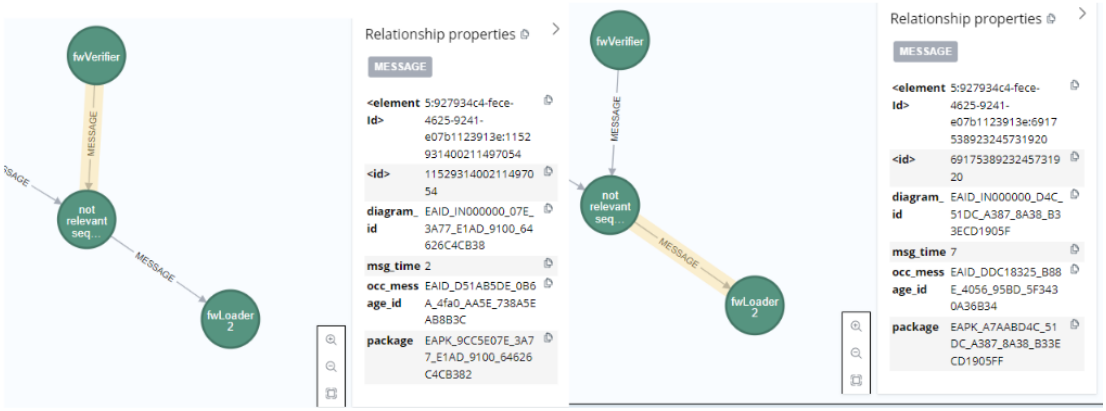

Figure 15: Scenario 2, example

we try to answer how log information helps in runtime analysis by using knowledge graphs.

State-of-the-art

Working with logs involves the following steps:

- Log collection;
- Log parsing;
- Log filtering;
- Anomaly/intrusion detection.

The information obtained in logs may vary. In the context of safety, there may be errors arising from aging systems or misconfigurations. This leads to unexpected values being recorded in logs. In terms of security, there may be cyber-pranks[60], when an intruder causes monitoring systems to report an error when there is none and the system stops or behaves differently. Machine learning methods[61][62] are helpful for diagnosing and detecting anomalies.

Log parsing usually depends on the initial log structure and is often relatively easy.

Logs filtering sometimes involves logs classification, Multinomial Naive Bayes[63] and statistical correlation analysis[64] can be used for this.

Possible scenarios

In our case, it is possible to use logs to enrich knowledge graph because of knowledge graphs' property to store various types of information.

We can consider the following approaches:

- Check log information before sending a message
  - If the log parameters show some out-of-border values, send alert message
  - This also may stop sending the first message
- If there is an unexpected message, record it to logs
  - For example, error message from a security feature to itself
  - The message that is marked as unexpected

Scenario 1

Consider an automotive vehicle with automatic braking safety feature and firewall security feature. These features are transferred into the knowledge graph through corresponding lifelines.

From the I-FASST method we learn that there is a relevant message between the lifelines. Then we check KG for the safety component and see that there is an anomaly: sensors' values are out-of-bounds. Then we print an alert message or record it to logs, etc. (Figure 16).

Scenario 2

In this case we check the logs information first to find warnings or errors.

If there is a warning about an error related to a feature, we should go to the lifeline of that feature and add a relationship between the

Log format and info may vary depending on the real system

```
PS C:\Users\egoroval\Documents\code\general-model-of-safsecfi> & "C:/Program Files/Python312/python.exe"
Alert! Anomaly Sensor value S1 is out-of-bounds detected in EAID_LL000000_8428_2537_A50F_802085E313F8
PS C:\Users\egoroval\Documents\code\general-model-of-safsecfi>
```

Figure 16: Scenario 1 — Printing an alert message to logs

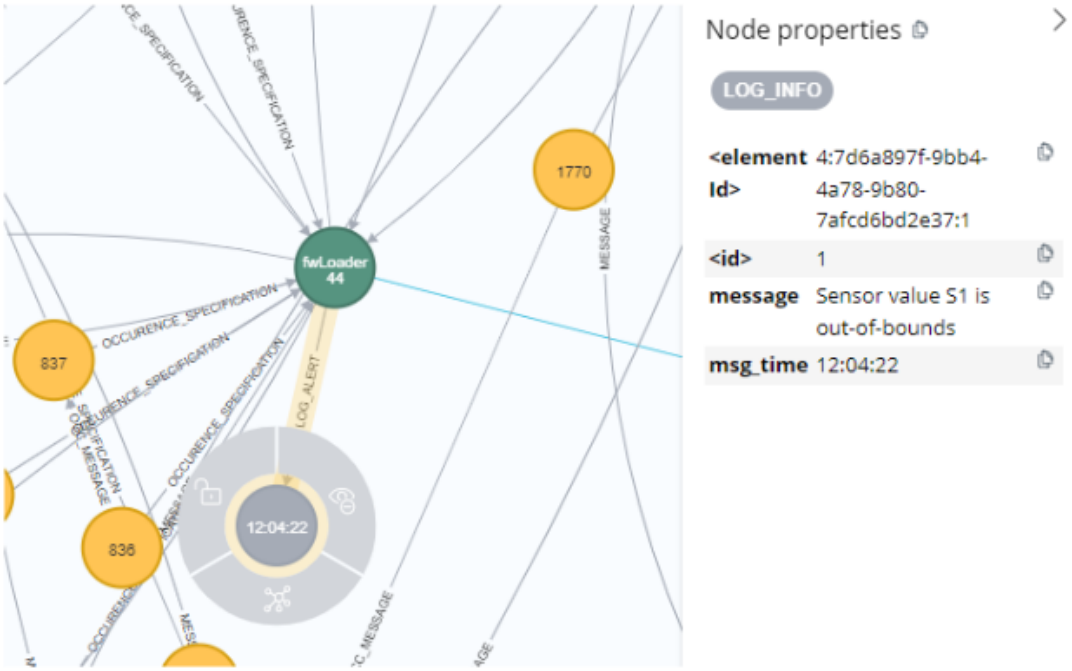

Figure 17: Scenario 2 — Adding logs info to the KG

error and the component that caused it (Figure 17). We can add any type of relationship we want, as well as any type of property.

It is also possible to mark a lifeline as an error, but this may hide an error's time, so a message approach may be more effective. These are implementation details and can be decided upon by engineers.

9 Project assessment

9.1 Overall cost

This project does not require any hardware except for a PC, but there are some software and extra resources needed to be used (Table 4).

Table 4: Project assessment costs

Item	**Cost (euros)**
Python 3.0+	0
IDE (in this project we used Visual Studio Code)	0
Neo4j database tools (we used Neo4j Desktop)	0
Gitlab for system control version	0
Enterprise Architect	0 (for 1 month)
Beluga, IEEE Explore and other archives	Not stated (LCIS funding)
Supervisor's time	Not stated
Internship allowance ~	3000
PC	~1000
Heating and maintenance of the workspace in a laboratory	Not stated
Technical support (for IT or administrative matters	Not stated

9.2 Results obtained in relation to the specifications

Now let's return to the specifications and see if any of our desired goals have been achieved (Table 5):

Table 5: Results obtained

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

9.3 Impact of the project to the company

There is no specific company that we work with, but we aim to help the industry as a whole. The knowledge graph approach can be potentially applied to various fields related to safety and security.

The work done potentially helps to store all relevant information in one place and visualize it, and find potential interactions using the I_FASST method.

The LCIS laboratory has its own goals. After discussion with Laurent Lefevre, the sustainability reference person of the laboratory, we established that the project is related to sustainable development and has future potential (see section 1.1).

10 Conclusions and future work

10.1 Conclusions

So far, we have the following results:

- The approach to convert an UML diagram into knowledge graph was found;
- The FIISS and I-FASST methods were implemented using knowledge graphs;
- The possible interactions between knowledge graphs and runtime verification have been discovered based on timing and logging.

10.2 Future work

The topic still offers a vast field of research, and there are many ideas to be inspired by.

The test case generation[65] is interesting to us, as it will allow us to automatically validate diagrams after they are generated.

Adding a user interface to the tools will improve user experience and hide the implementation details. Probably there are other uses for graphs in terms of runtime. We want to believe it is possible for industry to adopt our approach.

11 Summary

The goal of this work is to continue previous research done in the LCIS laboratory, which was dedicated to finding a method for detecting potential safety and security interaction paths in automotive systems using UML diagrams. We aim to improve this method by answering whether it can be implemented on a more general model such as knowledge graphs and comparing the paths obtained from both methods.

The second part of the project was a study of runtime verification opportunities and their applicability to the current I-FASST system. Potential interaction scenarios were identified: the first scenario involves adding a new message exchange to the system and tracking its effect on the system; the second scenario involves searching for potential interactions between sequence diagrams.

The forensic aspect was also taken into account, and scenarios were discovered for using log data together with knowledge graphs. For example, paths obtained in I-FASST can be filtered based on log data and a resulting table can be printed with alert messages, or those paths can simply be removed from the results.

This internship gave me a lot of new knowledge and experience. It was fun researching and meeting new people in the laboratory. I would like to thank everyone I met for their kindness and support. Maybe we will meet again this fall?

Bibliography

Articles

[1] Priyadarshini et al. “Feature-based software architecture analysis to identify safety and security interactions”. English. In: IEEE Computer Society, Mar. 2023, pp. 12–22. ISBN: 979-8-3503-9749-9. DOI: 10.1109/ICSA56044.2023.00010. URL: https://www.computer.org/csdl/proceedings-article/icsa/2023/974900a012/1MBRI5XXPHi.

[2] Priyadarshini et al. “ Software architecture based inter-feature analysis of safety and security interactions ”. In: *2024 19th European Dependable Computing Conference (EDCC)*. Los Alamitos, CA, USA: IEEE Computer Society, Apr. 2024, pp. 165–168. DOI: 10.1109/EDCC61798.2024.00040.

[3] Wei Luo et al. “Fast Game Verification for Safety- and Security-Critical Distributed Applications”. In: *IEEE Transactions on Dependable and Secure Computing* PP (Jan. 2024), pp. 1–18. DOI: 10.1109/TDSC.2024.3417022.

[4] Svitlana Shevchenko et al. “Game Theoretical Approach To The Modeling Of Conflicts In Information Security Systems”. In: *Cybersecurity Education Science Technique* 2 (Dec. 2023), pp. 168–178. DOI: 10.28925/2663-4023.2023.22.168178.

[5] Megha Quamara, Christina Kolb, and Brahim Hamid. “Analyzing Origins of Safety and Security Interactions Using Feared Events Trees and Multi-level Model”. In: Springer Nature, Sept. 2023, pp. 176–187. ISBN: 978-3-031-40952-3. DOI: 10.1007/978-3-031-40953-0_15.

[6] Wenli Shang et al. “A Tree-based Scheme to Assess the Safety and Security for Industrial Control Systems”. In: *IEEE Transactions on Dependable and Secure Computing* PP (Jan. 2024), pp. 1–11. DOI: 10.1109/TDSC.2024.3475632.

[7] Megha Quamara, Christina Kolb, and Brahim Hamid. "Analyzing Origins of Safety and Security Interactions Using Feared Events Trees and Multi-level Model". In: *Computer Safety, Reliability, and Security. SAFECOMP 2023 Workshops*. Ed. by Jérémie Guiochet et al. Cham: Springer Nature Switzerland, 2023, pp. 176–187.

[8] Elena Troubitsyna. "Formal Analysis of Interactions Between Safety and Security Requirements". In: Springer Nature, Sept. 2024, pp. 141–161. ISBN: 978-3-031-66672-8. DOI: 10.1007/978-3-031-66673-5_8.

[9] Inna Vistbakka and Elena Troubitsyna. "Pattern-Based Formal Approach to Analyse Security and Safety of Control Systems". In: Springer Nature, Oct. 2019, pp. 363–378. ISBN: 978-3-030-32871-9. DOI: 10.1007/978-3-030-32872-6_24.

[10] Siddhartha Verma et al. "Combined Approach for Safety and Security". In: Springer, Aug. 2019, pp. 87–101. ISBN: 978-3-030-26249-5. DOI: 10.1007/978-3-030-26250-1_7.

[11] Mosiur Rahaman et al. "AI Safety and Security". In: ResearchGate, Aug. 2024, pp. 354–383. ISBN: 9798369338605. DOI: 10.4018/979-8-3693-3860-5.ch011.

[12] H. Abdo et al. "A safety/security risk analysis approach of Industrial Control Systems: A cyber bowtie – combining new version of attack tree with bowtie analysis". eng. In: *Computers & security* 72 (2018), pp. 175–195. ISSN: 0167-4048.

[14] Xiaorong Lyu, Yulong Ding, and Shuang-Hua Yang. "Safety and security risk assessment in cyber-physical systems". In: *IET Cyber-Physical Systems: Theory & Applications* 4.3 (2019), pp. 221–232. DOI: https://doi.org/10.1049/iet-cps.2018.5068. URL: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-cps.2018.5068.

[15] Chidi Agbo and Hoda Mehrpouyan. "Conflict Analysis and Resolution of Safety and Security Boundary Conditions for Industrial Control Systems". In: *IEEE Explore* (May 2023). DOI: 10.48550/arXiv.2305.06185.

[16] Jing Wang et al. "Study on Dual-security Knowledge Graph for Process Industrial Control". chi. In: *Ji suan ji ke xue* 50.9 (2023), pp. 68–. ISSN: 1002-137X.

[17] Markus Fockel et al. "Semi-automatic Integrated Safety and Security Analysis for Automotive Systems". In: *ResearchGate* (Jan. 2022), pp. 147–154. DOI: 10.5220/0010778500003119.

[18] Helmut Martin et al. "Safety and Security Co-engineering and Argumentation Framework". In: *International Conference on Computer Safety, Reliability, and Security*. Sept. 2017, pp. 286–297. ISBN: 978-3-319-66283-1. DOI: 10.1007/978-3-319-66284-8_24.

[19] Roman Trentinaglia et al. "Automatically deriving test cases from safety-security dependencies". In: *22th escar Europe : The World's Leading Automotive Cyber Security Conference : Embedded Security in Cars* (Oct. 2024). DOI: 10.13154/294-12716.

[24] Zhenpeng Shi et al. "Uncovering CWE-CVE-CPE Relations with Threat Knowledge Graphs". en. In: *ResearchGate* (Sept. 2024). DOI: 10.48550/arXiv.2305.00632. URL: https://www.researchgate.net/publication/370443104_Uncovering_CWE-CVE-CPE_Relations_with_Threat_Knowledge_Graphs.

[25] Jeisson Vergara-Vargas et al. "Sarch-Knows: A Knowledge Graph for Modeling Security Scenarios at the Software Architecture Level". en. In: *ResearchGate*. Sept. 2024. DOI: 10.1007/978-3-031-66326-0_7.

[26] Romy Fieblinger, Md Tanvirul Alam, and Nidhi Rastogi. "Actionable Cyber Threat Intelligence using Knowledge Graphs and Large Language Models". In: *arXiv* arXiv:2407.02528 (June

2024). arXiv:2407.02528 [cs]. DOI: 10.48550/arXiv.2407.02528. URL: http://arxiv.org/abs/2407.02528.

[27] Kai Liu et al. "A review of knowledge graph application scenarios in cyber security". In: *arXiv* arXiv:2204.04769 (Apr. 2022). arXiv:2204.04769 [cs]. DOI: 10.48550/arXiv.2204.04769. URL: http://arxiv.org/abs/2204.04769.

[38] Yan Jia et al. "A Practical Approach to Constructing a Knowledge Graph for Cybersecurity". en. In: *ResearchGate* (Oct. 2024). DOI: 10.1016/j.eng.2018.01.004.

[39] Paul Ziemann, Karsten Hölscher, and Martin Gogolla. "From UML Models to Graph Transformation Systems". In: *Electronic Notes in Theoretical Computer Science* 127.4 (2005). Proceedings of the Workshop on Visual Languages and Formal Methods (VLFM 2004), pp. 17–33. ISSN: 1571-0661. DOI: https://doi.org/10.1016/j.entcs.2004.10.025. URL: https://www.sciencedirect.com/science/article/pii/S1571066105001763.

[41] Liang Huang et al. "Enhancing UML Class Diagram Abstraction with Knowledge Graph". In: ResearchGate, Sept. 2016, pp. 606–616. ISBN: 978-3-319-46256-1. DOI: 10.1007/978-3-319-46257-8_65.

[42] Eda Marchetti, Tauheed Waheed, and Antonello Calabrò. "Cybersecurity Testing in Drones Domain: A Systematic Literature Review". In: *IEEE Access* PP (Jan. 2024), pp. 1–1. DOI: 10.1109/ACCESS.2024.3495994.

[43] Emilia Cioroaica et al. "Towards runtime monitoring for malicious behaviors detection in smart ecosystems". In: *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)* (Oct. 1, 2019), pp. 200–203. DOI: 10.1109/issrew.2019.00072. URL: https://doi.org/10.1109/issrew.2019.00072.

[44] Srinivas Pinisetty et al. “Security of Pacemakers using Runtime Verification”. In: *IEEE Explore* (Oct. 1, 2018), pp. 1–11. DOI: 10.1109/memcod.2018.8556922. URL: https://doi.org/10.1109/memcod.2018.8556922.

[45] Imanol Mugarza, Jorge Parra, and Eduardo Jacob. “Cetratus: Towards a live patching supported runtime for mixed-criticality safe and secure systems”. In: *SIES 2018* (June 1, 2018), pp. 1–. DOI: 10.1109/sies.2018.8442088. URL: https://doi.org/10.1109/sies.2018.8442088.

[46] Marine Kadar et al. “Safety-Aware Integration of Hardware-Assisted Program Tracing in Mixed-Criticality Systems for Security Monitoring”. In: *IEEE* (May 1, 2021). DOI: 10.1109/rtas52030.2021.00031. URL: https://doi.org/10.1109/rtas52030.2021.00031.

[48] Julien Signoles, Nikolai Kosmatov, and Kostyantyn Vorobyov. “E-ACSL, a Runtime Verification Tool for Safety and Security of C Programs (tool paper)”. In: *Kalpa publications in computing* 3 (Jan. 12, 2018), pp. 164–153. DOI: 10.29007/fpdh. URL: https://doi.org/10.29007/fpdh.

[49] François Siewe and Guy Merlin Ngounou. “On the Execution and Runtime Verification of UML Activity Diagrams”. In: *Software* 4.1 (Feb. 27, 2025), p. 4. DOI: 10.3390/software4010004. URL: https://doi.org/10.3390/software4010004.

[50] Yang Liu and Jiankun Li. “Runtime Verification-Based Safe MARL for optimized Safety policy generation for Multi-Robot systems”. In: *Big Data and Cognitive Computing* 8.5 (May 16, 2024), p. 49. DOI: 10.3390/bdcc8050049. URL: https://doi.org/10.3390/bdcc8050049.

[51] Nathan Burow et al. “Control-Flow integrity”. In: *ACM Computing Surveys* 50.1 (Apr. 4, 2017), pp. 1–33. DOI: 10.1145/3054924. URL: https://doi.org/10.1145/3054924.

[52] Nicolo Maunero, Paolo Prinetto, and Gianluca Roascio. "CFI: Control Flow Integrity or Control Flow Interruption?" In: *IEEE Explore* 7 (Sept. 1, 2019), pp. 1–6. DOI: 10.1109/ewdts.2019.8884464. URL: https://doi.org/10.1109/ewdts.2019.8884464.

[53] John Criswell et al. "KCOFI: Complete Control-Flow Integrity for Commodity Operating system kernels". In: *IEEE Symposium on Security and Privacy* (May 1, 2014). DOI: 10.1109/sp.2014.26. URL: https://doi.org/10.1109/sp.2014.26.

[54] Christoph Spang et al. "DEXIE - an IoT-Class hardware monitor for Real-Time Fine-Grained Control-Flow integrity". In: *Journal of Signal Processing Systems (pp. 739–752)* 94.7 (Jan. 6, 2022). DOI: 10.1007/s11265-021-01732-5. URL: https://doi.org/10.1007/s11265-021-01732-5.

[55] Emanuele Parisi et al. "TitanCFI: Toward Enforcing Control-Flow Integrity in the Root -of- Trust". In: *IEEE* (Mar. 25, 2024), pp. 1–6. DOI: 10.23919/date58400.2024.10546873. URL: https://doi.org/10.23919/date58400.2024.10546873.

[56] H. Alotaibi and H. Zedan. "Runtime verification of safety properties in multi-agents systems". In: *IEEE Explore* (Nov. 1, 2010), pp. 356–362. DOI: 10.1109/isda.2010.5687238. URL: https://doi.org/10.1109/isda.2010.5687238.

[57] Said Daoudagh et al. "DAEMON: A Domain-Based Monitoring Ontology for IoT Systems". In: *SN Computer Science* 4.5 (Aug. 2023), p. 632. ISSN: 2661-8907. DOI: 10.1007/s42979-023-01975-y.

[58] Yunus Sabri Kirca et al. "Runtime verification for anomaly detection of robotic systems security". In: *Machines* 11.2 (Jan. 25, 2023), p. 166. DOI: 10.3390/machines11020166. URL: https://doi.org/10.3390/machines11020166.

[60] Venesa Watson et al. “Designing Trustworthy Monitoring Systems: Forensic Readiness for Safety and Security”. In: *IEEE Explore* (Oct. 1, 2018), pp. 155–160. DOI: 10.1109/icrms.2018.00038. URL: https://doi.org/10.1109/icrms.2018.00038.

[61] Orianna DeMasi, Taghrid Samak, and David H. Bailey. “Identifying HPC codes via performance logs and machine learning”. In: *ACM Digital Library* 13 (June 18, 2013), pp. 23–30. DOI: 10.1145/2465808.2465812. URL: https://doi.org/10.1145/2465808.2465812.

[62] Min Du et al. “DeepLog”. In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Oct. 27, 2017), pp. 1285–1298. DOI: 10.1145/3133956.3134015. URL: https://doi.org/10.1145/3133956.3134015.

[63] Obinna Johnphill et al. “An intelligent approach to automated operating systems log analysis for enhanced security”. In: *Information* 15.10 (), p. 657. DOI: 10.3390/info15100657.

[64] Edward Chuah et al. “Diagnosing the root-causes of failures from cluster log files”. In: *IEEE Explore* (Dec. 1, 2010), pp. 1–10. DOI: 10.1109/hipc.2010.5713159. URL: https://doi.org/10.1109/hipc.2010.5713159.

[66] Do Duc Anh Nguyen et al. “A Robust Approach for the Detection and Prevention of Conflicts in I2NSF Security Policies 2024”. en. In: ResearchGate, Oct. 2024. DOI: 10.1109/NOMS56928.2023.10154304. URL: https://www.naturl.link/WzVous.

[67] Andreas Vogelsang. “Feature Dependencies in Automotive Software Systems: Extent, Awareness, and Refactoring”. In: *Journal of Systems and Software* 160 (Nov. 2019), p. 110458. DOI: 10.1016/j.jss.2019.110458.

[69] Ashvin Goel et al. “Forensix: A Robust, High-Performance Reconstruction System”. en. In: *ResearchGate*. ICDCS 2005 Workshops, 2005. DOI: 10.1109/ICDCSW.2005.62. URL: https://www.researchgate.net/publication/220849276_Forensix_A_Robust_High-Performance_Reconstruction_System.

[70] Bogdan Groza et al. “Car-to-Smartphone Interactions: Experimental Setup, Risk Analysis and Security Technologies”. In: *5th International Workshop on Critical Automotive Applications: Robustness & Safety*. Naples, Italy: HAL Science, Sept. 2019. URL: https://hal.science/hal-02308417.

[72] Paul Doedens et al. “Safety and Security: A Delicate Balance”. In: Springer Nature, Dec. 2024, pp. 237–263. ISBN: 978-3-031-61223-7. DOI: 10.1007/978-3-031-61224-4_11.

[74] Emilia Cioroaica, Smruti Kar, and Ioannis Sorokos. “Comparison of Safety and Security Analysis Techniques”. In: CISIS 2021 and ICEUTE 2021, Jan. 2022, pp. 234–242. ISBN: 978-3-030-87871-9. DOI: 10.1007/978-3-030-87872-6_23.

[75] Stefano M. Nicoletti et al. “Model-based joint analysis of safety and security:Survey and identification of gaps”. In: *Computer Science Review* 50 (2023), p. 100597. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2023.100597. URL: https://www.sciencedirect.com/science/article/pii/S1574013723000643.

[78] Stephanie Drzevitzky. “Proof-Carrying Hardware: Runtime Formal Verification for Secure Dynamic Reconfiguration”. In: *IEEE Explore* (Aug. 1, 2010), pp. 255–258. DOI: 10.1109/fpl.2010.59. URL: https://doi.org/10.1109/fpl.2010.59.

[79] Ge Zhou et al. “Optimizing Monitor Code Based on Patterns in Runtime Verification”. In: *IEEE Explore* (July 1, 2017), pp. 348–354. DOI: 10.1109/qrs-c.2017.65. URL: https://doi.org/10.1109/qrs-c.2017.65.

[80] Zijiang Yang, Shiyong Lu, and Ping Yang. "Runtime Security Verification for Itinerary-Driven Mobile Agents". In: *IEEE* (Sept. 1, 2006), pp. 177–186. DOI: 10.1109/dasc.2006.42. URL: https://doi.org/10.1109/dasc.2006.42.

[81] Ruan De Clercq et al. "SOFIA: Software and control flow integrity architecture". In: *Computers & Security* 68 (Apr. 3, 2017), pp. 16–35. DOI: 10.1016/j.cose.2017.03.013. URL: https://doi.org/10.1016/j.cose.2017.03.013.

[82] Qiang Hao et al. "A Hardware Security-Monitoring Architecture based on data integrity and control flow integrity for embedded systems". In: *Applied Sciences* 12.15 (Aug. 1, 2022), p. 7750. DOI: 10.3390/app12157750. URL: https://doi.org/10.3390/app12157750.

[83] Thomas Chamelot, Damien Courousse, and Karine Heydemann. "SCI-FI: Control Signal, Code, and Control Flow Integrity against Fault Injection Attacks". In: *Design, Automation & Test in Europe Conference & Exhibition, 2015* (Mar. 14, 2022), pp. 556–559. DOI: 10.23919/date54114.2022.9774685. URL: https://doi.org/10.23919/date54114.2022.9774685.

[84] Zhou Zhou et al. "Jasmine: A Tool for Model-Driven Runtime Verification with UML Behavioral Models". In: *IEEE Explore* 3308 (Dec. 1, 2008), pp. 487–490. DOI: 10.1109/hase.2008.62. URL: https://doi.org/10.1109/hase.2008.62.

[85] Zhi-Bing Wang et al. "A Framework on Runtime Verification for Software Behavior". In: *IEEE Explore* (Jan. 1, 2012), pp. 20–23. DOI: 10.1109/isdea.2012.670. URL: https://doi.org/10.1109/isdea.2012.670.

[86] Antonio Coronato and Alessandro Testa. "Runtime verification of location-dependent correctness and security properties in Ambient Intelligence applications". In: *IEEE Explore* (Feb. 1, 2011), pp. 153–160. DOI: 10.1109/bcfic-riga.2011.5733231. URL: https://doi.org/10.1109/bcfic-riga.2011.5733231.

Books

[13] Siwar Kriaa. "Joint safety and security modeling for risk assessment in cyber physical systems". Theses. Université Paris Saclay (COmUE), Mar. 2016. URL: https://theses.hal.science/tel-01318118.

[47] Théo Serru. "Analyse de la cybersécurité des systèmes cyber-physiques par approche fondée sur les modèles : impact des cyberattaques sur la sécurité avec AltaRica". 2023CYUN1210. PhD thesis. 2023. URL: http://www.theses.fr/2023CYUN1210/document.

[59] Benjamin Schiller. "Graph-based Analysis of Dynamic Systems". PhD thesis. Technische Universität Dresden, 2017.

[65] Mohammad El Musleh. "Transformation of UML State Machine Diagram into Graph Database to Generate Test Cases". PhD thesis. Uppsala University, 2020. URL: https://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-426060.

[71] Megha Quamara. "An approach to co-design and analysis of safety and security for three-layered system modeling : models, formalisms, and tool support". eng. PhD thesis. École doctorale Mathématiques, informatique et télécommunications, 2022.

[73] Ahmad Nasser. "Securing Safety Critical Automotive Systems". PhD thesis. ResearchGate, June 2019.

Other

[13] Siwar Kriaa. "Joint safety and security modeling for risk assessment in cyber physical systems". Theses. Université Paris Saclay (COmUE), Mar. 2016. URL: https://theses.hal.science/tel-01318118.

[20] Guy Korland. *Knowledge graph vs vector database: Which one to choose?* Nov. 2024. URL: https://www.falkordb.com/blog/knowledge-graph-vs-vector-database/#:~:text=Knowledge%20Graphs%20and%20Vector%20Databases,for%20data%20querying%20and%20extraction..

[21] Wikipedia contributors. *Large language model*. Feb. 15, 2025. URL: https://en.wikipedia.org/wiki/Large_language_model.

[22] Kai Sun et al. *Head-to-Tail: How Knowledgeable are Large Language Models (LLMs)? A.K.A. Will LLMs Replace Knowledge Graphs?* 2024. arXiv: 2308.10168 [cs.CL]. URL: https://arxiv.org/abs/2308.10168.

[23] *Knowledge graph vs vector database: Which one to choose?* Section: Blog. July 1, 2024. URL: https://www.falkordb.com/blog/knowledge-graph-vs-vector-database/ (visited on 02/17/2025).

[28] Wikipedia contributors. *Concept map*. Dec. 2024. URL: https://en.wikipedia.org/wiki/Concept_map.

[29] Wikipedia contributors. *Conceptual graph*. July 13, 2024. URL: https://en.wikipedia.org/wiki/Conceptual_graph.

[30] Wikipedia contributors. *Sequence diagram*. en. Mar. 2025. URL: https://en.wikipedia.org/wiki/Sequence_diagram.

[31] *UML diagrams: A practical guide for software professionals*. en. URL: https://d1gmfi7dd8yhn4.cloudfront.net/learn/software-development/uml-diagrams-guide/.

[32] Andrew Disney. *How To Choose A Graph Database: We Compare 6 Favorites*. URL: https://cambridge-intelligence.com/choosing-graph-database/.

[33] *Graph database - Wikipedia*. URL: https://en.wikipedia.org/wiki/Graph_database.

[34] Reddit blog. *What graph DB to use for knowledge graphs for our LLM : r/dataengineering*. URL: https://www.reddit.com/r/dataengineering/comments/192abwa/what_graph_db_to_use_for_knowledge_graphs_for_our/.

[35] *ArangoDB vs. NebulaGraph vs. Neo4j Comparison*. URL: https://db-engines.com/en/system/ArangoDB%3BNebulaGraph%3BNeo4j.

[36] Neo4j. *APOC User Guide 4.1 - APOC Extended Documentation*. URL: https://neo4j.com/labs/apoc/4.1/.

[37] LangChain. *How to construct knowledge graphs — LangChain*. URL: https://python.langchain.com/docs/how_to/graph_constructing/.

[40] Gwendal Daniel. *UMLtoGraphDB: Mapping UML to NoSQL Graph Databases*. en-US. Nov. 2016. URL: https://modeling-languages.com/uml-to-nosql-graph-database/.

[47] Théo Serru. "Analyse de la cybersécurité des systèmes cyber-physiques par approche fondée sur les modèles : impact des cyberattaques sur la sécurité avec AltaRica". 2023CYUN1210. PhD thesis. 2023. URL: http://www.theses.fr/2023CYUN1210/document.

[59] Benjamin Schiller. "Graph-based Analysis of Dynamic Systems". PhD thesis. Technische Universität Dresden, 2017.

[65] Mohammad El Musleh. "Transformation of UML State Machine Diagram into Graph Database to Generate Test Cases". PhD thesis. Uppsala University, 2020. URL: https://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-426060.

[68] HackersCorner. *Hackers Corner Tactics, Techniques and Procedures*. URL: https://www.radware.com/getattachment/Security/Hackers-Corner/2218/HackersCorner-TTPsDDoS-FINAL-V3.pdf.aspx/?lang=en-US.

[71] Megha Quamara. “An approach to co-design and analysis of safety and security for three-layered system modeling : models, formalisms, and tool support”. eng. PhD thesis. École doctorale Mathématiques, informatique et télécommunications, 2022.

[73] Ahmad Nasser. “Securing Safety Critical Automotive Systems”. PhD thesis. ResearchGate, June 2019.

[76] Mihai Maruseac. *Testing graph databases*. URL: https://mihai.page/testing-graph-databases/.

[77] *What is a graph database - Getting Started*. URL: https://neo4j.com/docs/getting-started/graph-database/.