CONNECT
CCAM TRUST & RESILIENCE

D3.3

CONNECT Trust & Risk Assessment and CAD Twinning Framework (Final Version)

Project number:	101069688
Project acronym:	**CONNECT**
Project title:	Continuous and Efficient Cooperative Trust Management for Resilient CCAM
Project Start Date:	1st September, 2022
Duration:	36 months
Programme:	HORIZON-CL5-2021-D6-01-04
Deliverable Type:	Other
Reference Number:	D6-01-04 / D3.3 / 1.0
Workpackage:	WP 3
Due Date:	28th February, 2025
Actual Submission Date:	18th June, 2025
Responsible Organisation:	Ulm University
Editor:	Artur Hermann
Dissemination Level:	PU
Revision:	1.0
Abstract:	This deliverable describes the final version of the standalone trust assessment framework (TAF), the federated TAF and the digital twin version of the TAF (TAF-DT). For the standalone TAF, the federated TAF and the TAF-DT, the final architecture, implementation details, and evaluation results are provided. In addition to that, the approaches for trust quantification, the approach for calculating the required trustworthiness level (RTL), and details about the risk assessment framework are provided.
Keywords:	Trust Assessment Framework, Risk Assessment Framework, Digital Twin

Funded by the European Union

The CONNECT project is funded by the European Union under grant agreement No. 101069688. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or CINEA. Neither the European Union nor the granting authority can be held responsible for them.

Editor

Artur Hermann (UULM)

Contributors (ordered according to beneficiary numbers)

Nikos Fotos, Anna Angelogianni, Thanassis Giannetsos (UBITECH)
Ana Petrovska, Koffi Ismael Ouattara, Ioannis Krontiris, Theo Dimitrakos (HUAWEI)
Artur Hermann, Nataša Trkulja, Benjamin Erb, Frank Kargl (UULM)
Guillaume Mockly, Antonio Kung (Trialog)
Anderson Ramon Ferraz de Lucena, Alexander Kiening (DENSO)
Francesca Bassi, Ines Ben Jemaa (IRTSX)

Disclaimer

Executive Summary

This deliverable follows up on Deliverable D3.1 ”Architectural Specification of CONNECT Trust Assessment Framework”, which outlined the core ideas of CONNECT and its trust assessment framework and Deliverable D3.2 ”CONNECT Trust & Risk Assessment and CAD Twinning Framework (Initial Version)”, which provided a fine-grained architecture description of the Trust Assessment Framework (TAF). Deliverable D3.3 ”CONNECT Trust & Risk Assessment and CAD Twinning Framework (Final Version)” contains all material from D3.2, however in extended, augmented, and updated form.

In addition, this deliverable is a follow up of Deliverable D2.2 ”Operational Landscape, Requirements and Reference Architecture - Final Version”, which provided an extended version of the overall architecture of CONNECT and includes the Trust Assessment Framework as one of its main components, and Deliverable D6.1 ”Integrated Framework (First release) and Use Case Analysis”, which provided a refined version of the overall architecture focusing on interfaces and message exchanges.

Deliverable D3.3 describes three versions of the TAF: The standalone TAF, the federated TAF and the digital twin version of the TAF (TAF-DT). The Deliverable presents the final version of the concepts, the architecture, the implementation details, the evaluation results and all related aspects of the three TAF versions to run successfully. The main contributions of this deliverable are:

1. A fine-grained architecture description of the standalone TAF and federated TAF.
2. A fine-grained architecture description of the TAF-DT, i.e., a digital twin of a standalone TAF.
3. A detailed interface specification and implementation description of the standalone TAF, federated TAF, and TAF-DT.
4. A detailed description of the final version of the Trustworthiness Level Expression Engine (TLEE), which calculates a trust opinion for a proposition based on the evaluation of complex trust networks.
5. A detailed specification of the CONNECT Risk Assessment Framework which extends the TAF towards a larger framework that can provide trust evidence to the TAF.
6. A detailed specification of quantification algorithms to calculate trust opinions for three different trust sources.
7. A detailed description of the approach to calculate the Required Trustworthiness Level (RTL) based on a TARA.
8. Detailed evaluations of the TAF and all related components.

With these contributions, the Deliverable D3.3 presents the final status of the standalone TAF, federated TAF and TAF-DT. This also marks the successful completion of Work Package 3 and concludes major activities of the work package except for minor activities and provisioning of the open-sourced TAF.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Scope and Purpose

This deliverable follows up on Deliverable D3.1 ”Architectural Specification of CONNECT Trust Assessment Framework”, which outlined the core ideas of CONNECT and its trust assessment framework and Deliverable D3.2 ”CONNECT Trust & Risk Assessment and CAD Twinning Framework (Initial Version)”, which provided a fine-grained architecture description of the Trust Assessment Framework (TAF). Deliverable D3.3 ”CONNECT Trust & Risk Assessment and CAD Twinning Framework (Final Version)” contains all material from D3.2, however in extended, augmented, and updated form.

In addition, this deliverable is a follow up of Deliverable D2.2 ”Operational Landscape, Requirements and Reference Architecture - Final Version”, which provided an extended version of the overall architecture of CONNECT and includes the Trust Assessment Framework as one of its main components, and Deliverable D6.1 ”Integrated Framework (First release) and Use Case Analysis”, which provided a refined version of the overall architecture focusing on interfaces and message exchanges.

Deliverable D3.3 describes three versions of the TAF: The standalone TAF, the federated TAF and the digital twin version of the TAF (TAF-DT). The Deliverable presents the final version of the concepts, the architecture, the implementation details, the evaluation results and all related aspects of the three TAF versions to run successfully.

The main contributions in this deliverable are:

1. A fine-grained architecture description of the standalone TAF and federated TAF.
2. A fine-grained architecture description of the TAF-DT, i.e., a digital twin of a standalone TAF.
3. A detailed interface specification and implementation description of the standalone TAF, federated TAF, and TAF-DT.
4. A detailed description of the final version of the Trustworthiness Level Expression Engine (TLEE), which calculates a trust opinion for a proposition based on the evaluation of complex trust networks.
5. A detailed specification of the CONNECT Risk Assessment Framework which extends the TAF towards a larger framework that can provide trust evidence to the TAF.

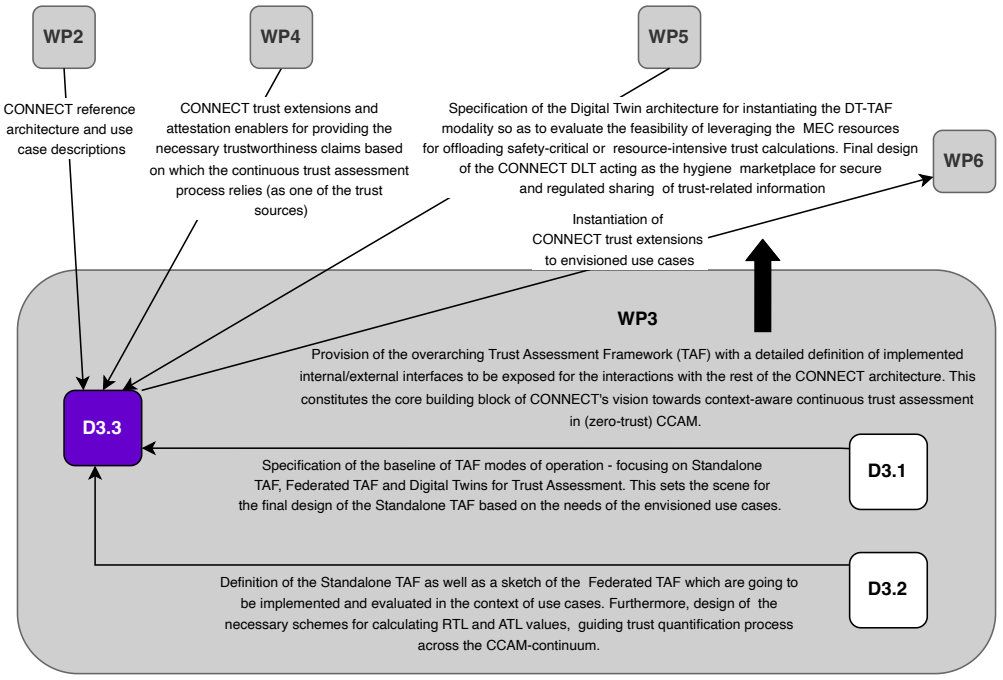


Figure 1.1: Relation of D3.3 with other WPs and Deliverables.

6. A detailed specification of quantification algorithms to calculate trust opinions for three different trust sources.

7. A detailed description of the approach to calculate the Required Trustworthiness Level (RTL) based on a TARA.

8. Detailed evaluations of the TAF and all related components.

1.2 Relationship with other WPs & Deliverables

With the documentation of the research road-map towards the design of a holistic Trust Assessment Framework (TAF), capable of quantifying the trustworthiness of individual CCAM entities (as detailed in D3.1), and the documentation of the first version of the standalone TAF including its internal building blocks, operations and interfaces to be exposed for enabling the interactions with the other CONNECT framework components (as detailed in D3.2), this deliverable proceeds with the final version of the TAF architecture, its interfaces and all internal building blocks.

In this context, Figure 1.1 depicts the direct and indirect relationships to other tasks and Work Packages (WPs). First, it distills the mechanisms based on which the **Required Trust Level (RTL) and Actual Trust Level (ATL) are defined** for guiding the **trust quantification process across the entire CCAM continuum**. Recall that CONNECT extends the standalone vehicle domain to safe and security solutions distributed from Vehicle to MEC and Cloud facilities so as to support the envisioned automation of *connected vehicles*: This vision is enabled by the vehicle's communication with other entities formulating the Vehicle-to-everything (V2X) landscape. The resulting "*CCAM continuum*" paradigm seeks to seamlessly and securely combine the available hardware and software (from the in-vehicle sensors to the MEC virtualized infrastructure) to

support the deployment and operation of certified (using ISO/SAE 21434 [1]) CCAM functions. Whereas the collective consideration (treatment) of the continuum resources presents opportunities for increased performance and a higher degree of automation, the individual Software (SW) and Hardware (HW) infrastructure may exhibit diverse yet dynamic trust states; and when those infrastructures are brought together to make-up the continuum strong trust assessment mechanisms need to also be deployed for asserting to the correct state of all these functional assets.

This is enabled through the design of the final variant of the TAF (**Standalone TAF** capable of analyzing the trustworthiness levels of entities in isolation) coupled with the (preliminary) definition of the necessary interfaces that need to be exposed for accommodating the interactions with the other CONNECT components. Especially, as it pertains to the trusted computing capabilities (offered by the CONNECT Trusted Computing Base developed in the context of WP4) towards the runtime integrity verification of all (SW and HW) elements comprising a CCAM function chain to be assessed. This functionality, together with those libraries offered for secure software upgrade and/or state migration (as reaction policies to the change of a node's trust level), comprise CONNECT's TCB capabilities for supporting the operations of the Trust Assessment Framework: *All attestation attributes constitute one of the trustworthiness sources based on which the TAF (as the core WP3 artifact) calculates the Actual Trust Level (ATL) of a (SW and/or HW) element which leads to the trust state characterization of the target node*.

Overall, the outcome of this deliverable serves as the final implementation milestone of CONNECT's efforts towards the design of an overarching trust assessment framework (TAF) by consolidating the core architecture of the standalone TAF which enables to model and reason about the complex set of trust relationships that need to be assessed in the CCAM ecosystem (based on our envisioned use cases). All these capabilities set the scene for the detailed experimentation and evaluation activities of all these trust extensions in the context of the envisioned use cases on Intersection Movement Assistance, Collaborative Cruise Control and Slow-Moving Traffic Detection (WP6).

1.3 Future Work

In the context of the CONNECT project and work package 3, our main focus was on the conceptual design, development, and implementation of the the trust assessment framework (TAF) in its three incarnations. CONNECT achieved to specify and implement almost all aspects of the TAF that were envisioned from the beginning of the project. Only for the federated TAF and the digital twin version of the TAF (TAF-DT) were there minor deviations from the original plan which can serve as inspiration for future work.

Federated TAF In the previous deliverables, we developed three approaches of a federated TAF, in which different types of information are exchanged between the individual TAF instances. The individual approaches of the federated TAF are described in this deliverable in Chapter 5. It was initially envisioned that the first two approaches would be implemented within the context of this work package while the third approach was left to future work. However, none of these approaches were appropriate for the use cases of the CONNECT project, where a federated TAF is used. As a result, a fourth approach of a federated TAF was created and implemented. Therefore, only this fourth approach of the federated TAF was implemented in this project while the other three are purely conceptual. Nevertheless, the other approaches may provide advantages in different scenarios and can be considered and evaluated in future work.

TAF-DT In the previous deliverables, we described two main functionalities that the TAF-DT should provide. The first functionality is state replication. State replication implies being able to maintain a communication channel between the vehicle and the digital twin and to verify that changes in the trust model in the vehicle are transferred to the digital twin through this channel. The second functionality is the remote execution of computations. Remote execution of computations implies that we are able to send a trust assessment request (TAR) to the twin and obtain an actual trustworthiness level (ATL) or a trust decision as a response.

The latter was realized. However, while the state replication functionality was specified as part of this project and is described in detail in Chapter 7, time constraints and delays within the project prevented an implementation of the state replication functionality. However, a preliminary evaluation of the performance of the TAF in a Digital Twin setup was conducted and is described in Chapter 7. The state replication functionality could be implemented in future work and use cases that benefit from such a remote execution.

1.4 Deliverable Structure

This deliverable, D3.3 “CONNECT. Connect trust & risk assessment and cad twinning framework (final version)” presents the final version of the trust assessment framework (TAF) and its evaluation, as presented in Chapter 3. To this end, the chapter outlines the requirements, assumptions, and design goals of the TAF. Building on these foundations, the final architecture of both the standalone and federated versions of the TAF is presented. In addition, the internal and external interfaces of the TAF are explained, including a detailed description of the specified APIs. The enabling technologies used for the implementation of the TAF, along with its key features, are also discussed in Chapter 3. Finally, the chapter provides an evaluation of the TAF. For this purpose, each individual trust requirement is listed along with its corresponding KPI. For each trust requirement it is assessed whether the corresponding KPI is fulfilled.

Next, Chapter 4 describes the final version of the Trustworthiness Level Expression Engine (TLEE). This chapter begins with an introduction to the high-level architecture and the functional specifications of the TLEE and its core components. It then presents the mathematical foundations of trust modeling, including established trust discounting and fusion operators, as well as a newly developed operator tailored to the specific needs of the CONNECT project. The chapter also outlines the internal interface between the TLEE and the TAM, describing how the TLEE interacts with other components of the TAF. It concludes with practical insights from the development of the TLEE.

Chapter 5 focuses on the different approaches of the federated TAF. The federated TAF will enable two TAFs to interact with each other. We distinguish four approaches of such an interaction: an interaction where one TAF requests a single trust source in a remote TAF (case 1), sending of a TAR to retrieve an ATL from by one standalone TAF to another standalone TAF (case 2), the case where two TAFs interact to jointly calculate an ATL where trust relationships span linked trust models residing in different TAFs (case 3), and finally the case the federated TAF architecture is leveraged to assess the same trust object by two TAF instances (case 4).

This deliverable then continuous in Chapter 6, where the performance of the TAF inside a TEE is evaluated. This chapter begins by motivating the need for executing the TAF inside a TEE. It then presents the experimental setup, discusses the evaluation results, and provides the conclusions derived form these results.

Next, Chapter 7 presents the Digital Twin version of the TAF (TAF-DT). The chapter begins with a motivation of the digital twin, followed by a description of the architecture of the TAF-DT. The chapter then outlines the evaluation setup and presents the corresponding results. It concludes with an outlook and further works.

This deliverable then continuous with Chapter 8 and the Risk Assessment Framework which is a major provider of trustworthiness evidence and also structural information to build trust models to the TAFs. The chapter begins with an overview and a functional specification of the Risk Assessment Framework, followed by a description of its internal architecture. The chapter also explains how to streamline the TARA process through automated attack path calculation. Additionally, it describes the transition from TARA to RTL and specifies the implemented interfaces. The chapter concludes with a description to reinforce RTL calculation by abstracting the risk quantification engine.

Chapter 9 continues with a description of algorithms for quantifying trust in the form of a subjective logic opinion based on received evidence. The particular challenge here is the translation of heterogeneous trust evidence into subjective logic opinions accessible to the TAF. The chapter covers trust quantification for three different trust sources: the Misbehavior Detection System (MBD), the Attestation and Integrity Verification (AIV), and the Trustworthiness Claims Handler (TCH). For each trust source, a dedicated quantification algorithm is described, accompanied by a detailed example to illustrate its application.

With the previous chapters in place, we have described all required elements for calculation of Actual Trustworthiness Levels (ATLs). What is missing is the calculation of Required Trustworthiness Levels (RTLs). This is considered in Chapter 10 where the final methodology is presented which calculates the RTL based on risk analysis through a TARA. The chapter starts with an overview of the methodology to calculate the RTL, followed by the introducing of a use case application. The methodology is then applied to this use case to illustrate the approach in detail.

Chapter 11 concludes this final version of the deliverable by summarizing the key contributions and presenting the final version of the Trust Assessment Framework (TAF). It reflects on the achieved objectives and outlines the overall outcomes of the work.

Chapter 2

Trust Assessment Terms and Definitions

This chapter summarizes terms and definitions related to trust assessment in general, as well as the the Trust Assessment Framework (TAF) itself. This vocabulary will act as the reference resource, throughout all project activities, so that all stakeholders can have a common understanding of the characteristics that could be used to describe the trustworthiness of a data item or node.

While this vocabulary primarily targets documentation of terms capturing the mode of operation of a TAF in the context of Autonomous Vehicles (AVs), it is intended for use horizontally in the information technology domain and all domains where Subjective Logic is used as the foundation of trust reasoning.

ATL (Actual Trustworthiness Level) The ATL reflects the result of an evaluation of a specific (atomic or complex) proposition for a specific scope provided by the TLEE. It quantifies the extent to which a certain node or data can be considered trustworthy based on the available evidence.

ATO (Atomic Trust Opinion) An ATO is a subjective logic opinion created by the TSM when quantifying one specific type of a Trust Source based on trustworthiness evidence. An ATO is formed by a Trust Source in the context of a single trust relationship.

PROP (Proposition) A *proposition* is a logic statement about some phenomenon of interest whose level of trustworthiness we are interested in assessing. A proposition could be 1) atomic—a proposition whose truth or trustworthiness can be directly assessed, or 2) composite, comprising of multiple atomic propositions. The proposition describes the fulfillment of the properties in relation to *data* or *nodes*.

RTL (Required Trustworthiness Level) The RTL reflects the amount of trustworthiness of a node or data that an application considers required in order to characterize this object as trusted and rely on its output during its execution.

TA (Trust Assessment Manager) The TAM is a component inside the TAF which orchestrates the overall process of trust assessment.

TAF (Trust Assessment Framework) A software framework which, given a trust model for a specific function running inside a CCAM system, is able to evaluate trust sources for trustworthiness evidence and evaluate propositions within the trust model to obtain their ATLs.

Optionally, also an RTL can be evaluated and trust decisions can be taken and communicated to the application.

TAF-API (TAF - Application Programming Interface) Application Programming Interface by which the TAF and its functionality can be accessed from an application.

TAF-DT (TAF - Digital Twin) The digital twin allows a vehicle to replicate its TAF including among others its TMs and TSs within a MEC. This allows a vehicle to outsource trust assessment to a MEC where the TAF-DT is expected to run inside a TEE so that confidentiality and integrity of its data and state can be protected from the MEC.

TDE (Trust Decision Engine) The TDE is a component inside the TAF which performs the last step before an output is provided to the application that requested trustworthiness assessment. The TDE either forwards the Actual Trustworthiness Level (ATL) calculated by the TLEE along to the application or outputs a Trust Decision (TD). A TD is created after comparing the ATL to the Required Trust Level (RTL) in a predetermined manner. Whether the output of the TAF is an ATL or a TD depends on the needs of the application requesting trustworthiness assessment.

TLEE (Trustworthiness Level Expression Engine) The TLEE is a component inside the TAF that calculates the level of trustworthiness for a concrete trust model instance and the proposition that needs to be evaluated. The TLEE uses the numerical values of the Atomic Trust Opinions computed based on the trust sources collected in the TSM. Based on these inputs, the TLEE calculates an ATL and provides it to the TA. The TLEE encapsulates most of the Subjective Logic formalism.

TM (Trust Model) Trust model is a graph-based model which is built on top of a system model which represents all components and data needed to perform a certain function. Components represented either create, transmit, process, relay, and receive the data used as input to a function. The vertices in a trust model correspond to an abstraction called trust objects, and the edges in a trust model correspond to trust relationships between a pair of trust objects. The trust model also encompasses a list of trust sources used to build up / quantify trust relationships by providing atomic trust opinions. The trust model is a main input to the TMM and the TLEE. Since trust is a directional relationship between two trust objects and it is always in relation to a concrete property or scope, then as part of the trust model, there can be multiple trust relationships between the same two trust objects, depending on different properties of the trust relationship, or the scope of the trust relationship.

TMM (Trust Model Manager) The TMM is a component inside the TAF responsible for storing trust models and making them accessible for TLEE and other purposes. In particular, it is able to provide TMs for specific functions running in a CCAM system, also considering different scopes that TMs may cover.

TO (Trust Opinion) The Trust Opinion is a numeric value which represents the trustworthiness of an entity as assessed by another entity based on relevant entity. Within the scope of CONNECT a Trust Opinion is expressed in form of a Subjective Logic binomial opinion ω_B^A where A is the entity assessing trustworthiness and B is the entity whose trustworthiness is being assessed..

Trust Objects Trust objects are core building blocks of a trust model. They represent entities that assess trust or for which trust is assessed. The trust objects are identified 1) based

on the *components* from the component diagram and 2) the *atomic propositions* (i.e., the properties about data or nodes for which trust assessment is conducted).

Trust Relationships A trust relationship is a directional relationship between two trust objects that are called trustor (i.e.,, the “thinking entity”, the assessor) and a trustee (one who is trusted). The trust relationship is always in relation to a concrete property and a certain scope.

Trustworthiness Tier Trustworthiness Tier is a categorization of the levels of trustworthiness which may be assigned by the TAF (or another Verifier that appraises the attestation results of a TC and communicates them to the TAF) to a specific Trustworthiness Claim.

Trustworthiness Claims A Trustworthiness Claim (TC) is a form of node-centric ATO provided by a TS and contains a specific data quote used for conveying the information needed by the TAF to make a decision on the trust level of an object. The TC is usually produced (by the Attester) so as to provide trustworthiness evidence (cf. “Trust Source”) that can be used for appraising the trustworthiness level of the Attester in a *measurable* and *verifiable* manner [**?**]. Measurable reflects the ability of the TAF to assess an attribute of the Attester against a pre-defined metric while verifiability highlights the need for all claims to have integrity, freshness and to be provably & non-reputably bound to the identity of the original Attester. Examples sets of TCs might include (among other attributes) evidence on system properties including: (i) **integrity** in the context that all transited devices (e.g., ECUs) have booted with known hardware and firmware; (ii) **safety** meaning that all transited devices are from a set of vendors and are running certified software applications containing the latest patches and (iii) **communication integrity**. For a more detailed list of possible system (behavioural) evidence that can be appraised as part of TCs, please refer to Section **??**.

TS (Trust Source) A TS manages one or multiple trustworthiness evidence inside the TAF. On request of the TA, it quantifies the trustworthiness of a trustee based on a specific type of evidence in form of an atomic trust opinion.

TSM (Trust Source Manager) The TSM is a component inside the TAF responsible for handling all available TS inside a TAF and to establish and integrate new TSs dynamically through a plugin interface.

Chapter 3

Trust Assessment Framework

3.1 Overview

CONNECT's Trust Assessment Framework (TAF) is a software system assessing the level of trustworthiness of entities with respect to a specific scope. The scope can be a property, such as the *integrity* of data. In this case, the TAF is assessing how trustworthy it is that the data has not been compromised. That said, the TAF is conceptually a reactive software system that receives incoming messages that provide updates about the trustworthiness of surrounding environment or relevant entities. These updates are then processed and integrated into an internal representation of trust sources, trust opinions, and relationships as part of trust models. At the same time, the TAF provides a service to other applications and entities by providing access to its trustworthiness assessment based on these internal representations and derived information. As a generic software solution, the TAF is deployable onto different runtime environments, in particular onto vehicles for in-vehicular and V2X use cases and onto MEC servers for stationary V2X usage in infrastructure.

3.1.1 Standalone Trust Assessment Framework

A Standalone Trust Assessment Framework is a TAF that is executed on a vehicle or a mobile edge computer (MEC) and quantifies received evidence *locally*. The trust assessment is executed within the TAF, based on information the TAF has received, quantified, and assessed *locally*. The standalone TAF does not incorporate trust information provided by another TAF and is thus independent from other TAFs. However, the entities providing information the TAF are not located in the TAF. These are external entities, such as a misbehavior detection system.

3.1.2 Federated Trust Assessment Framework

A federated TAF has the same capabilities as a standalone TAF, but in addition, it can also exchange trust information with other TAFs. The federated TAF could be used in scenarios, where a single node does not have access to the relevant evidence necessary to calculate a trust opinion. Thus, the federated TAF concept enhances the scope of a single, local TAF by exchanging trust information with other TAFs and this enables more comprehensive trust assessments to be made and involve multiple TAFs.

3.2 System Requirements and Specifications

Several requirements and specifications have been defined for the TAF in the course of the project. These requirements are divided into functional and non-functional requirements and are described below:

3.2.1 Functional Requirements

Functional requirements describe the behavior, features and functions that the TAF must provide to fulfill its intended purpose. The functional requirements for the TAF are described in the following:

Offloading of Trust Assessment: The TAF provides trust assessment as a service for co-located applications. By offloading the trust-related tasks to the TAF, an application can focus on its original tasks and rely on the TAF for any trust assessments.

Support for a Range of Trust Sources: A TAF needs to use evidence from a range of trust sources when assessing the trust levels. These trust sources can be local and/or remote, and they may require different communication styles and interaction patterns for collection.

Transforming Evidence into Trust Measures: Another requirement is a generic approach to let the TAF ingest arbitrary evidence that supports or refutes the trustworthiness of nodes or data and transform them into a common, internal trust representation that can be used for further inference in the STN, and eventually for decision making. At the same time, the internal representation must cope with fuzziness to reflect realistic assessments in face of uncertainty.

Dynamic Trust Modeling: A trust model requires a representation of the nodes involved, their relationships, methods for mapping incoming evidence to the trust relationships, and approaches for eventually deriving trustworthiness levels. While trust models are static in some cases, others are highly dynamic in the sense that they evolve, and the nodes and their relationships change over time.

Application Genericity: The TAF should be generic enough to be applicable in different scenarios, for example in in-vehicle networks or in inter-vehicle networks. Such a generalizable TAF would be much more widely applicable and would reduce or eliminate customization effort for each use case and could even be updated for use in a completely novel use case.

Configurability & Customizability: Based on different use cases and runtime environments (e.g., vehicles vs. MECs), the TAF requires configuration parameters to allow for certain trade-offs to address the requirements mentioned above. For instance, in the case of resource limitations or for use cases with many concurrently occurring entities, a TAF configuration should allow to trade accuracy for timeliness (e.g., by micro-batching updates under high load) or to trade lower response latencies for higher memory consumption (e.g., by extensively caching results). In addition, individual components of the TAF should be modularized so they can be swapped easily. This would allow for customized or optimized components for specific use cases or dedicated runtime environments.

3.2.2 Non-Functional Requirements

Non-functional requirements define quality attributes and constraints that the TAF must fulfill. The non-functional requirements for the TAF are described in the following:

Run-time performance: The TAF is intended for CCAM systems that are critical in terms of time and safety. Therefore, the TAF should be able to assess the trustworthiness of an entity, which can be a node or data item, within the specific time constraints set by the application. The reason for this is that the decisions of the CCAM system should be made based on the output of the TAF. Therefore, the TAF must be fast enough to not cause unacceptable delays in the decision-making process which could lead to safety issues.

Scalability: Both the scale of trust models and the scale of scenarios may vary a lot between different use cases the TAF is applied to. This means that the TAF needs to be able to successfully operate despite varying rates of incoming messages, varying numbers of applications that use the service of the TAF, varying levels of dynamicity and complexity of individual trust models, and varying numbers of entities that have to be assessed in these models. Hence, scalability is a key requirement for the TAF.

Efficiency: Given the critical functionality of the TAF, it is important that updates received by the TAF are reflected as fast as possible in the trustworthiness assessment calculated the TAF. As the actual worth of assessments substantially depends on the timeliness of their availability, low processing delays are mandatory. At the same time, the runtime environment of the TAF might be resource-constrained so that computational efficiency is vitally important.

Resilience: There are many ways to attack the TAF itself, resulting in the TAF providing incorrect output or no output at all. Such attacks could be aimed at changing the trust relationships, the trust sources considered, or the trust opinions between two entities. Therefore, the TAF should include mechanisms to be resilient against possible attacks on its operations.

Security: Given the critical functionality of the TAF, it is important that its processing and results are protected in terms of their integrity from malicious manipulation. Furthermore, confidentiality may be a requirement as far as the trust model and values may provide insights into vehicle and systems that should not be public.

3.2.3 Constraints and Assumptions

The prototype implementation of the TAF is a proof-of-concept system to showcase the feasibility of the TAF and its concepts. However, it is not intended to be a fully-fledged, production-ready platform. The prototype implementation is based on the following constraints and assumptions:

General Purpose Runtime Environments: Although being targeted for automotive environments, the prototype focuses on general-purpose runtime systems (i.e., commodity hardware with desktop/server CPUs) instead of highly specialized embedded systems. This choice eases rapid development and is also in line with the target architectures of MEC deployments.

Abstracted Communication Layer: The TAF communicates with a lot of different components, both within the same physical system and with other remote entities. In reality, these interactions are based on a plethora of different communication technologies, protocols, and message encodings. In order to streamline the prototype development, the prototype environment uses an abstraction layer that allows simple, message-based communication between all involved entities. Furthermore, a human-readable message encoding (JSON) has been chosen to simplify development, testing, and debugging.

3.3 System Design

3.3.1 System Overview

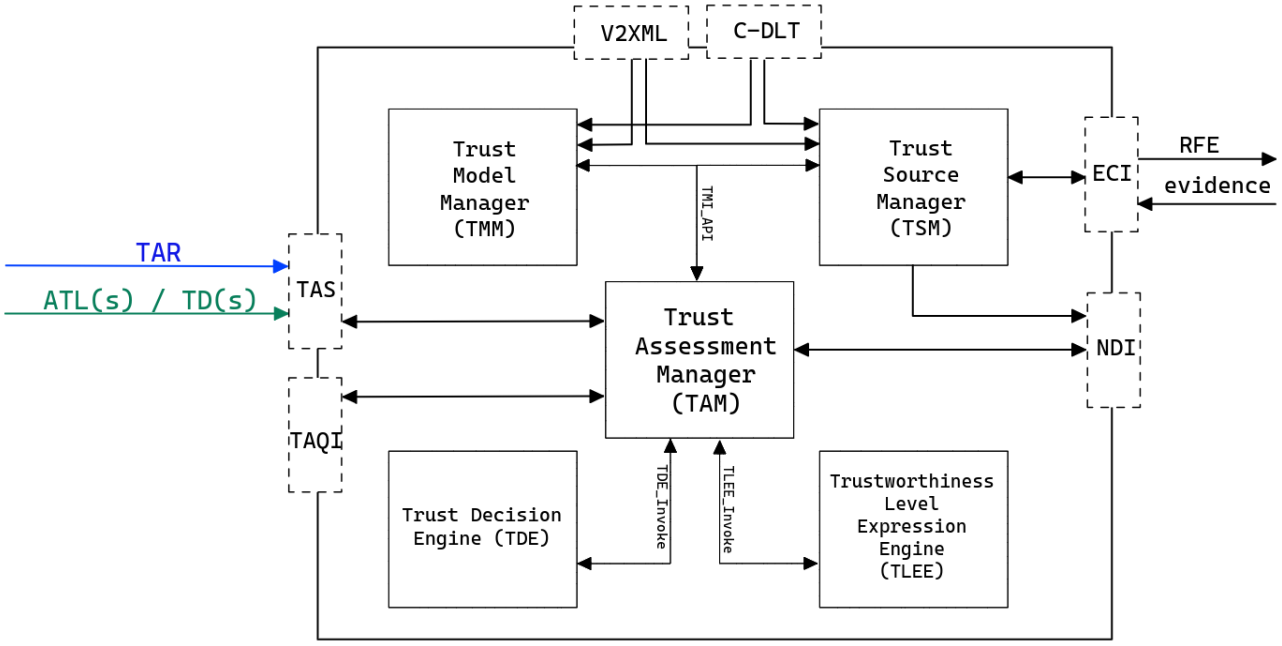


Figure 3.1: High-level architecture of the Trust Assessment Framework (TAF). This includes the five main components: Trust Model Manager, Trust Source Manager, Trust Assessment Manager, Trust Decision Engine, and Trustworthiness Level Expression Engine. The illustration also shows the three main internal interfaces of the TAF, namely the Trust Model Instance API (TMI_API) and the interfaces for the TLEE Invocation (TLEE_Invoke) and the TDE Invocation (TDE_Invoke). In addition, the six external interfaces and services are depicted – the Trust Assessment Service (TAS), the Trust Assessment Query Interface (TAQI), the V2X Message Listener (V2XML), the Connect DLT (C-DLT), the Evidence Collection Interface (ECI), and the Node Discovery Interface (NDI).

The TAF is internally organized as a modular system consisting of different components, as illustrated in the high-level architecture in Figure 3.1. The *Trust Model Manager* (TMM) provides an internal repository of available trust models and instantiates appropriate trust model instances upon application requests. Furthermore, it manages the full life cycle of all trust model instances by incorporating potential changes. The *Trust Source Manager* (TSM) maintains a list of possible trust sources that could be used as evidence for assessing the trustworthiness of an entity. By continuously assessing evidence from external trust sources, the TSM dynamically incorporates trust opinions into existing trust model instances. The *Trustworthiness Level Expression Engine* (TLEE) takes a trust model instance as an input and assesses the level of trustworthiness of the included propositions. The *Trust Decision Engine* (TDE) eventually forms trustworthiness and trust decision based upon output from the TLEE. Finally, the *Trust Assessment Manager* (TAM) is a central component that orchestrates updates from the TMM & TSM and schedules TLEE computations and TDE invocations. The TAM also implements external interfaces so that the trust assessment service of the TAF can be accessed by applications. More details about the individual components of the TAF architecture are described in Section 3.5, following a specification of the external services (see Section 3.4).

3.3.2 Design Goals & Key Design Principles

The basic architecture of the TAF follows a *concurrent, event-driven architecture* in which a number of components await, process, handle, and potentially also emit events. Based on the different external services and interfaces as well as internal tasks the TAF has to integrate, we can separate these components conceptually into the following groups:

(i) Components that process updates from outside the TAF and publish them as TAF-internal events (e.g., V2X Message Listener).
(ii) Components that process internal updates by triggering changes of internal representations (e.g., Trust Model Manager).
(iii) Components that observe changes to internal representations and schedule downstream computations (e.g., Trust Assessment Manager).
(iv) Components that execute dedicated computations upon invocation (e.g., TLEE).
(v) Components that provide services to external entities in order to access internal representations and computation results (e.g., Trust Assessment Manager).

Note that this separation corresponds to an unidirectional flow of execution for the most part: External updates (e.g., CPM messages, evidence) are captured and eventually reflected in internal representations (e.g., trust model instances). Internal representations are then used as an input to run computations (TLEE) and store their results (e.g., cache for recent ATLs). Orthogonal to this, the results of recent computations are used to provide services to applications (e.g., respond to a Trustworthiness Assessment Request (TAR), send notifications).

At the same time, the TAF represents a monolithic application that runs as a single application process on one machine. This choice of architecture and separation of concerns between components allows for the following design goals:

High concurrency: Being designed as a single application process, the TAF requires extensive multi-threading to take advantage of actual parallelism of modern CPU architectures and to provide scalability and performance, even under load. In our event-based architecture, this can easily be done by using dedicated event processors for the different components. Depending on the statefulness and data consistency requirements of components, some components can even employ a pool of threads to handle events concurrently inside the component. No internal events are published to the outside of the TAF. Instead, only well-defined services/interfaces are used for external interaction with the TAF.
That said, the TAF does not use an external message queue/broker *between its components* for a number of reasons: (i) additional message latencies due to network messaging (ii) unrealistic messaging abstractions (e.g., having an unbounded message queue between components by potentially offloading messages to disk), and (iii) increased overhead when using additional threads for sending and consuming messages. Instead, the TAF only uses language-internal mechanisms (i.e., shared memory) for disseminating events internally.

Low coupling: The TAF architecture provides low coupling between most components. Stronger coupling (i.e., direct function calls) is only used between the Trust Assessment Manager and the TLEE for scheduling computationally expensive operations and for subsequent calls to the TDE. Low coupling for the rest of the TAF is desirable for a number of reasons:

(i) As processing and emitting events is the primary mode of interaction between components, components do not need to know much about the internals of other components. Instead, components only need to know potential event types and can implement reactive behavior to these events. New event types can be defined without

interfering with existing components and implementations can be switched transparently as long as required event handlers are implemented.

(ii) The separation of components into independent event processors backed by threads or thread pools allows for bulk-heading behavior in case of load peaks, which is more favorable than back-pressuring behavior given the responsibilities of the TAF. When using back-pressure, an overloaded component slow downs upstream components and eventually slows down the system's ability to process any additional messages from the outside. Such a behavior is important if a system requires guaranteed processing of each single message and needs to throttle down an external source. However, in our scenario, we have no control over the amount and volume of external messages. Even further, we would deliberately drop older messages over newer messages for processing if necessary, as newer messages usually contain a more recent update from the outside world. This type of load-shedding helps to keep the service running with reasonable latency. With bulk-heading, we can isolate components in a way that an overloaded component has minimal impact on other components. For instance, this would allow the TAM to invoke the TLEE and the TLEE to run its computations even if the Trust Source Manager is under high contention due to an unexpected surge of messages.

Single Writer Principle: Conceptually, the TAF could be considered to be a highly concurrent in-memory database that maintains trust model instances and associated data items and eventually runs calculations on subsets of its data. Over their life cycle, most trust model instances will experience phases with very high update rates. Given the potentially large number of concurrent trust model instances, as well as the high number of concurrent modifications to these instances, it is important to be aware of potential bottlenecks. Our approach to allow for high levels of concurrency is (i) to reduce the amount of shared state and usage of concurrent data structures whenever shared state is necessary, (ii) to apply the single writer principle whenever possible. In this principle, only a single execution unit is allowed to modify a data structure while others have read-only access. This principle prevents write contention between multiple writers and is also aligned with implicit optimizations of modern CPU architectures (i.e., caching). Similarly, we consider ownership as a concept to be used between the TAM and the TLEE. The TAM derives a copy of the structure and the values from the latest state of a trust model instance to be used for an TLEE computation. These copies are then passed to the TLEE as parameters and will not be accessed or modified by the rest of the TAF anymore. Similarly, the resulting data structure returned from the TLEE is considered to be immutable.

Pluggable Modules: TAF components that interact with the outside world (e.g., V2X Message Listener) can be swapped for mockup modules. This facilitates development and testing, but it also allows us to use specific adapters when running emulations or workload generators.

3.4 External Interfaces & Services

In this section, we specify the system design underlying the TAF prototype within the CONNECT project. To this end, we specify the communication and interaction of the TAF with other entities outside the TAF (e.g., applications and their respective functions, components on the same node, other vehicles, MECs). This includes (i) services that the TAF provides to other entities, (ii) services from other entities that the TAF utilizes for its own internal implementation, (iii) listening

interfaces to passively receive notifications from other external components.

3.4.1 Interface & Service Overview

We assume three different message exchange patterns to be used by the TAF and its communication partners and its environment:

Request/Response messaging is used for accessing services with a traditional communication pattern. In this case, the client needs to know the server identity, the service it provides, and how to address the server. The request/response pattern can also be used with session semantics (e.g., Trust Assessment Service). Here, the client first establishes a sessions with a special request/response before being able to send further, session-specific requests.

Publish/Subscribe is a message pattern for the exchange of notifications or events, usually between a known set of publishers and subscribers. In case subscribing entities are directly contacting publishers in order to signal their subscription, there is a direct coupling between both parties. This implies knowledge of the entities about the identities of the other entities. Explicit subscriptions are required when the consumer needs to tell the publisher the exact content it wants to subscribe to (i.e., publisher-based message filtering/dissemination) or when a subscription needs to managed explicitly in a service (e.g., specific timeouts for subscriptions).

However, when using a messaging middleware like Kafka, as we do in our prototype, publish/subscribe can also be implemented in a more decoupled way by directly using topics. So instead of knowing the list of subscribers (i.e., for publishers when sending notifications) or publishers (i.e., for subscribers when signaling subscriptions), it is now sufficient to only know the name of a topic to publish to or to subscribe to. This enables communication flows between entities – irrespective of mutual knowledge of their identities. A caveat here is that Kafka topics allow new consumers to read older messages that have been published even before the subscription has occurred. This behavior needs to be addressed to prevent unwanted side-effects or unrealistic assumptions about the communication channel becoming an unbounded archive of all previous messages.

One-Way Notifications are messages sent from one entity to a potentially undefined group of receivers. Network scenarios for this type of messaging include broadcast, multicasts, or geocasts. When using Kafka, such broadcast-like notifications can either be implemented by using a well-known topics that all potential subscribers are expected to be consumers of (e.g., predefined broadcast topic). Alternatively, the sending entity must know all actual receivers and address them appropriately by using corresponding topics (e.g., cell-based/location-based topics for dynamic scenarios).

Table 3.1 illustrates which message exchange patterns are used as part of the interaction between the TAF and other entities.

Please note that the specification of how to handle request/response communication might also apply to parts of the publish/subscribe message exchanges when subscription requests are handled explicitly. Here, the subscribing entity would invoke a request to the publishing entity to ask for the name of the topic that is used for notifications. If subscriptions are handled implicitly (i.e., by directly subscribing to a named, well-known Kafka topic), this does not apply.

Table 3.1: List of services and external interfaces and their usage in different TAF deployment settings. Furthermore, the table lists the different role of the TAF in specific service settings.

Service/External Interface	TAF (Standalone)	TAF (Federated)
Trust Assessment Service	✓ S, Pub	✓ S, Pub
V2X Message Listener	✓ R	✓ R
Trust Assessment Query Interface	✓ C/S	✓ C/S
Evidence Collection Interface	✓ C, R, Sub	✓ C, R, Sub
NTM Service (external application)	–	✓ S, R

C: client role; **S**: server role; **R**: receiver/listener role; **Pub**: publisher role; **Sub**: subscriber role
NTM Service: helper service for federation; uses Trust Assessment Service and feeds Evidence Collection Interface (see Chapter 5)

Table 3.2: List of external interfaces with the involved entities, message exchange patterns and a high level description.

Service Name	Involved Entities	Message Exchange Pattern	Description
Trust Assessment Service	TAF, Application/-Function	Request/Response, Publish/Subscribe	Applications initialize sessions with the TAF by stating relevant trust model template ids; In a session an application can invoke different kinds of TARs and/or subscribe to ATL updates.
V2X Message Listener	V2X Module, TAF	One way notification	Received CAMs / CPMs are forwarded to the TAF.
Evidence Collection Interface	TAF, Evidence Source (e.g. Misbehavior Detection System, AIV)	Request/Response, Publish/Subscribe, One way notifications	The TAF either actively queries a trust source for evidence or receives updates about evidence from trust sources.
Trust Assessment Query Interface	Vehicle TAF, MEC TAF	Request/Response	The Vehicle TAF receives a trust value list containing trust opinions sent from the MEC.

3.4.2 Trust Assessment Service

The Trust Assessment Service (TAS) represents the primary service a TAF provides to its clients — the continuous assessment of trustworthiness of relevant entities. In order to receive such a trust assessment, clients have to establish a session with the TAF and indicate the trust model template they are interested in. Once a session is established, the client can then either request trust assessments or subscribe for notifications in case of changes in the trustworthiness of the target entity.

This service represents a stateful, session-based protocol in which a client indicates interest in a target trust model (by specifying the trust model template to be used) and the TAF as the server

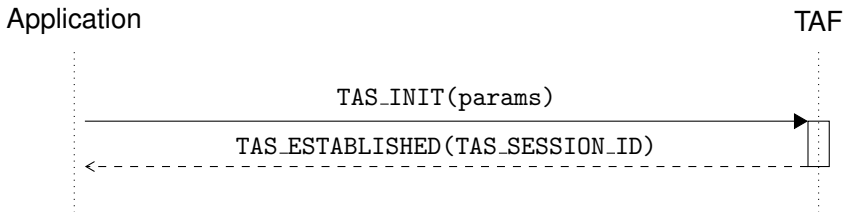
ensures the processing of matching trust model instances for the lifetime of that session.

A client can establish zero or more sessions with a TAF. Each session requires exactly one trust model template to be used within this session. This means that a single client is allowed to run multiple sessions in parallel with the same TAF. Having multiple concurrent sessions is particularly necessary when a client is interested in trustworthiness assessments for different trust model templates at the same time.

For receiving ATLs during an established session, the client can choose between pull-based TARs (i.e., polling using request/response) or event-based subscriptions (i.e., publish/subscribe).

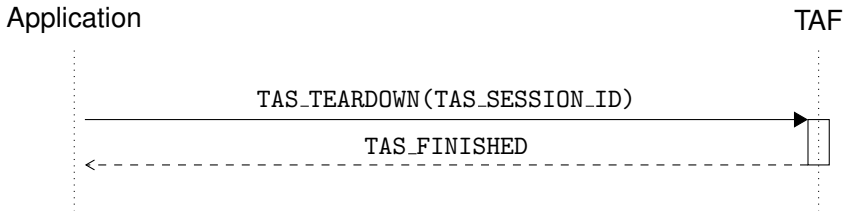
Session Initiation

In the session instantiation, the application tells the TAF which trust model template should be used by sending a TAS Initialization Message (TAS_INIT) which includes the Trust Model Template ID among other parameters. The TAF then internally prepares session handling for this session (e.g., by preparing internal data structures). Once ready, the TAF will respond with the ID of the new session.



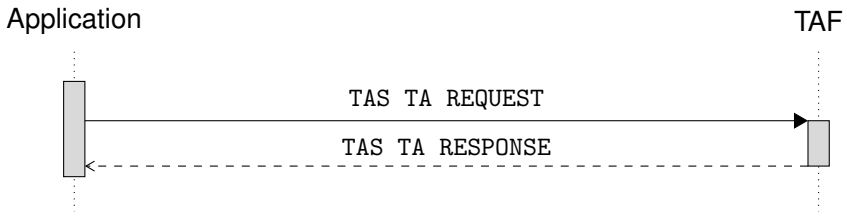
Session Tear-Down

Before shutting down, the application must signal the TAF to tear down the session. This allows the TAF to free all resources allocated for this session. For simplicity, the current design does not include a lease-based session model in which sessions would be purged automatically in case of client inactivity or missing renewals of the session.



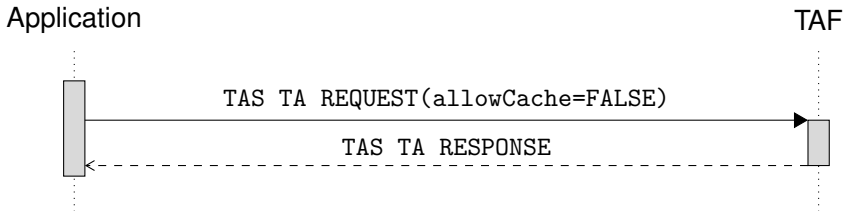
Regular Trust Assessment Requests

Once a session has been established, the application can send TARs to TAF whenever a recent trustworthiness assessment is needed. The TAF will respond to that request in an best-effort way and will potentially use cached results that have been calculated recently.



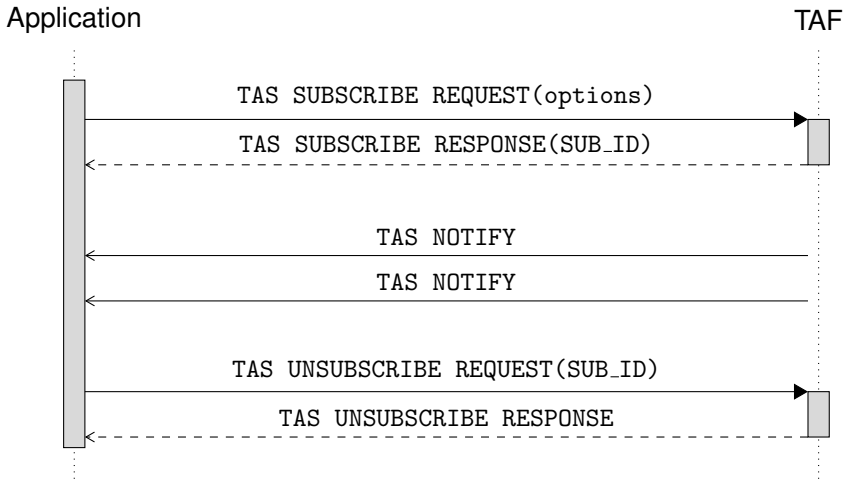
Non-Cached Trust Assessment Requests

In case an application requires freshly calculated assessments and does not accept results potentially precalculated recently by the TAF, it can request non-cached results from the TAF. This is a deliberate trade-off by the application to favor latest results over shorter response times.



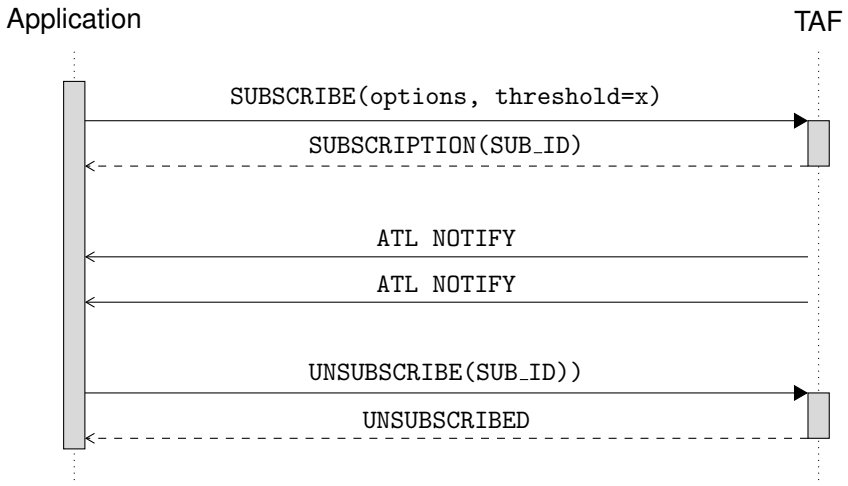
Subscription for Trust Level Changes

In addition to the previously described polling-based approaches, application can also subscribe for changes of ATLs. In the first approach, the application subscribes to any change. This means, whenever an ATL has been recalculated and is not identical to the previously published value, the application will get notified.



Subscription for Threshold-based Changes

An alternative approach for publish/subscribe include a threshold provided by the application during the subscription. In this case, the TAF will notify the application whenever an ATL has been (re-)calculated and change in the resulting value exceeds the threshold specified by the application.

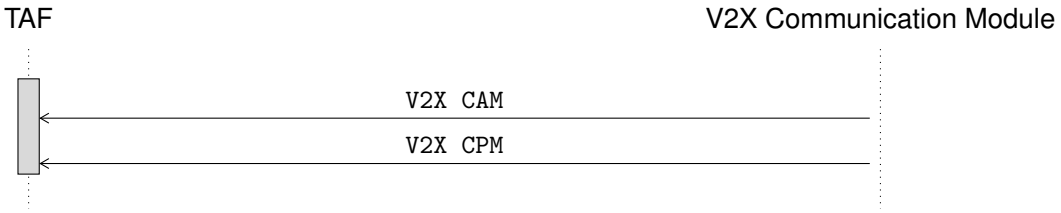


3.4.3 V2X Message Listener

The V2X Message Service is implemented by a TAF-external component that receives CAMs and CPMs (i.e., V2X Communication Module). This component provides a notification interface so that the TAF can register a listener and also receives notifications about incoming V2X messages.

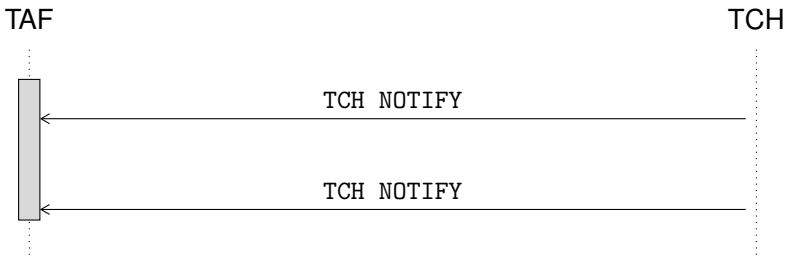
V2X Message Notifications

By registering a listener, the TAF receives copies of incoming CAMs and CPMs. These messages can then be processed by TAF-internal components.



TCH Notifications

TCH messages are created by the trustworthiness claims handler (TCH) and provide evidence about the trustworthiness of a vehicle or a MEC. TCH messages are processed by the TAF as evidence. However, TCH message have a secondary function that can be assigned to the listener tasks: TCH message can be used by the TAF in order to learn about the presence of other entities via these reports. Once a TCH message arrives at the TAF, the information about the trustee (i.e. the entity about which the trust opinion is created) can be extracted from the message and used.



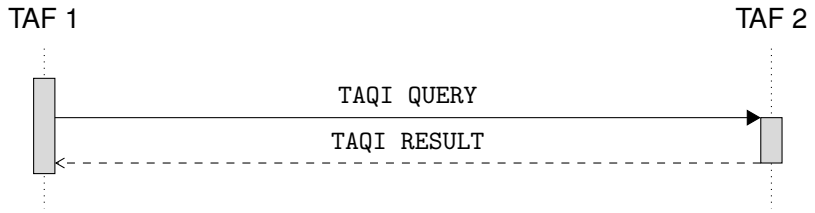
3.4.4 Trust Assessment Query Interface

The Trust Assessment Query Interface represents an auxiliary service of a TAF and enables other TAF instances to query trust assessments of that instance. Hence, this interface enables TAF-to-TAF interaction and can be seen as a first building block for federated behavior in which multiple TAFs enhance their own perspective by incorporating assessments provided by other TAFs.

Note that this interface is substantially different from the primary service interface of TAF, the trust assessment service. The Trust Assessment Query Interface is used by applications or application functions that are collocated with the TAF. Furthermore, the trust assessment service determines which trust models are instantiated by TAF based on existing sessions. In turn, the Trust Assessment Query Interface represents a stateless, pull-based, read-only query protocol in which a TAF can ask for trust values of a remote TAF.

Trust Assessment Query

A trust assessment query (TAQ) contains a selector to identify trust values the querying TAF is interested in. A selector specifies a trust model template type and potential propositions in corresponding trust model instances. Using wildcards allows a selector to make broader queries, e.g., queries that can ask for any trust model instance in which a node with a certain ID is part of. The queried TAF then returns a potentially empty list of trust values satisfying the selector.



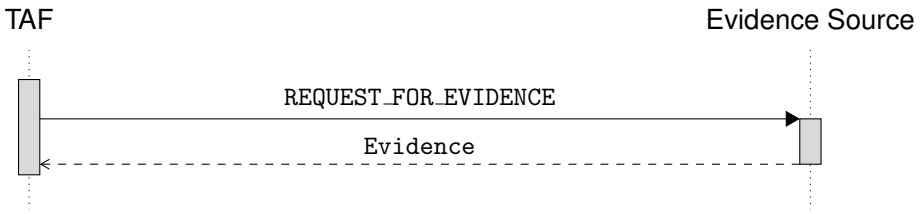
3.4.5 Evidence Collection Interface

The Evidence Collection Interface provides different mechanisms to collect information about other entities to be used by the trust source manager. This includes: (i) actively polling known nodes for evidence, (ii) subscribing for evidence notifications from know nodes, or (iii) receiving evidence updates via one-way messages.

The TAF implements the following interaction scheme so that the TSM can choose between them at runtime.

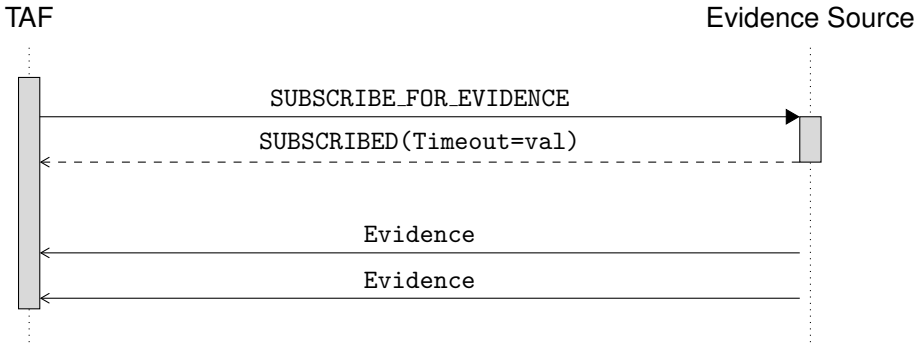
Pull-based Evidence Collection

In this case, the TAF knows about an available evidence source (e.g., by using the Node Discovery Interface) and specifically queries the target for evidence.



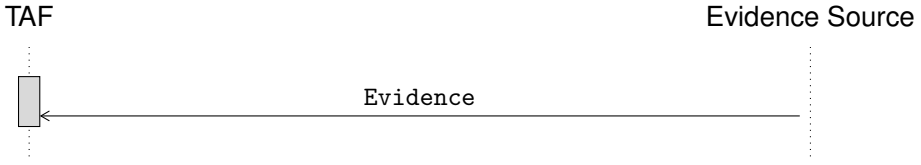
Subscription-based Evidence Collection

As an alternative to the previous interaction, a TAF can also use a known source of evidence and subscribe for evidence updates. These subscriptions have a timeout and need to be renewed.



One-Way Evidence Notifications

A third option is the reception of an unsolicited message that contains evidence. This evidence can the be used by the TAF, if applicable.



3.5 Internal Interfaces & Components

3.5.1 Overview of Interfaces & Components

Internally, the TAF consists of a number of components, as well as user-defined trust models. This means that the TAF can take on several roles: (i) For the designers of trust models, the TAF is a

Table 3.3: List of internal and external interface exposure in the TAF components.

Interface/Service	TAM	TMM	TSM	TLEE	TDE
External					
Trust Assessment Service	✓				
V2X Message Listener		✓	✓		
CONNECT DLT		✓	✓		
Trust Assessment Query Interface	✓				
Evidence Collection Interface			✓		
Node Discovery Interface	✓		✓		
Internal					
TLEE Invocation	✓			✓	
TDE Invocation	✓				✓
APIs for User-Defined Code					
Trust Model Template API	✓	✓			
Trust Model Instance API	✓	✓			
Trust Source Quantifier API			✓		

runtime platform that provides the execution environment so that trust models can be initialized and updated, and (ii) finally at runtime, for vehicular applications that utilize the functions of the TAF, the TAF represents a *service* to outsource trustworthiness decisions based on available trust models.

In order to fulfill these roles, there is a boundary between platform internals that are entirely hidden from the applications and trust models, programming APIs that trust model designers can use to develop own trust models, and external interface that can be accessed by the applications at runtime. For the APIs, the TAF employs the so-called *Hollywood Principle*, a variant of dependency inversion. This means that designers implement their trust model as a set of functions against an internal API. However, the trust models are very limited in what they can do, as the primarily define how a trust model needs to be modified upon external updates. Instead, the TAF calls specific functions of a user-defined trust model based on internal events, hence the TAF always maintains the flow of control. We will explore the corresponding APIs for such user-defined models for trust model templates, trust model instances, and trust source quantifiers later in this chapter.

In addition, there are internal interfaces between internal components to specify their interaction semantics (e.g., how the TAM can invoke the TLEE).

Table 3.3 illustrates the different components of the TAF and how they relate to different internal and external services or APIs.

3.5.2 Trust Assessment Manager

The Trust Assessment Manager (TAM) is a central component for the TAF that is responsible for a number of tasks: (i) providing the trust assessment service functionalities to applications (including application session management); (ii) processing and incorporating changes of trust model instances when indicated by the trust model manager or the trust source manager; (iii) scheduling of TLEE computations; and (iv) scheduling of TDE function calls.

The TAM hereby decouples and compartmentalizes independent functions of the TAF:

1. It intentionally separates the observation of external changes from the update of the corresponding internal representations (i.e., trust model instances).
2. It intentionally separates the update of internal representations from the (re-)calculations of ATLs (i.e., TLEE) and trust decisions (i.e., TDE).
3. It intentionally separates the internal state handling from external requests (i.e., TAS).

In doing so, the TAM can deliberately schedule TLEE and TDE executions depending on (i) the number of pending change events for the trust model instance, (ii) the amount of trust model instance updates not yet reflected in the previous calculation, (iii) the staleness of previous calculation results stored in a cache, (iv) the demand for updated decisions based on actual sessions or pending requests (i.e., non-cached TARs), and (v) the current number of other concurrently issued executions for other trust model instances.

Internally, the TAM has a configurable number of workers that share the existing trust model instances and allow for parallel processing of updates and TLEE calculations.

This gives us a high level of flexibility to configure and scale the TAF to different load scenarios and runtime environments with varying factors of parallelism and caching.

3.5.3 Trust Model Manager

The Trust Model Manager is a component of the TAF responsible for managing Trust Model Templates and Trust Model Instances. Trust Model Templates are application-specific templates created at design-time which specify all the relevant information for the instantiation of Trust Model Instances. A Trust Model Template is always matched to a specific function/application and is meant to be used by the TAF to inform about which data the application needs as input to run a certain function, i.e., which data the TAF needs to output the Actual Trustworthiness Level for. Moreover, the Trust Model Template also specifies all the relevant Trust Objects, Trust Model Instantiation Policies, Trust Relationships, Trust Sources, and Trust Methods which are needed. Trust Model Instances are instantiated at run-time based on the Trust Model Templates and they are used to store Atomic Trust Opinions, Trust Opinions, and Actual Trustworthiness Levels assessed during run-time.

Trust Model Templates are stored in a Trust Model Template Database (TMT-DB) which is managed by the Trust Model Manager. Each Trust Model Template in the TMT-DB has a Trust Model Template ID (TMT-ID) associated with it, unique to each Trust Model Template. The TMT-ID values are pre-agreed upon and known to both the TAF and the application requesting a Trustworthiness Assessment, and, in most cases, do not change over time.

In the process of initializing a session with the TAF, an application will send the TMT-ID as part of the **initialization message**. The Trust Assessment Manager then forwards the TMT-ID to the Trust Model Manager which then either:

1. locates the corresponding Trust Model Template in the local TMT repository or, if not,
2. queries the Distributed Ledger Technology (DLT) where additional Trust Model Templates are stored for that specific TMT-ID.

If the specific TMT-ID is not found either in the local TMT-DB or in the DLT, then the Trust Model Manager returns a null value to the Trust Assessment Manager.

If, however, a corresponding Trust Model Template is found, then the Trust Model Manager will either:

1. directly create a Trust Model Instance based on this template and return a reference to the instance to the Trust Assessment Manager, or
2. return a reference to a dynamically spawn instances at later points in time upon certain events.

During the run-time phase, the TAF will receive a Trustworthiness Assessment Request (TAR) from an application, which will, among other pieces of information, contain a TMT-ID. This indicates that the application wants the TAF to perform trust assessment based on that specific Trust Model Template. The Trust Assessment Manager will use the TMT-ID to check whether there is already a Trust Model Instance associated with it. If there is, and if the Actual Trustworthiness levels have already been assessed and stored in the instance, the TAM will return the corresponding Actual Trustworthiness Levels to the application. If there is no Trust Model Instance for the corresponding TMT-ID yet, then the Trust Assessment Manager forwards the TMT-ID to the Trust Model Manager to start instantiating the instance(s).

In general, the Trust Model Manager is responsible for:

1. Creating appropriate Trust Model Instances during run-time based on the input it receives from the Trust Assessment Manager, as well as the incoming Cooperative Awareness Messages (CAMs) and Collective Perception Messages (CPMs).
2. Adding new trust objects to already existing Trust Model Instances based on appropriate input (CAMs and CPMs).
3. Deleting expired trust objects (such as CPMs representing vehicles which the TAF has not heard about in a while).
4. Deleting Trust Model Instances when a vehicle for which the instance was instantiated is no longer sending CAM or CPMs after a certain period of time.
5. Managing and updating the Trust Model Template Database.

3.5.4 Trust Source Manager

The **Trust Source Manager (TSM)** is a component of the TAF responsible for managing **Trust Source Quantifier Instances (TSQ-I)**.A TSQ-I is created at run-time based on the corresponding Trust Source Quantifier Template (TSQ-T) which is created at design time. Each TSQ-I is assigned to exactly one Trust Source. Trust Sources are external entities outside of the TAF. They provide evidence about the trustworthiness of a node or a data item. Examples for Trust Sources are a misbehavior detection system or the implemented security controls in a node. The TSQ-I receives evidence from the corresponding Trust Source and calculates an atomic trust opinion based on this evidence. To be able to do that, the corresponding TSQ-T describes how the evidence provided by the Trust Source should be interpreted. In addition, in the TSQ-T the corresponding formulas and calculation rules for calculating the belief, disbelief and uncertainty of the atomic trust opinion are defined. See Chapter 9 for the calculation rules and formulas of the trust sources misbehavior detection and implemented security controls.

There are two options for where the TAF obtains the TSQ-T. Both options are described in the following:

1. The TSQ-Ts could be stored in the TAF. They are created during design time and are installed together with the TAF on the vehicle/MEC when the corresponding node is set up by the manufacturer. Updates of the TSQ-Ts would be possible by an Over-The-Air update of the software of the TAF.
2. The TSQ-Ts could be stored in the cloud/DLT. If a TSQ-T is required for the calculation of an atomic trust opinion and this template is not stored locally in the TAF, the TAF requests the template from the cloud/DLT and will store it in its local memory.

In the trust model instance described in the previous section, it is specified which TSQ-T should be used to calculate the corresponding trust opinion. The TSQ-Ts have unique IDs (TSQ-T IDs). As already described in the previous section, an application sends the TMT-ID as part of the initialization message when a session is initialized between the application and the TAF. At this initialization phase, the TSM will process the trust model instance and extract the corresponding TSQ-T IDs for each trust relationship. In addition to the TSQ-T IDs, further metadata is also provided in the trust model for each trust source. Based on this metadata a TSQ-I is created out of the TSQ-T. This metadata contains information for the calculation of the trust opinions. An example for this are weights describing the importance of each misbehavior detector when using a misbehavior detection system as a trust source. These weights are used as inputs for the calculation rules and formulas described in the TSQ-T. In addition to the metadata necessary for the calculation rules and formulas, the trust model instance also includes information about where the TSM should request the evidence from. For example, for misbehavior detection as a trust source, it is specified that the evidence will be requested from the misbehavior detection system running on the vehicle/MEC. For security controls as a trust source, it is specified that the evidence will be requested from the Attestation and Integrity Verification (AIV) component. Based on the TSQ-T ID and the corresponding metadata provided in the trust model instance, the TSM creates an instance based on TSQ-T, which is the TSQ-I.

3.5.5 Trustworthiness Level Expression Engine

The Trust Source Manager, calculates the Trust Opinions for each trust relationship. However, Trust Models frequently consist of multiple interconnected trust relationships, creating a trust network. To determine the Trust Opinion for the entire trust network—also referred to as the Actual Trust Level (ATL) throughout this deliverable—a collection of specialized functions is required. These functionalities are supplied by various modules within the TLEE. Further details about the Trustworthiness Level Expression Engine are outlined in Chapter 4.

From the project's initiation, the TLEE was designed to meet critical requirements, including operation under strict timing constraints and the ability to evaluate trust in complex networks involving both direct and indirect relationships. It integrates key concepts such as transitive trust and trust fusion to ensure comprehensive trust assessment. Moreover, the engine must dynamically adapt its calculations at run-time to accommodate structural changes in the Trust Model Instance (i.e., the evolving Trust Network), where new relationships may form and existing ones may be discarded.

3.6 Realization of Trust Models

3.6.1 Trust Model Overview

In the CONNECT Deliverable 3.1 [2], we had defined a trust model as:

> *A trust model is a graph-based model which represents all components and data needed to perform a certain function. It consists of trust objects and directional trust relationships between trust objects (i.e., the trust network). It also stores trust sources used to quantify trust relationships.*

This is a relatively generic and broad definition which we now extend by focusing on the trust model's functionality.

At its core, a Trust Model is a directed acyclic graph data structure whose nodes we refer to as **Trust Objects**. A visual concept of a Trust Model is shown in Figure 3.2.

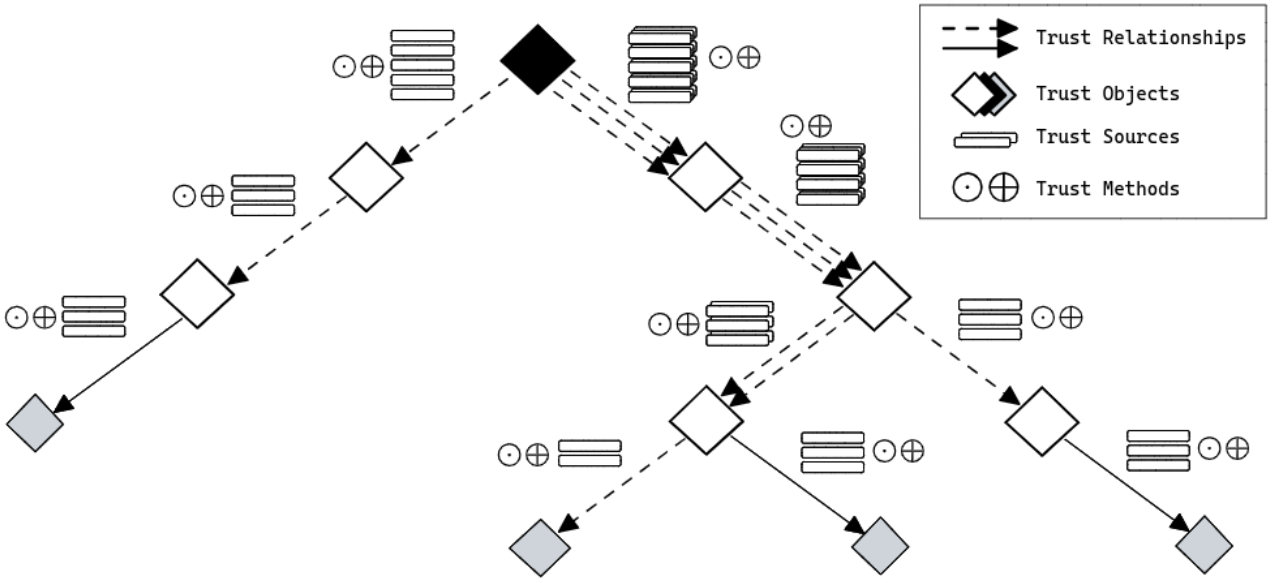


Figure 3.2: A visual concept of a Trust Model.

A Trust Model always has a single node that has no incoming edges which we refer to as the *root node* of the Trust Model. The root node represents the entity *for which* the trustworthiness is being assessed which we refer to as the **Agent** (the black Trust Object in Figure 3.2). The Trust Model also has at least one leaf node representing the entity whose trustworthiness is being assessed which we refer to as the **Proposition** (the grey Trust Objects in Figure 3.2). There are at times intermediary nodes between the root node and the leaf nodes as shown in Figure 3.2 (the white Trust Objects).

The arrows in the visual concept shown in the Figure 3.2 represent that there is a **Trust Relationship** between trust objects, where the source node of the arrow is referred to as a *Trustor* and the target node of the arrow is referred to as a*Trustee*. A Trust Relationship was defined in Section 4.1 of Deliverable D3.1 [2] and is expressed in form of a subjective logic binomial opinion, $\omega_{Trustee}^{Trustor}$, as defined in Section 5.4.1 of Deliverable D3.1 [2].

The **Trust Sources** embedded in the Trust Model represent the entities which the Trust Assessment Framework needs to collect evidence from to quantify and assess a single Trust Relationship. There is a set of Trust Sources attached to every single Trust Relationship inside a Trust Model, but their number and type will vary among Trust Relationships.

The **Trust Methods** are also embedded in the Trust Model as they are necessary to either quantify a set of evidence from a single Trust Source to an Atomic Trust Opinion, to fuse Atomic Trust Opinions into a Trust Opinion, or to fuse two or more Trust Opinions into an Actual Trustworthiness Level.

The capabilities of a trust model can be generalized as follows:

(i) It allows the security analyst to *specify* all information at *design-time* which are relevant for trustworthiness assessment and are used by the TAF at run-time.
(ii) It allows to *store* quantifications of trustworthiness at different stages of trustworthiness assessment during TAF's *run-time*.

Specifically, a Trust Model:

1. *Specifies* the **Trust Objects** whose trustworthiness should be assessed with the goal of evaluating the **Actual Trustworthiness Level(s)** for a specific application or function.
2. *Specifies* the **Trust Relationships** representing Trust Opinions between any two Trust Objects which need to be quantified.
3. *Specifies* the **Trust Model Instantiation Policies** for instantiating additional Trust Objects inside a single Trust Model Instance at run-time when needed. A Trust Model will change its structure and incorporate, for example, new trust objects if a new vehicle appears in a cooperative driving scenario.
4. *Specifies* a list of all of the **Trust Sources** whose evidence is needed for assessing the trustworthiness of Trust Objects in form of a **Trust Opinion**.
5. *Specifies* mathematical Trust Methods for fusing Atomic Trust Opinions to create **Trust Opinions**.
6. *Specifies* a mathematical method for fusing two or more **Trust Opinions**, when needed to produce an **Actual Trustworthiness Level**.
7. *Stores* all of the assessed **(Atomic) Trust Opinions** for each individual Trust Relationship.
8. *Stores* all of the assessed **Actual Trustworthiness Levels**.

A trust model at design time contains specifications of the potential structure of the model and how the trust model can be instantiated, populated, and maintained at runtime.

3.6.2 Trust Model Templates and Instances

To be able to distinguish between a design-time trust model and a run-time trust model, we defined the following two manifestations of trust models – templates and instances:

Trust Model Template (TMT) is a design-time manifestation of a trust model. A Trust Model Template is created at design-time with the goal of capturing all of information necessary for the Trust Assessment Framework to assess trustworthiness for a specific application upon the receipt of a Trustworthiness Assessment Request. A Trust Model Template specifies all of the relevant Trust Objects, Trust Model Instantiation Policies, Trust Relationships,

Trust Sources, and Trust Methods which are to be instantiated as part of a Trust Model Instance. As such, a uniquely identifiable, application-specific Trust Model Template is used by the Trust Model Manager to instantiate a corresponding Trust Model Instance at run-time, making a TMI completely dependent on the design of the TMT.

Trust Model Instance (TMI) is a run-time manifestation of a trust model. A Trust Model Instance reflects the TAF's current and latest view on the world and is thus a single, internal source of truth for any computations to be done during trust assessment. The TAF uses the information inside a Trust Model Instance to know which Trust Sources to instantiate, which Trust Relationships need to be quantified in form of a Trust Opinion, and which Trust Opinions form an ATL. However, a TMI is also designed to serve as a data structure to store Atomic Trust Opinions, Trust Opinions, and Actual Trustworthiness Levels during the lifetime of an instance. As such, a TMI will be continually updated with recalculated opinions and levels.

Trust model templates can specify varying levels of dynamicity regarding the topology used by the model at runtime. Static trust model instances have a fixed topology that is already fully determined at design time in their template. In turn, there are also trust model instances whose graph structure can change over time based on the policies inside their corresponding template **and** external input to the TAF, such as CAMs and CPMs. We refer to the prior as **Static Trust Model Instances** and the latter as **Dynamic Trust Model Instances**.

Static Trust Models Instances

The structure of Static Trust Model Instances is solely determined by their Trust Model Templates. The graph structure of the static Trust Model Instance stays the same throughout the lifetime of the instance and it does not change based on any input received during run-time.

In case of in-vehicular applications which utilize the in-vehicle network to collect input data, like in our running example, the graph structure of the Trust Model Instance is identical to the predefined structure specified in the corresponding Trust Model Template. Its structure is static in the sense that it does not change over time, i.e., all of the different Trust Relationships and the Trust Objects stay the same. This is because static Trust Models Instances are instantiated for applications whose data flow is fixed and for which the nodes the data flows through are known in advance. However, different types of trustworthiness opinions such as Atomic Trust Opinions, Trust Opinions and ATLs will be stored inside the instance at run-time and they are expected to change over time depending on changes in evidence received.

An example Trust Model Template for the use-case of Adaptive Cruise Control (ACC) is shown in Figure 3.3. As can be seen from the figure, the TMT consists of Trust Objects, Trust Relationships, Trust Methods, and Trust Sources. The TMT has a single root node, VC, representing the Vehicular Computer, which is the Agent for which the TAF would be assessing trustworthiness as this is where the ACC application will be running. Moreover, there are four leaf nodes, R_{data}, G, C_{data}, and L_{data}, representing the radar data, the GNSS sensor, the camera data, and the LIDAR data, which are the propositions, i.e., the entities whose trustworthiness we need to assess in form of Actual Trustworthiness Levels. In order to do that, we need to first quantify every single relationship which is represented by a directed edge, including the relationships between the Vehicle Computer Trust Object, VC, and the Zonal Controller Trust Objects, ZC5 and ZC2. This will be done by analyzing evidence from pre-determined Trust Sources and by using preselected Trust Methods to perform the quantification of evidence, all of which have already been

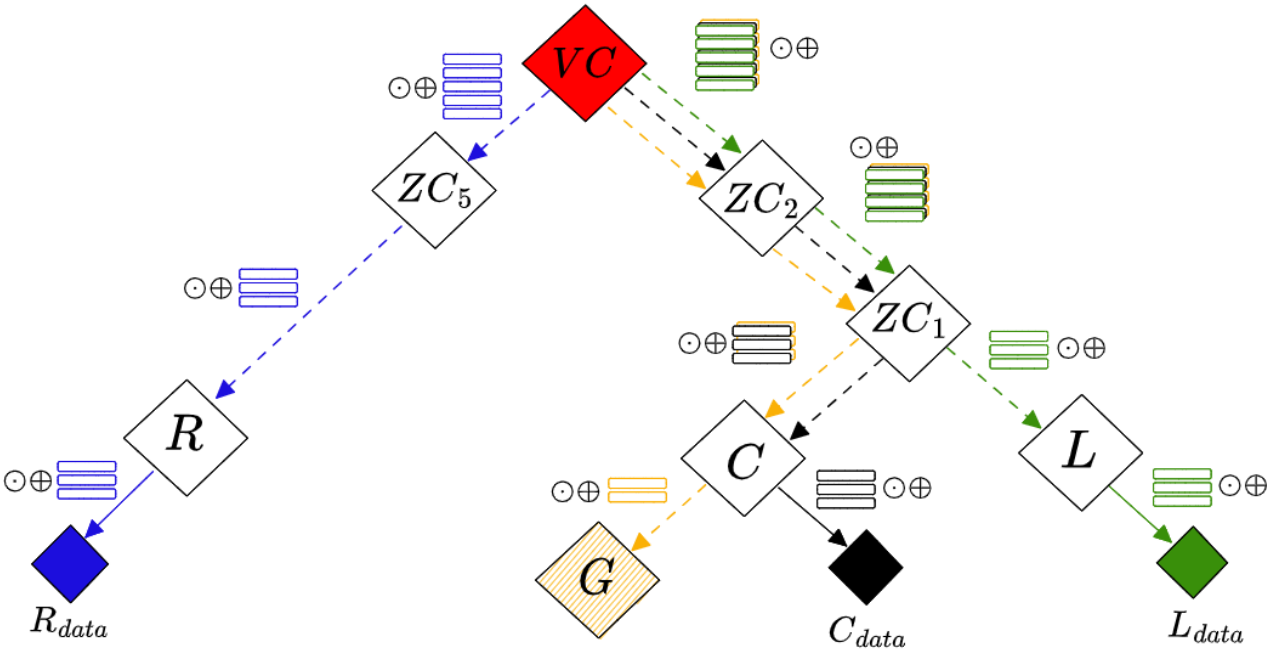


Figure 3.3: A visual concept of a Trust Model Template - static use-case.

specified in the TMT. But before all of this is done at run-time, a Trust Model Template needs to be instantiated to create a Trust Model Instance.

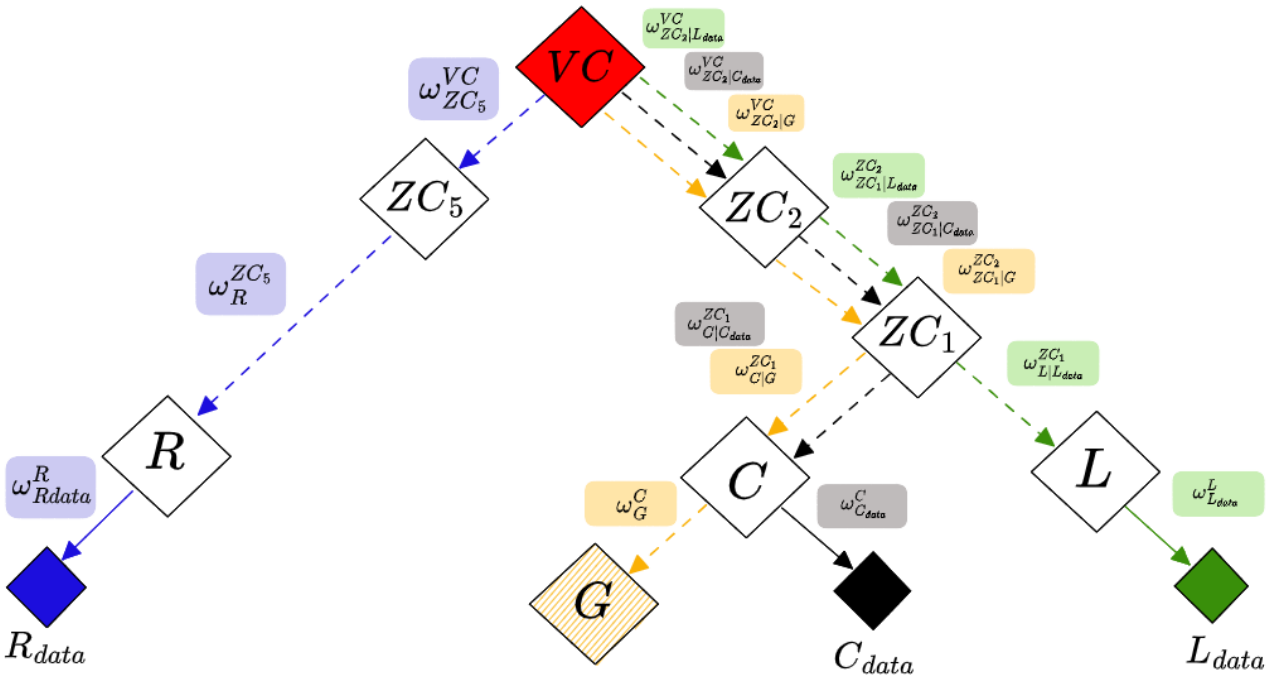


Figure 3.4: A visual concept of a static Trust Model Instance

An example of a Static Trust Model Instance for the use-case of Adaptive Cruise Control is shown in Figure 3.4. As can be seen from the figure, the structure of the TMI is the same as that of its corresponding TMT in Figure 3.3. What is different, however, is that the instance will, at some point, contain numerical values which have been stored in it in the process of trustworthiness assessment. The figure shows only Trust Opinions between relevant Trust Objects, such as $\omega^{VC}_{ZC_5}$,

$\omega^{L}_{L_{data}}$, etc. for the sake of not overloading the figure, but the TMI that the figure represents is intended to also cache the Atomic Trust Opinions that the TSM uses to create the Trust Opinions, as well as the ATLs which the TLEE forms from Trust Opinions through the process of trust discounting.

Note that for applications which are represented by Static TMIs, a single TMI is needed to assess the trustworthiness of all the of relevant input data. That is, a single static TMI per application is enough to calculate all of the necessary propositions. This is not the case for Dynamic Trust Model Instances.

Dynamic Trust Models Instances

Dynamic Trust Model Instances are instantiated based on their Trust Model Templates **and** based on external input received by the TAF, such as Cooperative Awareness Messages (CAMs) and Collective Perception Messages (CPMs). Unlike in the case of Static TMIs, there can be multiple Dynamic TMIs per single application i.e. multiple Dynamic TMIs all based on a single TMT. This is due to the fact that Dynamic TMIs are instantiated for each sender of a CAM or a CPM, and each sender that sends a CAM/CPM to the ego vehicle running the TAF will have its own corresponding TMI inside the TAF.

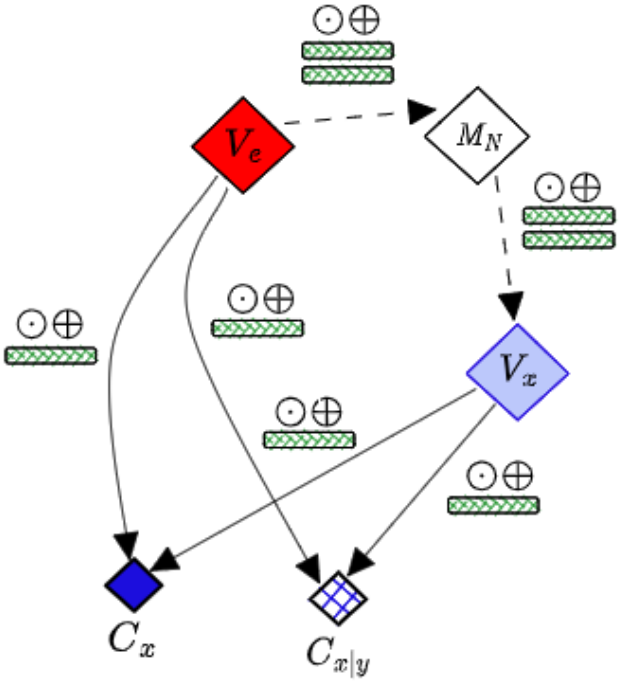


Figure 3.5: A visual concept of a Trust Model Template - dynamic use-case.

An example of a Trust Model Template used for the use-case of a Local Dynamic Map (LDM) which would store trustworthiness of vehicular information extracted from CAMs and CPMs is shown in Figure 3.5. As can be seen from the figure, this TMT also consists of Trust Objects, Trust Relationships, Trust Methods, and Trust Sources. The TMT has a single root node, V_e, representing the ego vehicle, which is the Agent for which the TAF would be assessing trustworthiness. Moreover, there are two leaf nodes in this TMT, C_x, representing a CAM message sent by vehicle V_x, and $C_{x|y}$, representing a CPM message sent by vehicle V_x containing information about vehicle V_y. These are the propositions, i.e., the entities whose trustworthiness the TAF

needs to assess in form of Actual Trustworthiness Levels. In order to do that, the TAF would build a Trust Opinion on C_x and $C_{x|y}$ directly. However, this TMT suggests that a secondary opinion on the received CAM and CPM should be built by collecting an opinion of a Mobile Edge Computer (MEC), M_N, on the vehicle V_x which sent the messages. There is also a relationship between the V_x and its own messages, as well as a relationship between the ego vehicle, V_e, and the M_N that provided an opinion on the vehicle which sent the messages V_x. But before all these relationships are assessed at run-time, this Trust Model Template would need to be instantiated to create a Dynamic Trust Model Instance.

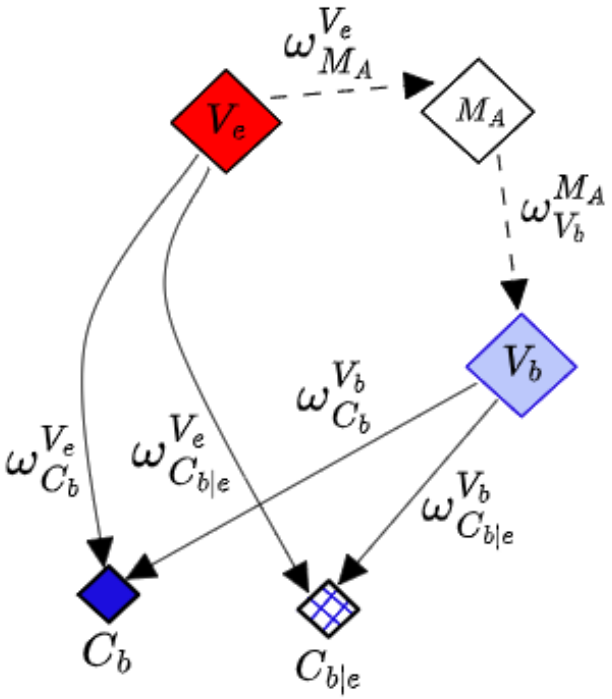


Figure 3.6: A visual concept of a Dynamic Trust Model Instance

An example of a Dynamic Trust Model Instance for the use-case of a Local Dynamic Map is shown in Figure 3.6. As can be seen from the figure, the graph structure of the initial TMI is the same as the fixed structure specified in its corresponding TMT in Figure 3.5. However, Trust Objects have been instantiated for specific entities. For example, a Trust Object has been instantiated for a specific vehicle, V_b, which sent a CAM, C_b, containing information, such as speed and position, about itself, and a CPM, $C_{b|e}$, containing information about the ego vehicle. Trust Objects have been instantiated for these two messages as well in this TMI. Moreover, a Trust Object has been instantiated for a specific Mobile Edge Computer, M_A, which is the MEC that can provide an opinion on the sender vehicle, V_b. As can be noted, a Dynamic Trust Model Instance focuses on a single sender vehicle and the trustworthiness of its messages. Similar to Static TMIs, Dynamic TMIS will cache Atomic Trust Opinions, Trust Opinions, and ATLs at run-time and they are also expected to change over time depending on changes in evidence.

Moreover, the graph structure of the Dynamic Trust Model Instance is expected to change during the lifetime of the instance. This is because Dynamic Trust Model Instances represent vehicular applications which use CAMs and CPMs as input and, as such, require the TAF to calculate trustworthiness of incoming CAMs and CPMs. As CAMs and CPMs are sent about different vehicles at different points in time, the Trust Model Manager needs to dynamically update the structure of a Trust Model Instance to include new Trust Objects and remove the ones which are

no longer relevant.

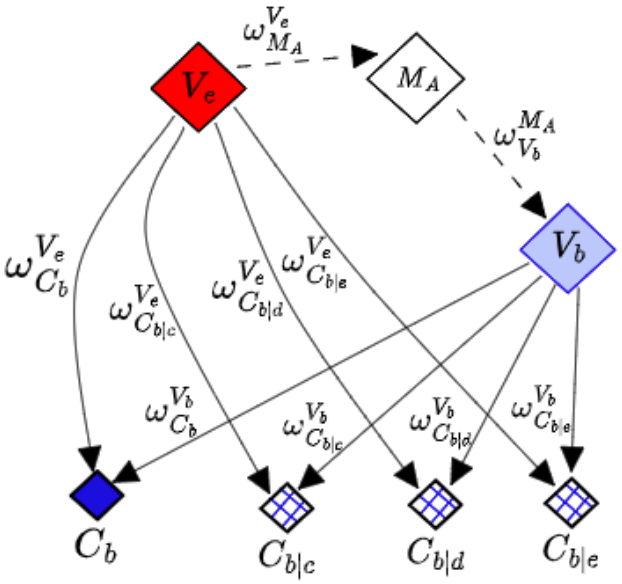


Figure 3.7: A visual concept of a Dynamic Trust Model Instance whose structure changed

An example of what the change of the structure of a Dynamic TMI could look like is shown in Figure 3.7. This example TMI is an extension of the initial TMI shown in Figure 3.6. It would have been extended upon the receipt of a CPM from the same vehicle V_b that contained information about vehicles V_c and V_d. These would be represented as Trust Objects $C_{b|c}$ and $C_{b|d}$ in the updated Dynamic TMI shown in Figure 3.7. Should vehicles V_c and V_d get out of detection range of the sender vehicle V_b, V_b would no longer include information about these vehicles in its CPM messages received by the ego vehicle V_e. This would trigger the deletion of Trust Objects $C_{b|c}$ and $C_{b|d}$ at certain point in time.

The dynamic addition and deletion of Trust Objects to a single Trust Model Instance can result in multiple changes of the structure of the Trust Model Instance over time. Hence, it is important to be able to distinguish between different TMIs at different points in time. Moreover, we anticipate that even the Trust Model Templates may need to be changed over time, albeit significantly less frequently as compared to the Trust Model Instances. For this, we propose to have the Trust Model Versioning scheme as explained in the following subsection.

3.6.3 Trust Model Programming Model

The programming model for trust models consists of two separate APIs, both following an inversion of control approach: The trust model developers provide user-defined functions implemented against the interface of these APIs. At runtime, the TAF then calls these user-provided functions in the appropriate runtime context inside the TAF.

The Trust Model Template API specifies templates of trust models and provide the basis to spawn actual instances of the trust model. This also includes specifications with types of trust sources and evidence are used in this trust model and what internal events should be used as a trigger to spawn new trust model instances.

In turn, the Trust Model Instance API specifies how a specific instance of a trust model evolves over its life cycle. This includes the instantiation of the trust model instance, the handling of

updates to the instance, provision of trust model data to the TLEE, and eventually, the destruction of the trust model instance with potential cleanup operations.

Trust Model Template API

The Trust Model Template API defines static properties of the trust model and defines how instances can be spawned. Currently, there are the following spawn options:

Static Spawning When using static spawning, there is a 1:1 mapping between the trust assessment session (requested by the application at the TAF) and trust model instance. Whenever a session gets created, the session specifies the trust model template, and upon establishing the session, the TAF will create exactly on trust model instance as well. This instance will exist as long as the underlying session is running.

Dynamic Spawning This variant allows the deferred creation of one or more instances within a session, based on certain types of internal events. This events include:

- Learning about the presence of other vehicles extracted from `V2X_CPM` messages.
- Learning about the presence of other vehicles extracted from `TCH_NOTIFY` messages.

Based on the selected trigger, the trust model template will spawn a new trust model instance for each target entity once it becomes present. Usually, observed entities have a time-to-live that is extended for each new contact. However, if an entity is not observed via the two mechanisms listed above, the entity will be marked as gone after a timeout, and the corresponding trust model instance for that entity will be destroyed as well.

The following functions illustrate the API for trust model templates. It lists all the functions that need to be implemented by the trust model developer to add a new trust model template:

`TemplateName()( string`

Returns the name of the trust model template.

returns A string representation of the trust model template.

`Version() string`

Returns the version of the trust model template.

returns A string representation indicating the implementation version of the trust model template, using semantic versioning format (e.g., `1.0.2`).

`Spawn(params map[string]string, context TafContext) ([]TrustSourceQuantifier, TrustModelInstance, DynamicTrustModelInstanceSpawner, error)`

This function prepares the spawning operation of trust model instances for this template. It could either directly return a trust model instance (static spawning), or return a dynamic spawner which creates new instances upon internal events (dynamic spawning).

`params`	An optional key/value map of parameters to be used for this trust model in this session.
`context`	An internal struct containing runtime information of the TAF, including the TAF configuration.
returns	In case of a non-error call, the function returns the set of trust source quantifiers to be used for this type of trust model in this session. Next it *either* returns a trust model instance *or* a dynamic spawner that will create instances dynamically. The spawner must implement callback functions to react to relevant internal events.

`EvidenceTypes() []EvidenceType`

Returns a list of EvidenceType(s) used by instances of this template.

returns	the list of evidence types used by this trust model. As each evidence type is associated with a trust source, it also indirectly lists the trust sources used by this trust model.

`Description() string`

Returns a textual description of the template.

returns	the template's description.

`Type() TrustModelTemplateType`

Returns the TrustModelTemplateType of this template, i.e., whether the spawning operation is static, or dynamic. If it dynamic, it also encodes whether this is based on observed vehicles (CPMs) or trustees (TCH).

returns	the internal template type.

`Identifier() string`

Returns an identifying string that includes the name and the version of the template.

returns	the template's full identifier.

`SigningHash() string`

Returns a textual representation of a hash that states the authenticity and integrity of the trust model.

returns	the signing hash of that template.

Trust Model Instance API

The Trust Model Instance API specifies how an instance of a trust model evolves over time. When implementing a new trust model, the designers and developers of that trust model not only need to implement the template as illustrated before, but they also need to implement the trust model instance against the following interface:

```
ID() string
```

Returns the name of the trust model template.

returns A string representation of the trust model template.

```
Version() int
```

Returns the version number of the state of this instance. The version number is a logical clock that gets incremented for each set of change operations executed on the instance. This includes both changes to the topology of the trust model and changes of the opinions.

returns the latest version number.

```
Fingerprint() uint32
```

Returns a numerical fingerprint of the structure of the instance. The exact number has no semantic meaning and should be treated similar to a hash code. However, if the fingerprint of two versions of an instance are equal, their structure is identical.

returns the latest fingerprint.

```
Structure() trustmodelstructure.TrustGraphStructure
```

Returns the topological structure of the instance in form of an adjacency list between the trust objects. This is used by the TLEE to get the topological structure of the trust model instance.

returns the structure as an adjacency list struct.

```
Values() map[string][]trustmodelstructure.TrustRelationship
```

Returns the trust values of an instance. More precisely, in returns for each scope a list of trust relationships. The trust relationships each specify the opinions between trustor and trustee in that scope. Again, this is used by the TLEE to get the topological structure of the trust model instance.

returns the relationship values for each scope.

```
Template() TrustModelTemplate
```

Returns the template this instance is based upon.

returns the corresponding template.

`RTLs() map[string]subjectivelogic.QueryableOpinion`

Returns the required trust levels for each proposition. This is used by the Trust Decision Engine to do the final decision on ATLs values calculated by the TLEE.

returns a map that contains propositions and their corresponding required trust levels encoded as subjective logic opinions.

`Initialize(params map[string]interface)`

Initialize is called once when an instance has been spawned before it receives the first updates. A map of optional (runtime) parameters can be passed to Initialize to configure the instance in addition to parameters used at spawn time.

`params` An optional key/value map of parameters to be used for this trust model instance in this session.

`Update(update Update) bool`

Update applies an update operation on the instance, potentially modifying the structure and/or values of the underlying trust graph of this instance. Currently, valid update operations include `ADD_TRUST_OBJECT`, `REMOVE_TRUST_OBJECT`, and `UPDATE_ATO`.

`update` Internal update operation that needs to be applied to the instance.
returns The returning boolean indicates whether this change requires a recalculation of ATLs (so whether the TLEE should be called after the update).

`Cleanup()`

Cleanup is called once before an instance gets removed from a TAM worker and destroyed.

returns (no return value)

In summary, a trust model instance has the following behavior, as shown in Figure 3.8:

3.7 Realization of Trust Sources

3.7.1 Trust Source Overview

The **Trust Source Manager (TSM)** is a component of the TAF responsible for managing **Trust Source Quantifier Instances (TSQ-I)** As the TSM manages the TSQ-Is, it is aware of which instance needs evidence from which trust source. Therefore, the TSM handles the communication between the TAF and the trust sources. As already mentioned before, these are external entities that represent trust sources, such as a misbehavior detection system. The TSM either requests evidence or subscribes to the external entity so that it receives evidence as soon as the evidence has changed. The TSM also manages received evidence and forwards it to the corresponding TSQ-I. For this purpose, a unique identifier is necessary in each received evidence and which can be used by the TSM to map the evidence to the corresponding TSQ-I.

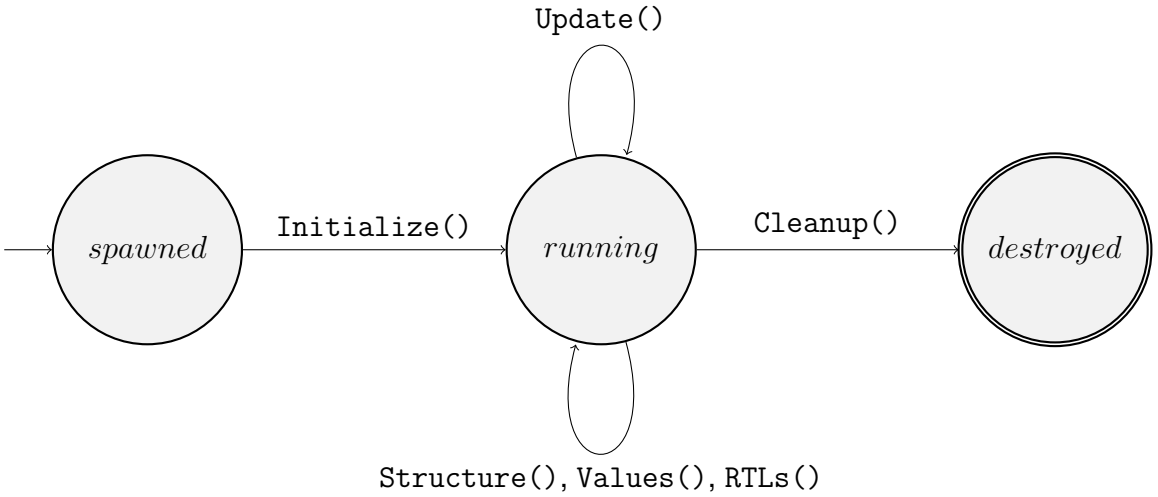


Figure 3.8: Life-cycle of a trust model instance with API calls. The `Update()` operations apply changes to the trust model instance. The `Structure()`, `Values()` and `RTLs()` functions are used to get the state of the trust model instance and run the TLEE calculation & TDE execution.

Based on the received evidence, each TSQ-I creates an atomic trust opinion. This is a subjective trust opinion calculated by a TSQ-I that has not been fused or discounted with trust opinions provided by other TSQ-Is. For one trust relationship in the trust model instance, several trust sources can be specified. The TSM is aware of which trust sources should be used for each trust relationship. After the corresponding TSQ-Is have calculated the atomic trust opinions for the specified trust sources of a trust relationship, the TSM fuses the atomic trust opinions to a trust opinion which is assigned to the corresponding trust relationship. For the fusion of the trust opinions, different fusion operators are possible. The operator to be used for the fusion is also specified in the trust model instance for each trust relationship. An overview of the described approach is shown in Figure 3.9.

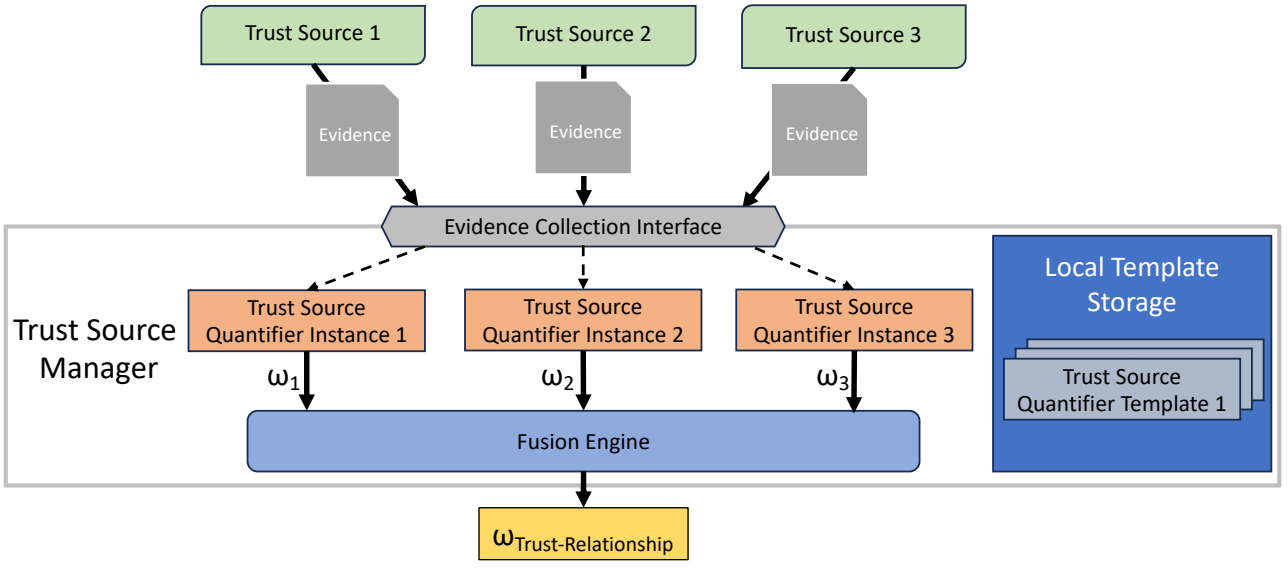


Figure 3.9: High level overview of the trust source manager handling receiving evidence and calculating a trust opinion for a single trust relationship.

3.7.2 Trust Source Programming Model

The user-facing programming model for trust sources consists of two separate APIs: an API to connect trust sources to the TAF (Trust Source Provider API) and an API to quantify concrete evidence as part of a trust model (Trust Source Quantifier API).

The Trust Source Provider API is TAF-global, meaning that a specific implementation of that API enables a trust source for the entire TAF and all of its potential trust models. In contrast, the Trust Source Quantifier API is specific to a trust model implementation. This means that different trust model can use the same trust source, however they can use different quantification mechanisms to derive their trust opinions from that source. Even further, quantifiers are instantiated for each session and can be configured via optional parameters. So there could be two sessions that use the same type of trust model with the same trust sources and the same quantification mechanism, but the output of the trust source quantifiers of the two sessions could still vary due to different per-session configurations (e.g., changing the weights for certain evidence items).

Trust Source Provider API

The Trust Source Provider API defines how arbitrary trust sources can be integrated into the TAF. It specifies how any network message or network service endpoint can be turned into a trust source that can be processed and integrated by the TAF at runtime.

Although this API is implemented as an experimental, proof-of-concept feature, the final prototype of the TAF is shipped with hard-coded trust sources as defined by the specific use cases.

The Trust Source Provider API differentiates various types of trust sources, depending on the interaction scheme and semantics:

Pull-based sources are trust sources that require the TAF to actively request evidence. This can either be done periodically, or based on some internal event (e.g., a client application asking the TAF for latest assessments, which requires the TAF to update its knowledge about pull-based sources).

Push-based sources are trust sources that notify the TAF in case of updates. These notifications are in most cases either received via broadcasts or they require an active subscription of the TAF at the trust sources.

Another parameter is the scope of a trust source when used inside the TAF for populating trust models:

TAF-scoped sources are trust sources where a single instance of the trust source provider can be used for the entire TAF. Hence, evidence updates to various trust models can be fed this single provider.

Session-scoped sources are trust sources that require a specific subscription for a trust assessment session. This is usually the case when the session requires configurable subscriptions that are only valid for that session, and evidence updates cannot be shared with other sessions using the same trust source.

Trust Source Quantifier API

The Trust Source Quantifier API consists of a single, user-provided function that converts arbitrary evidence into an atomic trust opinion.

The function is stateless, but can access configuration parameters (e.g., weights) of the associated session:

```
func(values map[EvidenceType]interface) subjectivelogic.QueryableOpinion
```

A quantifier function to convert evidence from a trust source into an atomic trust opinion.

`values`	As input, the function expects a map in which the key specifies the concrete type of evidence and the value encodes the corresponding raw evidence. The exact type of raw evidence varies and depends on the evidence type (e.g., boolean value, numerical value).
returns	The quantifier returns the atomic trust opinion calculated based on the set of evidence values provided as input.

3.8 Prototype Details

3.8.1 Enabling Technologies

Golang

One of the initial technical decisions concerned the selection of an appropriate programming language for implementing the TAF prototype. This choice was guided by four key requirements: (i) the language had to support high-level abstractions to facilitate rapid prototyping, (ii) it needed to run reliably across a range of architectures and systems, (iii) it had to offer efficient runtime performance, and (iv) it had to be compatible with execution inside a trusted execution environment, specifically Gramine.

Based on these criteria, an initial shortlist included C/C++, Rust, and Go. Scripting languages such as Python or Node.js were excluded early on due to their limited performance characteristics. Java and other JVM-based languages were also discarded, primarily because of insufficient support for Gramine.

Although Rust and C/C++ fulfilled the technical requirements, they posed a significant productivity barrier: the majority of the teams involved had limited practical experience with these languages. In contrast, Go provided a more accessible development environment without sacrificing runtime efficiency or deployment flexibility. As a result, Go was selected as the implementation language for the TAF prototype.

Apache Kafka

Kafka was chosen as the message layer for the project to provide a robust and language-agnostic abstraction for communication between the TAF and other external components. Its publish-subscribe model offers a clean separation between message producers and consumers, enabling

flexible integration of heterogeneous components. Kafka supports arbitrary message contents, allowing us to transmit structured data such as JSON without imposing rigid serialization constraints. Additionally, the rich ecosystem of Kafka tooling—including connectors, monitoring solutions, and client libraries for numerous programming languages—facilitates development, testing, and operations across the entire prototype architecture.

JSON

To enable structured, interoperable communication between the TAF and external components, JSON was chosen as the message format. JSON offers several advantages in this context: it is language-agnostic, human-readable, and widely supported across platforms and programming environments.

To define and document message types, JSON Schemas were employed. This approach not only standardizes message structures but also facilitates automated validation during development and at runtime.

Within the TAF prototype and accompanying helper applications, established libraries and tooling were used to serialize and deserialize (marshal/unmarshal) JSON messages to and from native data structures. These tools also enforce schema compliance during message processing to ensure consistency and correctness across system boundaries.

3.8.2 Key Features

The following list showcases the main implementation features of the final prototype, in addition to the other fulfilled requirements evaluated in Section 3.9.

Generic Platform We have succeeded in implementing the TAF prototype as a single platform that covers all use cases, trust models, trust sources, and target environments with one code base and as a single-packaged software application. This also includes federation features.

Portability & Adaptability Being a generic software platform, the TAF provides builds for several hardware architectures as well as wide range of configuration options. This allows the TAF to be used on hardware-restricted architectures with a low level of configurable parallelism while the same implementation can also be executed on a many-core server with a different CPU architecture and using a high number of parallel workers. In doing so, the same TAF implementation can conceptually be run on both vehicles and MEC servers, without any necessary code changes. Important configuration options include the level of parallelism (i.e., number of TAS workers), debugging configurations (i.e., log levels, auxiliary debug output files), and the configuration of other components (e.g., communication endpoints, TLEE version to be used).

Pluggable Trust Models An important design decision was the separation of trust model implementations and the TAF as a generic runtime platform to execute arbitrary trust models at runtime. This allows the implementation of new trust models against the APIs provided by the TAF without a strong coupling between the TAF internals and the trust model implementations. In fact, the Trust Source and Trust Model programming models provide an abstraction in such a way that all internal complexities of TAF are hidden and the trust models cannot access internal logic of the TAF beyond the defined APIs. New trust models can

be added to the TAF simply by adding their code implementations into a new trust model folder of the plugin folder of the TAF.

Web UI One additional feature of our TAF prototype is the optional web-based user interface that allows to explore sessions and trust model instance while the TAF is running. This is not only very helpful as a development and debugging feature, it also helps to explain and understand the concepts and mechanisms of the TAF in a live environment.

3.8.3 Featured Trust Models and Trust Sources

In total, the prototype TAF implementation features seven different trust models which rely on four different types of trust sources, as shown in Table 3.4.

Use Case	Trust Models	Trust Sources
1. Cooperative Adaptive Cruise Control	VCM@0.0.1	AIV
2. Intersection Movement Assist	IMA_STANDALONE@0.0.1	TCH, MBD
	IMA_STANDALONE@0.0.2	TCH, MBD
	NTM_STANDALONE@0.0.1	TCH, MBD
	IMA_FEDERATED@0.0.1	NTM, MBD
3. Task Offloading	TO@0.0.1	TCH
Slow Moving Traffic Detection	SMTD@0.0.1	TCH, MBD

Table 3.4: Scenarios, Trust Models, and Trust Sources.

3.9 Evaluation

This section evaluates the fulfillment of technical requirements and KPIs of the TAF as discussed in Chapter 5 of D2.2. These requirements include *functional requirements* (FR.TR.1 Generalizability, FR.TR.4 Correctness, FR.TR.6 Flexibility of Trust Sources, FR.TR.5 Robustness and Resilience) as well as *non-functional requirements* (FR.TR.2 Performance, FR.TR.3 Scalability).

3.9.1 Evaluation Strategy

The evaluation strategy consists of two separate approaches: (i) For most functional requirements, the outcome of the actual prototype is compared against the criteria of the original requirements. In most cases, this is a simple comparison of the actual state of the prototype with the desired target state. (ii) For the remaining requirements, performance evaluations are executed and the results of these measurements are compared against KPIs. This approach requires the actual execution of the prototype in an appropriate runtime environment and corresponding workloads.

3.9.2 Generalizability (FR.TR.1)

This requirement states that the final TAF prototype should be able to handle *three heterogeneous CCAM use cases*, capturing in-vehicle, vehicle to vehicle and vehicle to MEC (and vice versa) scenarios as well as different trust relationships.

As shown in Table 3.4, the TAF prototype supports seven different trust models which rely on four different types of trust sources. These trust model map to three heterogeneous use cases and cover five different scenarios:

1. Use Case: Cooperative Adaptive Cruise Control ✓
   (a) Scenario: Vehicle Computer Migration ✓
      - Trust Models: `VCM@0.0.1`
      - Instantiation: application-triggered
      - Trust Sources: AIV
      - in-vehicle
2. Use Case: Intersection Movement Assist ✓
   (a) Scenario 1: Federated Setup ✓
      - Trust Models: `IMA_FEDERATED@0.0.1`, `NTM_STANDALONE@0.0.1`
      - Instantiation: TCH-triggered (MEC), CPM-triggered (vehicles)
      - Trust Sources: MBD, TCH, NTM
      - vehicle-to-vehicle (CPMs), MEC-to-vehicle (federated)
   (b) Scenario 2: Standalone Setup ✓
      - Trust Models: `IMA_STANDALONE@0.0.1`, `IMA_STANDALONE@0.0.2`
      - Instantiation: CPM-triggered (vehicles)
      - Trust Sources: MBD, TCH
      - vehicle-to-vehicle (CPMs)
3. Use Case: Task Offloading/Slow Moving Traffic Detection ✓
   (a) Scenario: Task Offloading ✓
      - Trust Models: `TO@0.0.1`
      - Instantiation: TCH-triggered
      - Trust Sources: TCH
      - vehicle-to-MEC
   (b) Scenario: Slow Moving Traffic Detection ✓
      - Trust Models: `SMTD@0.0.1`
      - Instantiation: CPM-triggered (vehicles)
      - Trust Sources: MBD, TCH
      - vehicle-to-vehicle (CPMs)

Given the wide range of different use cases, scenarios, trust models, trust sources, and interactions covered by our TAF prototype, we can confidently confirm its generalizability.

3.9.3 Run-Time Performance (FR.TR.2) & Scalability (FR.TR.3)

In terms of the run-time performance, we had targeted a processing time (latency) below 100 ms for the arrival of new evidence up until the calculation of the corresponding ATL result value.

For the scalability, we extended the run-time performance considerations by also adding more concurrent trust models (i.e., the TAF has to maintain multiple trust model in parallel) and higher update rates (i.e., new evidence arrives in higher frequencies).

We combined the analysis of both non-functional requirements by executing performance evaluations of the TAF.

Evaluation Goals

The specific goal of this evaluation is the measurement of *latency* (i.e., time between receiving an evidence update and providing a new ATL) as a function of (i) different trust models, (ii) varying numbers of concurrent trust models, (iii) varying frequencies of evidence updates, and (iv) different hardware setups.

This evaluation should also explore whether the original latency requirements can still be fulfilled under higher load scenarios.

Evaluation Setup

The experimental setup includes four parameters (i.e., trust model, scenario size, runtime environment, update rates) with varying experimental levels, as defined below:

Trust Model This parameter defines the trust model that runs on the evaluated TAF. For the measurements, we selected two distinct trust models:

- `IMA_STANDALONE@0.0.1`: A vehicle-based trust model that observes the trustworthiness of another vehicle as well as observations of that vehicle. For the evaluation, the number of observation is hardcoded to 16, meaning that each trust model instance consists of 18 trust nodes (16 observations, ego vehicle, target vehicle), as well as 33 trust relationships (relationships from ego and target vehicle to the observations, relationship from ego to target vehicle).
- `TO@0.0.1`: A MEC-based trust model that observes the trustworthiness of a target vehicle based upon TCH reports. This trust model is limited to two trust nodes and a single trust relationship for each instance (MEC and target vehicle).

In Deliverable D6.1, it was initially specified that performance evaluations would be conducted in the context of the CACC use case. However, to enable a more meaningful assessment of run-time performance and scalability, the trust model for the Intersection Movement Assist (IMA) use case was selected instead. The IMA scenario is significantly more complex and involves a larger number of nodes, making it better suited for evaluating run-time performance and scalability. Given that the trust model for the CACC use case is much simpler and involves far fewer nodes, its performance results would outperform those obtained with the more complex IMA model. Therefore, using the IMA use case for evaluation provides a more rigorous test and does not impose any limitations.

Scenario Size The scenario size defines how many other vehicles exist in the evaluation run. Thus, this parameter specifies the number of parallel trust model instances at runtime, as the TAF spawns a trust model instance for each observed entity (in both trust models). Scenarios range from sparse to very crowded setups:

- 16 other vehicles
- 64 other vehicles
- 256 other vehicles
- 1,024 other vehicles

Runtime Environment The runtime environment specifies the system on which the TAF gets executed. Here, we selected a server variant that corresponds to the MEC side as well as a hardware-constraint machine to emulate a vehicular computer for the evaluation.

- *server machine* (AMD EPYC 9274F with 24 cores and 48 threads, 192GB RAM with 10Gbit NIC) running Ubuntu 24.04 LTS (Kernel: 6.8.0-51-generic)
- *Raspberry Pi 3* (Cortex-A53 quad core CPU, 512 MB RAM with a 1Gbit NIC) running Raspberry Pi OS Lite / Debian 12 (Kernel: Linux 6.6.74+rpt-rpi-v8)

Update Rates This parameter specifies the frequency in which other entities send evidence updates to the TAF. Note that resulting total load in terms of induced throughput is the product of the scenario size and the update rate:

- 1 update per second
- 10 updates per second

In addition to the runtime environment the TAF runs on (i.e., server vs. Raspberry Pi), the testbed consisted of two additional, dedicated machines with a setup identical to the *server*: a Kafka broker and a client generating and inducing the workload. For the broker, we used Apache Kafka 3.7.1 in the default configuration. The workload was created using `xk6-kafka 0.31.1`,

Not all of the hypothetical $2 \times 4 \times 2 \times 2$ setups had been explored. Instead, the MEC trust model had only been assigned to the MEC runtime environment while the vehicular trust model had only been used on the Raspberry Pi. We also capped the maximum load at a rate of 2,560 updates per second for the vehicular setup.

We have deployed the TAF prototype at v0.4.1 using a configuration that featured the internal TLEE and a thread pool size equivalent to the corresponding system architecture of the runtime environment. Debugging outputs and the interactive web frontend had been disabled in the evaluation builds.

For each configuration of the evaluation, we reset the testing environment, deployed the TAF, and then induced the target workload (i.e., evidence update messages) for 60 seconds after a warm-up phase that we excluded from the measurements. Within the 60 seconds, we measured and collected the processing latency for *all* incoming evidence updates using `/usr/bin/time` and `perf`, respectively. We also controlled the actual Kafka message consumption rate at the TAF to make sure that the actual throughput is kept. After the collection, we analyzed the distribution of latency measurements. In addition, we measured mean CPU utilization and memory consumption for the TAF during these runs.

Experiments & Outcomes

In summary, measurements had been conducted for the following run configurations for the vehicular trust model:

- Raspberry PI ×`IMA_STANDALONE@0.0.1` × 16 vehicles × 1/s updates
- Raspberry PI ×`IMA_STANDALONE@0.0.1` × 16 vehicles × 10/s updates
- Raspberry PI ×`IMA_STANDALONE@0.0.1` × 64 vehicles × 1/s updates
- Raspberry PI ×`IMA_STANDALONE@0.0.1` × 64 vehicles × 10/s updates
- Raspberry PI ×`IMA_STANDALONE@0.0.1` × 256 vehicles × 1/s updates
- Raspberry PI ×`IMA_STANDALONE@0.0.1` × 256 vehicles × 10/s updates

Likewise, the evaluation of the MEC-based trust model was conducted using the following runs:

- Server × `TO@0.0.1` × 64 vehicles × 1/s updates

Setup	Proc. Times (ms)		Resource Usage	
	p_{50}	p_{99}	CPU Util.	Mem (kB)
16 × 1/s	1.854	2.612	1.50%	21,504
16 × 10/s	1.439	2.304	12.75%	22,548
64 × 1/s	1.746	2.218	5.75%	23,400
64 × 10/s	1.317	9.730	36,75%	24,360
256 × 1/s	1.127	4.737	16.50%	29,580
256 × 10/s	1,385.2	1,681.6	78.75%	45,756

Table 3.5: Results for the *vehicle*-oriented measurements using the Raspberry Pi.

Setup	Proc. Times (ms)		Resource Usage	
	p_{50}	p_{99}	CPU Util.	Mem (kB)
64 × 1/s	0.185	0.279	0.08%	36,480
64 × 10/s	0.093	0.187	0.48%	39,456
256 × 1/s	0.186	0.269	0.29%	40,692
256 × 10/s	0.106	0.276	1.81%	42,164
1024 × 1/s	0.156	0.309	0.96%	43,776
1024 × 10/s	0.949	95.944	5.10%	52,448

Table 3.6: Results for the *MEC*-oriented measurements using the 24-core server.

- Server × `TO@0.0.1` × 64 vehicles × 10/s updates
- Server × `TO@0.0.1` × 256 vehicles × 1/s updates
- Server × `TO@0.0.1` × 256 vehicles × 10/s updates
- Server × `TO@0.0.1` × 1,024 vehicles × 1/s updates
- Server × `TO@0.0.1` × 1,024 vehicles × 10/s updates

The collected latency measurements have resulted in the following median (p50) and p99 processing times, as shown on in Table 3.6 and Table 3.5.

Discussion of Results

The evaluation results show that the TAF prototype is able to cope with increasing loads, even when running in resource-constrained environments.

The vehicle-oriented evaluations show good performance results for small and moderate loads. Concurrently maintaining 256 trust models with a total of 4,608 trust objects and 8,448 trust relationships still yields 99th percentile latencies below 5 ms for 256 updates per second. Once the update rates exceed 1,000 updates per second, there is a significant increase in CPU utilization and also the average processing latency then exceeds 1 second. This result is in line with our assumptions, as the TAF prototype has neither been optimized for outdated single-board computer architectures, nor does it currently apply load shedding in overload situations (i.e., drop updates to maintain latency thresholds).

The MEC-oriented evaluations indicate that the TAF can easily handle more than thousand parallel trust model instances with update rates above 10,000 updates per second. Here, more than

99% of updates are handled within 100 ms, while half of the updates are processed even within less than 1 ms. At the same time, the memory footprint is negligible and the CPU utilization indicates that the TAF could even handle much more computationally complex updates at similar rates.

That said, we argue that both the runtime performance requirements as well as the scalability requirements have been generally met by out prototype TAF.

3.9.4 Correctness (FR.TR.4)

This requirement states that the TAF must be able to produce a correct ATL for a proposition. More precisely, the requirement specifies that at least 70% of attacks should be detected by the TAF. To evaluate this, we use the trust model for the **Cooperative Adaptive Cruise control** and for the *Intersection Movement Assist (Scenario 2)*. With these two trust models, all trust sources and almost all trust topologies are reflected and evaluated.

For each trust model we specify several scenarios. For each scenario we describe the attack, the corresponding evidences and the expected output of the TAF. By comparing the expected output with the actual output of the TAF, we evaluate the correctness of the TAF.

For all scenarios, we use the projected probability to make a trust decision. For this purpose, we always use an RTL of 0.7. This value was determined empirically. However, in real world scenarios an extensive analysis would be necessary to determine an appropriate RTL (see Chapter 10).

Cooperative Adaptive Cruise Control

The trust model instance for the Cooperative Adaptive Cruise Control is shown in Figure 3.10. For this use case, the AIV is used as a trust source to calculate the trust opinions of the individual trust relationships. The types of evidence provided by the AIV to the TAF are shown below. For each security control, the AIV indicates whether the control is implemented and whether it has detected malicious behavior.

1. Secure Boot
2. Secure Over The Air Update
3. Access Control
4. Application Isolation
5. Control Flow Integrity
6. Configuration and Integrity Verification

In our evaluation, we focus on the proposition VC_1 in the trust model instance, since the calculations for the proposition VC_1 and VC_2 are exactly the same. In the following, be provide two scenarios, one scenario in which VC_1 is benign and one scenario in which VC_1 is malicious. For each scenario, the behavior of the TAF is evaluated.

Benign Scenario In the benign scenario, VC_1 is not compromised. Therefore, all foreseen security controls are active in VC_1 and did not detect any malicious behavior. The detailed values for each security control provided by the AIV to the TAF are shown in the following.

1. Secure Boot: implemented and no detection

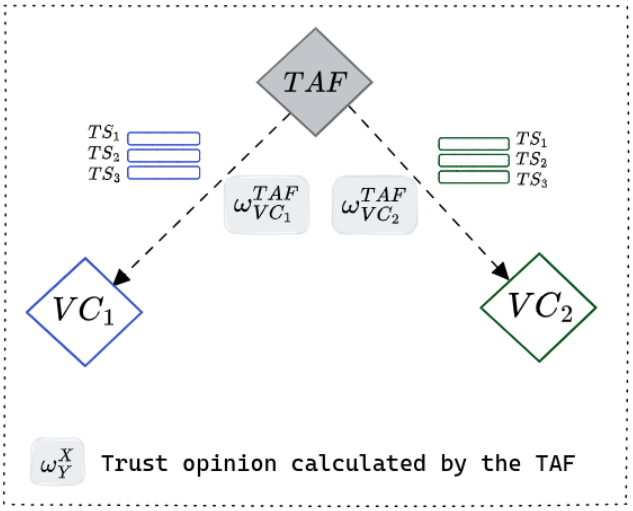


Figure 3.10: Trust Model Instance of CACC use case

2. Secure Over The Air Update: implemented and no detection
3. Access Control: implemented and no detection
4. Application Isolation: implemented and no detection
5. Control Flow Integrity: implemented and no detection
6. Configuration and Integrity Verification: implemented and no detection

Since VC_1 is benign, the expected output of the TAF is that the proposition VC_1 is trustworthy. Based on the specified evidences, the TAF provides the trust opinion $\omega_{ATL} = (0.76, 0.1, 0.14)$ for VC_1. The disbelief derives from the design time trust opinion specified for this use case. The uncertainty derives from the specified weights for each security controls. Further details about the quantification method are described in Chapter 9. This results to the projected probability of 0.83. As the RTL is 0.7, the TAF decides that VC_1 is trustworthy. Thus, the TAF provides a **correct decision** for this scenario.

Malicious Scenario In the malicious case we assume that the operating system of VC_1 is compromised. In this scenario, all foreseen security controls in VC_1 are active. The security control *Secure Boot* failed as the OS is compromised. Therefore, VC_1 is not trustworthy. The detailed values for each security control provided by the AIV to the TAF are as follows:

1. Secure Boot: implemented and detection
2. Secure Over The Air Update: implemented and no detection
3. Access Control: implemented and no detection
4. Application Isolation: implemented and no detection
5. Control Flow Integrity: implemented and no detection
6. Configuration and Integrity Verification: implemented and no detection

Based on the specified evidence, the TAF provided the trust opinion $\omega_{ATL} = (0.0, 1.0, 0.0)$ for VC_1. This results in the projected probability of 0.0. As the RTL is 0.7, the TAF decides that VC_1 is not trustworthy. Thus, the TAF provides a *correct decision* for this scenario.

Intersection Movement Assist

The trust model instance for the Intersection Movement Assist (IMA) is shown in Figure 3.11. For this use case, the TCH and the MBD are used as trust sources. The types of evidence provided by the TCH are shown in the following. For each security control, it is provided whether the control is implemented and whether it has detected malicious behavior.

1. Secure Boot
2. Secure Over The Air Update
3. Access Control
4. Application Isolation
5. Control Flow Integrity
6. Configuration and Integrity Verification

The types of evidences provided by the MBD are shown in the following. For each misbehavior detector, it is provided if the detector has detected malicious behavior.

1. Distance Plausibility
2. Speed Plausibility
3. Speed Consistency
4. Position Speed Consistency
5. Kalman Position Consistency
6. Kalman Position Speed Consistency (Speed)
7. Kalman Position Speed Consistency (Position)
8. Local Perception Verification

In our evaluation, we focus on the proposition C_27_19 in the trust model instance, since the calculations are exactly the same for each proposition. In the following, we provide a scenario where C_27_19 is benign. Furthermore, we provide two scenarios in which C_27_19 is malicious. In the first malicious scenario, the TCH provides negative evidence. In the second malicious scenario, the MBD provides negative evidence. For each scenario, the behavior of the TAF is evaluated.

Benign Scenario In the benign scenario, the observation C_27_19 is benign. Therefore, the TCH and the MBD did not detect malicious behavior. The detailed values for each security control provided by the TCH to the TAF are as follows:

1. Secure Boot: implemented and no detection
2. Secure Over The Air Update: implemented and no detection
3. Access Control: implemented and no detection
4. Application Isolation: implemented and no detection
5. Control Flow Integrity: implemented and no detection
6. Configuration and Integrity Verification: implemented and no detection

The detailed values for each misbehavior detector provided by the MBD to the TAF are as follows:

1. Distance Plausibility: no detection
2. Speed Plausibility: no detection

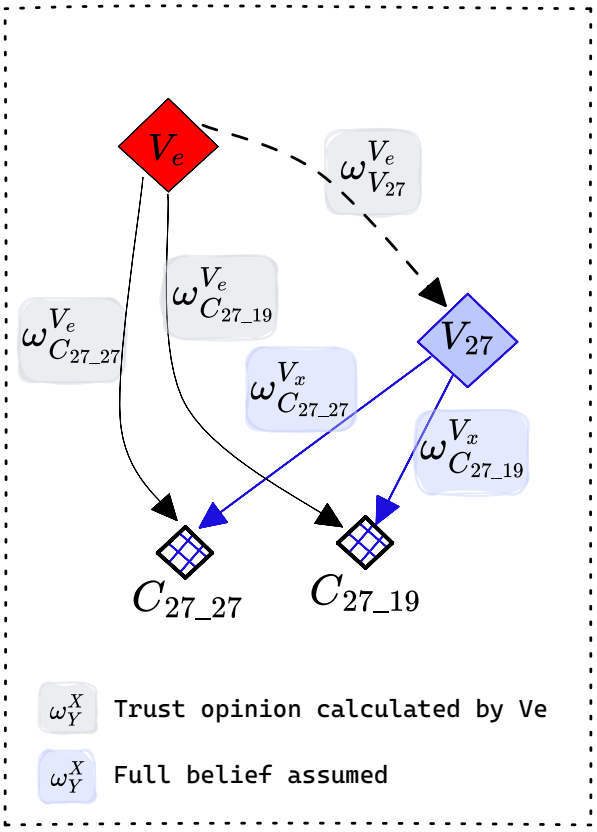


Figure 3.11: Trust Model Instance of IMA use case

3. Speed Consistency: no detection
4. Position Speed Consistency: no detection
5. Kalman Position Consistency: no detection
6. Kalman Position Speed Consistency (Speed): no detection
7. Kalman Position Speed Consistency (Position): no detection
8. Local Perception Verification: no detection

As the observation C_27_19 is benign, the expected output of the TAF is that the proposition is trustworthy. Based on the specified evidence, the TAF provided the trust opinion $\omega_{ATL} = (0.98, 0.0, 0.02)$ for the proposition C_27_19. Further details about the quantification method are described in Chapter 9. This results in a projected probability of 0.99. As the RTL is 0.7, the TAF decides that C_27_19 is trustworthy. Thus, the TAF provides a **correct decision** for this scenario.

Malicious Scenario - TCH In this malicious case, we assume that the operating system of the vehicle computer of the sending vehicle V_27 is compromised. Therefore, the secure boot of the vehicle computer failed. This results in the following values for each security control provided by the TCH to the TAF:

1. Secure Boot: implemented and detection
2. Secure Over The Air Update: implemented and no detection
3. Access Control: implemented and no detection
4. Application Isolation: implemented and no detection

5. Control Flow Integrity: implemented and no detection
6. Configuration and Integrity Verification: implemented and no detection

In contrast to the TCH, the MBD did not detect malicious behavior. This results in the following values for each misbehavior detector provided by the MBD to the TAF:

1. Distance Plausibility: no detection
2. Speed Plausibility: no detection
3. Speed Consistency: no detection
4. Position Speed Consistency: no detection
5. Kalman Position Consistency: no detection
6. Kalman Position Speed Consistency (Speed): no detection
7. Kalman Position Speed Consistency (Position): no detection
8. Local Perception Verification: no detection

Since the vehicle V_27 is compromised, there is a very high risk that the data sent is compromised as well. Therefore, the expected output of the TAF is that the proposition is trustworthy. Based on the specified evidence, the TAF provided the trust opinion $\omega_{ATL} = (0, 1, 0)$ for the proposition C_27_19. This results in the projected probability of 0.0. Since the RTL is 0.7, the TAF decides that C_27_19 is not trustworthy. Thus, the TAF provides a **correct decision** for this scenario.

Malicious Scenario - MBD In this malicious case, we assume that the position in the observation C_27_19 provided by vehicle V_27 has been compromised in the vehicle. However, none of the security controls have detected misbehavior, which leads to the following security controls provided by the TCH to the TAF:

1. Secure Boot: implemented and no detection
2. Secure Over The Air Update: implemented and no detection
3. Access Control: implemented and no detection
4. Application Isolation: implemented and no detection
5. Control Flow Integrity: implemented and no detection
6. Configuration and Integrity Verification: implemented and no detection

In contrast, the misbehavior detection system has detected malicious behavior. Since the position is compromised, all position related detectors have detected misbehavior. This results in the following values for each misbehavior detector, which are provided from the MBD to the TAF:

1. Distance Plausibility: detection
2. Speed Plausibility: no detection
3. Speed Consistency: no detection
4. Position Speed Consistency: detection
5. Kalman Position Consistency: detection
6. Kalman Position Speed Consistency (Speed): no detection
7. Kalman Position Speed Consistency (Position): detection
8. Local Perception Verification: detection

As the observation C_27_19 is compromised, the expected output of the TAF is that the proposition is not trustworthy. Based on the specified evidence, the TAF provided the trust opinion $\omega_{ATL} =$

$(0.59, 0.38, 0.03)$ for the observation C_27_19. This results in the projected probability of 0.6. As the RTL is 0.7, the TAF decides that the observation C_27_19 is not trustworthy. Thus, the TAF provides a **correct decision** for this scenario.

Summary

We have evaluated in total two different trust models, three different trust sources and five different scenarios. For each scenario, the expected output of the TAF is equal to the actual output of the TAF. In this way, we have shown that the calculations in the TAF provide the expected output and the TAF works correctly. Table 3.7 provides an overview of the individual use cases, scenarios, expected output of the taf and the actual output of the TAF. As 100% of the attack cases were detected correctly, the correctness-requirement is fulfilled.

Use Case	Scenario	Expected Output	Actual Output
Cooperative Adaptive Cruise Control	Benign Scenario	Trustworthy	Trustworthy ✓
Cooperative Adaptive Cruise Control	Malicious Scenario	Not Trustworthy	Not Trustworthy ✓
Intersection Movement Assist	Benign Scenario	Trustworthy	Trustworthy ✓
Intersection Movement Assist	Malicious Scenario 1	Not Trustworthy	Not Trustworthy ✓
Intersection Movement Assist	Malicious Scenario 2	Not Trustworthy	Not Trustworthy ✓

Table 3.7: Overview of the evaluation results of the correctness of the TAF.

3.9.5 Robustness and Resilience (FR.TR.5)

One core requirement related to the deployment of TAF has to do with the inherent need of resilience and robustness of the overall trust-related calculations. This would limit the attack vectors that may affect the trust assessment process and, consequently, the trust decisions of the TAF. Of course, one approach would be to include the overall TAF component as part of the CONNECT Trusted Computing Base presented in [3]. However, this contradicts the overarching effort of minimizing the TCB that needs to be maintained and measured. Consequently, an alternative approach would be to run it as a trusted component running in an isolated, confidential environment. Nevertheless, it is crucial to take into consideration that this requirement should not compromise the safety-critical CCAM applications running along with the TAF computations.

Based on this consideration, the aim is to evaluate the overhead that is inflicted by different flavors of trusted execution environments. To this extent, Chapter 6 presents a comparison of different TEE environments leveraging software- and hardware- based root of trust elements. Based on the findings of this evaluation, an interesting insight refers to the possibility of new secure element technologies (e.g., Intel TDX environments) to support the FR_TR_5 requirement without affecting the safety-critical operational profile of CCAM applications.

3.9.6 Flexibility of Trust Sources (FR.TR.6)

This requirement states that the final TAF prototype should be able to handle at least *three different trust sources*.

As shown in Table 3.4, the TAF prototype supports four different types of trust sources. These trust sources map to three heterogeneous use cases:

1. Use Case: Cooperative Adaptive Cruise Control ✓
   (a) Scenario: Vehicle Computer Migration ✓
      - Trust Models: `VCM@0.0.1`
      - Trust Sources: AIV
2. Use Case: Intersection Movement Assist ✓
   (a) Scenario 1: Federated Setup ✓
      - Trust Models: `IMA_FEDERATED@0.0.1`, `NTM_STANDALONE@0.0.1`
      - Trust Sources: MBD, TCH, NTM
   (b) Scenario 2: Standalone Setup ✓
      - Trust Models: `IMA_STANDALONE@0.0.1`, `IMA_STANDALONE@0.0.2`
      - Trust Sources: MBD, TCH
3. Use Case: Task Offloading/Slow Moving Traffic Detection ✓
   (a) Scenario: Task Offloading ✓
      - Trust Models: `TO@0.0.1`
      - Trust Sources: TCH
   (b) Scenario: Slow Moving Traffic Detection ✓
      - Trust Models: `SMTD@0.0.1`
      - Trust Sources: MBD, TCH

Given the wide range of different trust sources, which are the AIV, TCH, MBD and NTM covered by our TAF prototype, we can confidently confirm the flexibility of trust sources.

3.9.7 Evaluation Summary

We have evaluated our TAF in detail to analyze whether it fulfills each trust requirement and the KPIs specified for the requirement. The evaluation showed that the KPIs specified for each requirement are fulfilled. An overview of the requirement, the corresponding KPIs and the results of the evaluation are shown in Table 3.8.

Trust Requirement	KPI	Result
FR.TR.1 Generalizability	Three CCAM use cases	Three CCAM Use cases ✓ (Vehicle Computer Migration, Intersection Movement Assist, Task Offloading / Slow Moving Traffic Detection)

FR.TR.2 Run-time Performance	Less than 100 ms	In the most complex trust models we used in the evaluations (see Section 3.9.3), 99% are handled within 100 ms ✓ (Half of the requests are handled within less than 1 ms)
FR.TR.3 Scalability	For up to 50 trust objects, the response time should remain within 100ms	In the most complex evaluation setting with 4,608 trust objects, 99% or the requests are handled within 100 ms ✓ (Half of the requests are handled within less than 1 ms)
FR.TR.4 Correctness	Detection of at least 70% of the events of interest representing possible attacks	100 % detection of incidents of risk considering an existing compromise evaluated within our evaluations (see Section 3.9.4) ✓
FR.TR.5 Robustness and Resilience	TAF needs to be executed within a TEE	TAF can be executed within a TEE ✓
FR.TR.6 Flexibility of Trust Sources	At least three trust sources	Four trust sources ✓ (AIV, TCH, MBD, NTM)

Table 3.8: Overview of the evaluation results of the TAF.

Chapter 4

Trustworthiness Level Expression Engine

As explained in the previous section, the TSM is responsible for creating, or quantifying Trust Opinions on a level of Trust Relationships. Furthermore, as defined in [2], the Trust Model combines various trust relationships among different trust objects. In response, TLEE is responsible for assessing the trustworthiness on the level of the trust model (i.e., the trust network). To calculate the trust opinion for the whole trust network, a set of functionalities are necessary that are provided by different modules in the TLEE. In section 4.1, we first explain the high-level architecture and the functional specifications of the different components of the TLEE. In Section section 4.2, we present the mathematical definitions of Josang's trust discounting and fusion operators, along with our newly proposed trust discount operator, developed in response to newly identified requirements from the CONNECT project. Furthermore, in section 4.3, we describe the internal API of the TLEE with the TAM (as shown in Figure 3.1), where we detail on the interface of the TLEE with the rest of the TAF components. Lastly, we conclude this chapter with some development insights of the TLEE in section 4.4.

4.1 Architecture and Functional Specifications

In Figure 4.1, we show the high-level TLEE architecture. As previously explained in Section 3.5 and showed in Table 3.3, TLEE communicates with the TAM through a single internal interface ($TLEE_{Invoke}$ / `RunTLEE`) that triggers the execution of the five main components of the TLEE: *Sub-Trust Model Extractor*, *DSPG transformer*, *Expression Synthesizer*, *Meta-to-Concrete Expression Converter* and *Evaluator*. Firstly, the *Sub-Trust Model Extractor* takes an input a Trust Model. A Trust Model can comprise of a single or multiple propositions. The propositions are the variables in the leaves of the Trust Model, for which the TLEE needs to calculate the opinions, precisely, the actual trust values (ATLs). In fig. 4.2 and fig. 4.3, we show two Trust Models with multiple ($Rdata$, G, $Cdata$ and $Ldata$) propositions and single proposition (X), respectively. Currently, we consider different propositions as independent variables and we do not consider any contextual dependencies among the variables. Thus, the initial TLEE component extracts distinct (Sub-)Trust Models per proposition, which are then passed to the next modules of the TLEE. The remaining TLEE components—*the DSPG transformer*, *Expression Synthesizer*, *Meta-to-Concrete Expression Converter* and *Evaluator*—are each instantiated separately for every Sub-Trust Model, ensuring one dedicated instance per TM. Additionally, it is important to emphasize that in order to cope with the dynamic nature and the continuous changes in the structure of the trust models, *the DSPG transformer*, *Expression Synthesizer* and *Meta-to-Concrete Expression Converter*

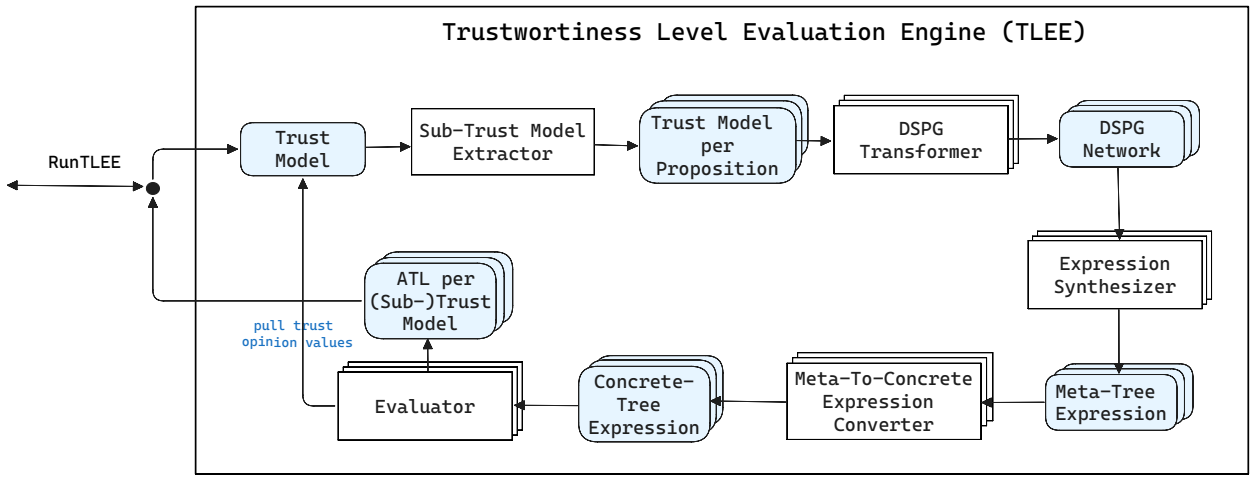


Figure 4.1: TLEE Architecture

operate symbolically by rewriting expressions on trust opinion variables. Lastly, the *Evaluator* calculates the final trust opinion or the Actual Trust Level (ATL) for each Sub-Trust Model.

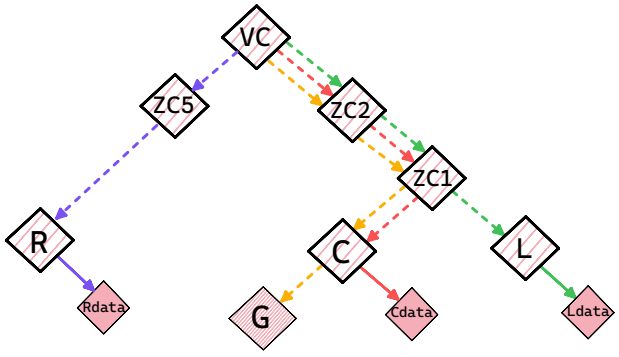


Figure 4.2: A Trust Model with multiple propositions

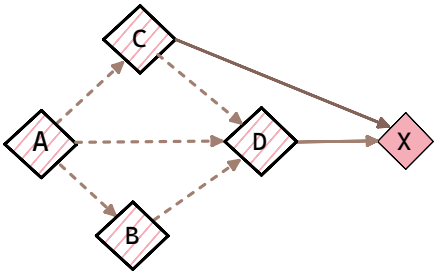


Figure 4.3: A Trust Model with a single proposition

The TLEE "reduces" the trust network into an equivalent single edge (from the root to the leaf of the network) for each proposition by applying trust discounting and fusion operations. This consolidation allows the derivation of a unified trust opinion, known as the Actual Trust Level (ATL). We also demonstrate this process in figures 4.4 and 4.5. As previously explained, fig. 4.4 shows an example of a trust model as received from the TAM, with four different propositions $Rdata$, G, $Cdata$ and $Ldata$. All the trust opinions (ωs) that are depicted as part of this trust model are on a level of trust relationship. These opinions are calculated and quantified by the TSM, and their numerical values (quadruples of $(belief, disbelief, uncertainty, baserate)$) are sent to the TLEE as part of the Trust Model.

To summarize, the TLEE is responsible for aggregating the trust opinions on the trust relationships that have been previously calculated by the TSM, and passed to the TLEE through the Trust Model. Based on this aggregation—trust discounting and fusion—the final ATL (per proposition) is derived. Figure 4.5 shows four ATLs (ω^{VC}_{Rdata}, ω^{VC}_{G}, ω^{VC}_{Cdata} and ω^{VC}_{Ldata}), one for each proposition of the Trust Model. These consolidated edges encapsulates the comprehensive trust flow from the root of the trust model to the proposition, while considering the various trust relationships on

the path and their corresponding trust opinions. Finally, we provide the algebraic expressions of the trust discounting and fusion operators later on in this chapter, as part of section 4.2.

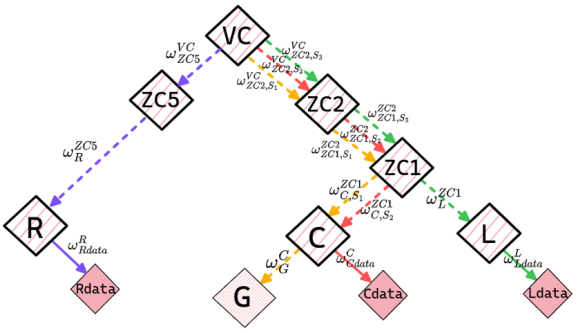


Figure 4.4: TM as received from the TSM

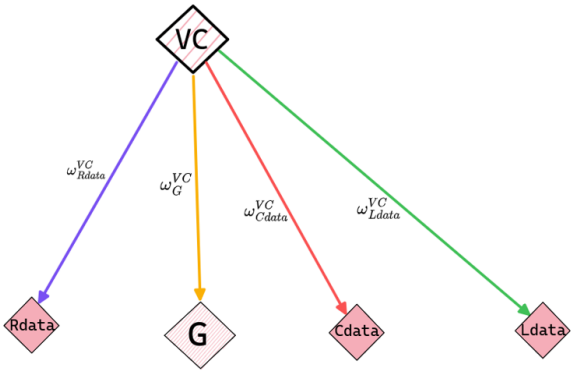


Figure 4.5: TM as aggregated by the TLEE

In the following, we explain in more detail the necessary processes to calculate the ATL values and the TLEE's functional specifications: the inputs, the outputs and a short summary of the functionality of the five main TLEE components.

Sub-Trust Model Extractor.

```
Input: trust model / trust network
Output: sub-TMs (one for each proposition)
```

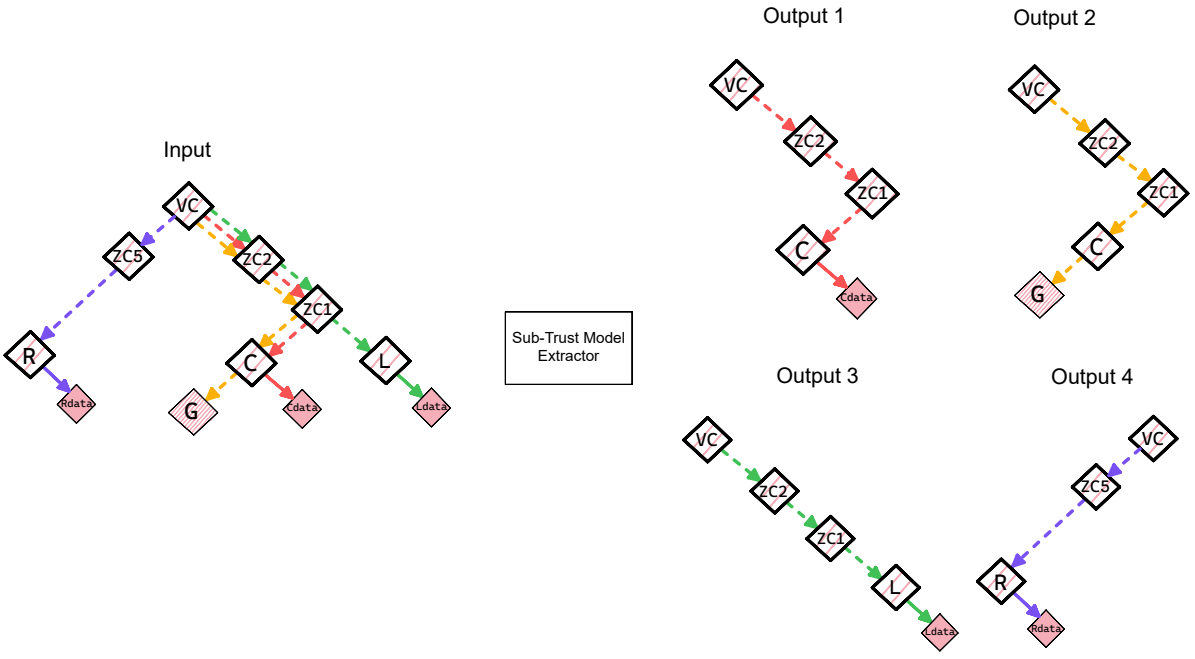


Figure 4.6: Sub-Trust Model Extractor.

Figure 4.6 shows the input and the output of the Sub-Trust Model Extractor. A Trust Model (TM) comprises multiple propositions, represented as leaf-node variables, for which the TLEE computes opinions or Actual Trust Levels (ATLs). In the current implementation, propositions are

treated as independent variables, with no consideration of contextual dependencies between them. In response, as the first component of the TLEE, the Sub-Trust Model Extractor isolates each proposition into its own (Sub-)Trust Model. Each extracted Sub-Trust Model serves as input to subsequent TLEE components, facilitating a modular evaluation framework where each proposition is processed independently.

DSPG Transformer.

```
Input: Sub-TM
Output: trust network in a DSPG format
```

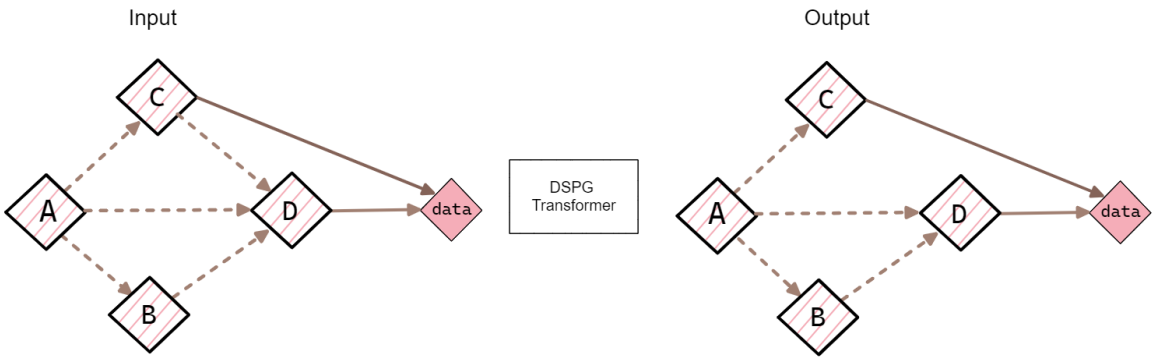


Figure 4.7: DSPG Transformer.

To illustrate the transformations enabled by the DSPG transformer, we adopted a different Trust Model compared to the one described earlier. Specifically, the sub-TMs depicted in Figure 4.6 focus solely on a single path from the root to the leaf, and these Trust Networks are already in a DSPG format. In this scenario, the TLEE only requires the discount operator to simplify these TMs into a single edge, eliminating the need for DSPG transformations.

Figure 4.7 shows the input and the output of the DSPG transformer. DSPG stands for Directed Series-Parallel Graph (DSPG) [12], and we have already given a detailed explanation on Subjective Trust Networks and DPSG, as part of section 5.3.4. in [4]. To shortly recap: a subjective trust network (STN) is a graph representation of trust and belief relationships from agents, via other agents and sensors to target entities/variables, where each trust and belief relationship is expressed as a subjective opinion [12]. Please note that throughout this chapter we might use the terms 1) trust model, 2) trust network and 3) STN, interchangeably. On the other side, for the trust network to be able to be analysed, i.e., for the operators for fusion and trust discounting to be applied to referral and functional trust relationships, trust network needs to be represented as a Directed Series-Parallel Graph (DSPG). DSPGs have a fundamental role in the TLEE implementation.

Definition 4.1.1 (Directed Series-Parallel Graph (DSPG) [12])**.** A graph is called a Directed Series-Parallel Graph (DSPG) if it can be decomposed as a combination of Series and Parallel graphs and it only consists of directed edges that form paths without loops from the source to the target.

By employing trust discounting and fusion operations, the DSPG trust network can be effectively transformed into an equivalent single edge, as previously shown in Figure 4.5. For that reason,

as part of the *DSPG Transformer*, we first check if the received trust network is in a DSPG format. If the trust network is a non-DSPG, then we do the necessary transformation.

To summarize, as input the DSPG transformer receives a trust network, which could be a complex non-DSPG or a DSPG graph. The functionality of this component is to first checks if the network is a DSPG graph. If not, it synthesizes the complex non-DSPG to a DSPG graph. To perform the transformation, we remove the smallest possible number of edges that break the DSPG structure. Alternatively, during the transformation, some additional information can be used as a heuristic (e.g., the opinions from the TSM) to make a more informed decision which edges to remove.

Expression Synthesizer.

```
Input: trust network in a DSPG format
Output: meta-tree expression
```

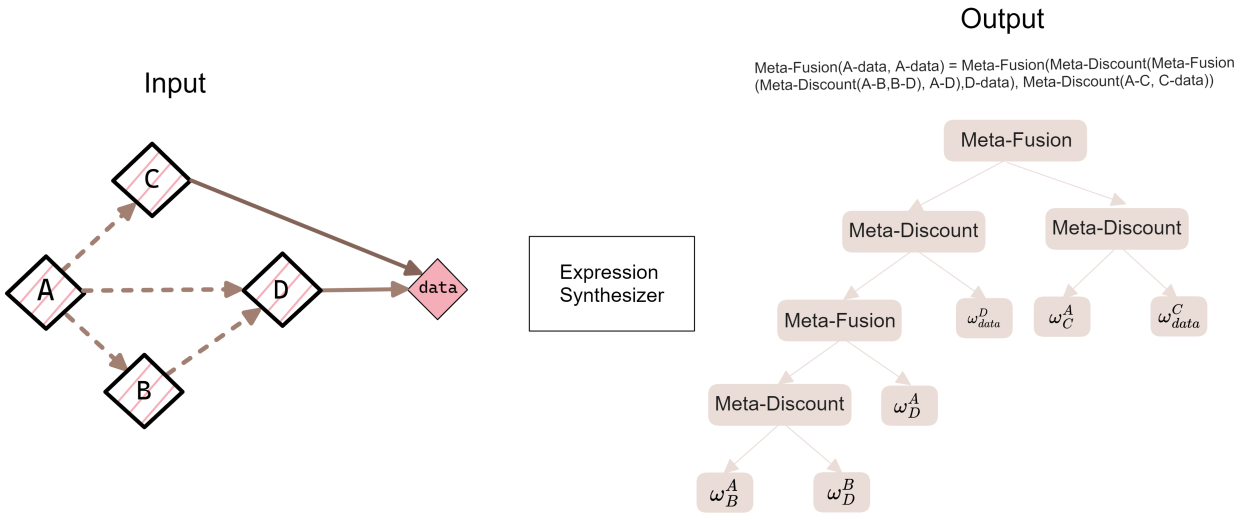


Figure 4.8: Expression synthesizer.

Figure 4.8 shows the input and the output of the *Expression Synthesizer*. As input this component receives a trust network in a DSPG format, and outputs a Meta-Tree Expression. The Meta-Tree Expression could be in tree or String format, as shown in the figure. Therefore, the functionality of this module is to build the Meta-Tree Expression from the trust network. This includes many sub-functionalities, where the two main one are: calculation of the nesting levels and graph analysis based on which the expression is built. Lastly, our aim is to build an overall solution that is flexible and generalizable. For that reason, we have designed an engine that is math model agnostic and operates on a symbolic level, in order to support future extensions to other mathematical theories that enable dynamic trust assessment beyond SL. As a result, this module produces a meta-tree expression, which we will explain in more detail later in this section.

Step 1: Nesting levels calculation

In Figure 4.9, we show the calculated nesting levels for the exemplary DSPG. The concept of nesting level is important for analysing trust networks represented as a DSPG. To understand and define nesting levels, we first need to explain the concept of parallel-path subnetwork (PPS). First,

a DSPG graph can consist of multiple subnetworks that themselves are DSPGs that can contain parallel paths [12]. Therefore, we are interested in identifying subnetworks within a DSPG that contain parallel paths. A parallel-path subnetwork (PPS) in a DSPG is the set of multiple paths between a pair of connected nodes (e.g., nodes A and D, or A and $data$ in Figure 4.9). The nesting level of an edge reflects how many PPSs the edge is part of in the DSPG. Each edge has a specific nesting level greater or equal to 0. For example, a trust network consisting of a single trust path, has trust edges with nesting level 0, because the edges are not part of any subnetwork of parallel paths. If we refer again to Figure 4.9, then we can see that the edges between nodes A and D are part of two PPSs, and have nesting level of 2; whereas the edges between nodes A and C, C and $data$, and D and $data$ are part of one PPS, therefore have nesting level of 1.

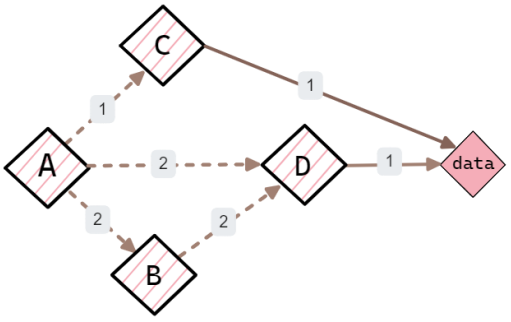
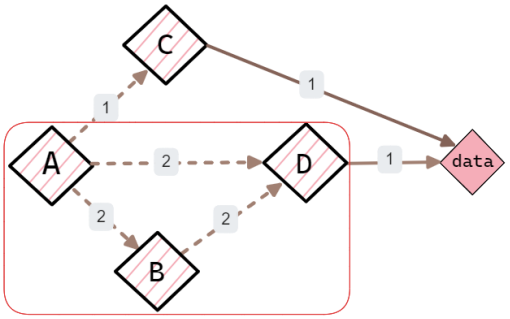


Figure 4.9: Nesting levels for the exemplary DSPG.

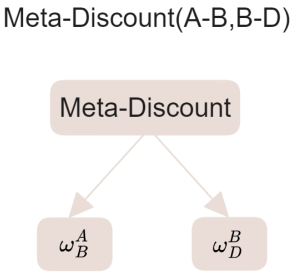
Step 2: Building Expression

Based on the calculated nesting levels, we start with building the expression. In the first phase in the process of building the expression, see Figure 4.10, we first discount the opinions ω_B^A and ω_D^B, to build the opinion ω_D^A. On the left hand-side of the figures you can see the trust network including its nesting levels, and how it changes during the graph reduction process. However, on the right hand-side of the figures, we show how the expressions are built iteratively in each of the phases. In the second phase, see figure 4.11, there are two opinions ω_D^A that need to be fused as part of this phase, to derive a single ω_D^A opinion. In the third and fourth phase, see figure 4.12 and figure 4.13, we discount the opinions ω_D^A and ω_{data}^D, and ω_C^A and ω_{data}^C, respectively. In this process we derive two opinions from A to $data$ (ω_{data}^A). In the final phase, see figure 4.14, we fuse both of the opinions from the fourth phase, and derive the final ω_{data}^A.

As previously explained, our aim is to build an overall solution that is flexible and generalizable. For that reason, we have designed an engine that is math model agnostic and operates on a symbolic level, in order to support future extensions to other mathematical theories that enable dynamic trust assessment beyond SL. For instance, using other formal frameworks for trust assessment under uncertainty, e.g., Epistemic SL or EBSL (”evidence-based subjective logic), or even using homomorphic approaches to abstract from a particular mathematical model, as part of future research efforts. As part of our design of the TLEE, we have implemented this by first introducing meta-operators (meta-fusion and meta-discount), as part of the Expression Synthesizer module. Hence, the output of this module is a meta-tree expression containing meta-operators that are instantiated to concrete operators in the next module.

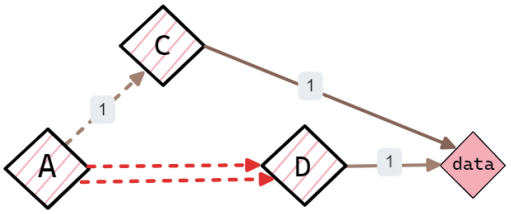


(a) Graph reduction

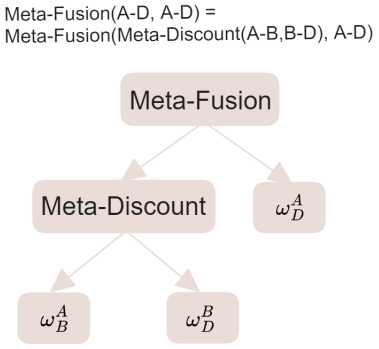


(b) Meta-tree expression

Figure 4.10: Building expression, phase 1.

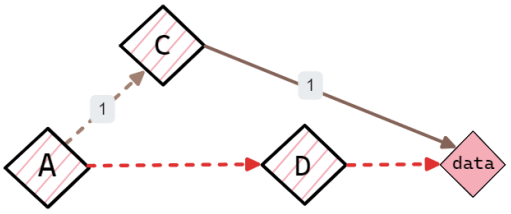


(a) Graph reduction

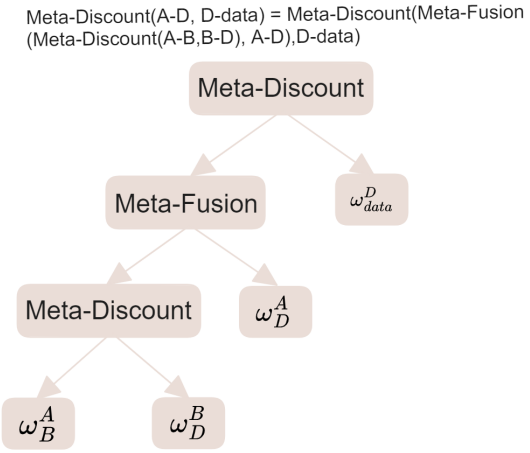


(b) Meta-tree expression

Figure 4.11: Building expression, phase 2.

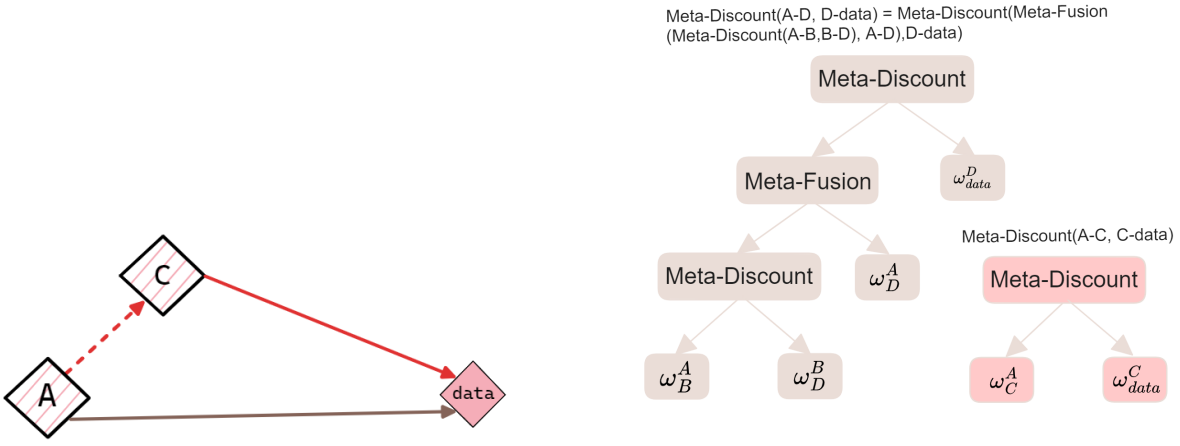


(a) Graph reduction



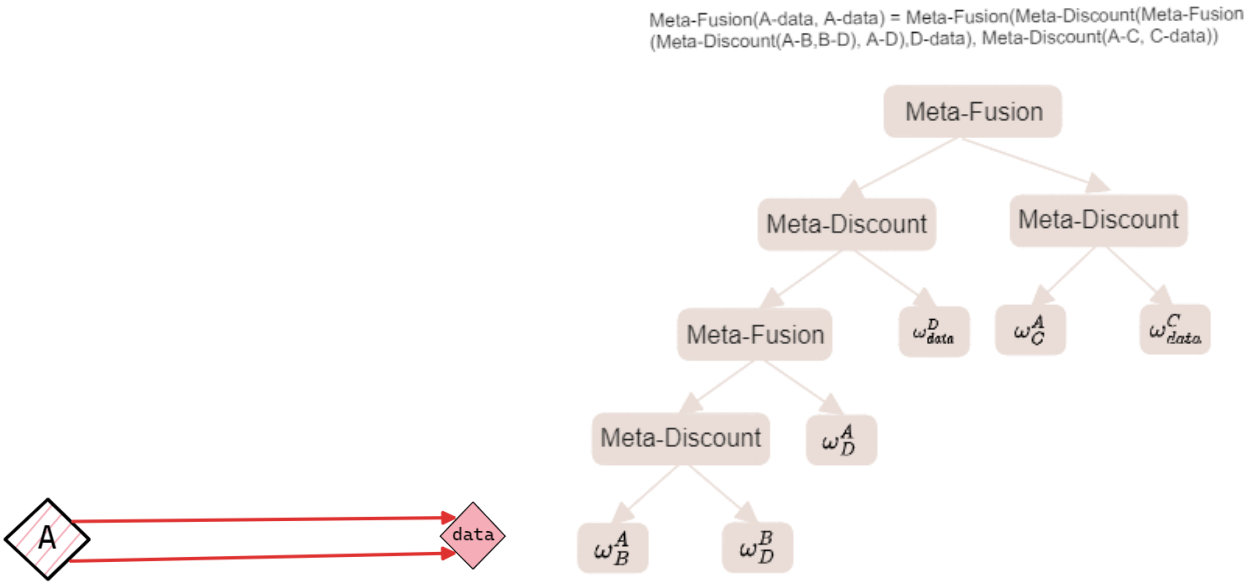
(b) Meta-tree expression

Figure 4.12: Building expression, phase 3.



(a) Graph reduction (b) Meta-tree expression

Figure 4.13: Building expression, phase 4.



(a) Graph reduction (b) Meta-tree expression

Figure 4.14: Building expression, phase 5.

Meta-to-Concrete Expression Converter.

```
Input: meta-tree expression (and optionally, math model & fusion operators)
Output: concrete-tree expression
```

We introduced a separate component *Meta-to-Concrete Expression Converter*, with a very simple functionality. Specifically, this component takes two inputs: 1) the meta-tree expression generated by the previous component, and 2) optionally, a mathematical model (such as SL, EBSL, etc.) along with fusion operators. It then converts these inputs into a concrete tree expression. In other words, the functionality of this component is to map the meta operators with concrete operators based on the specific mathematical model that can be given as an input parameter.

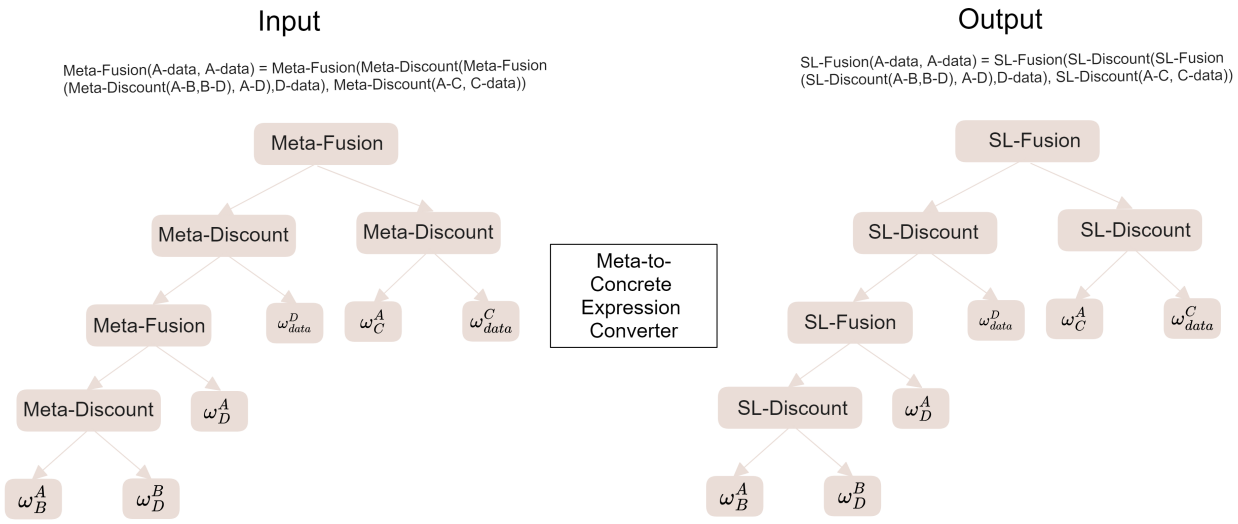


Figure 4.15: Meta-to-Concrete Expression Converter.

Since as part of CONNECT project, we are exclusively building the TAF with the subjective logic theory, the Math Model parameter (SL) is hard-coded in the TLEE, and all the information about the fusion operators are provided as part of the Trust Model. If the fusion operators are not already indicated in the trust model (i.e., the rewritten expression), then the component can also receive a parameter that specifies the concrete fusion operator (e.g., cumulative, average, etc.) that will be used for all the META-FUSION that do not have a specified fusion operator. As a result, the trust model and, in turn, the TLEE graph analysis can adjust to different use case demands.

Figure 4.15 shows the input and the output of the Meta-to-Concrete Expression Converter. As input the Meta-to-Concrete Expression Converter receives the Meta-Tree Expression from the previous module. The Meta-to-Concrete Expression Converter outputs a Concrete-Tree Expression in tree or String format.

Evaluator.

```
Input: concrete-tree expression + trust opinions on trust relationships
Output: ATL (one per sub-TN) and (optional) stringified Expression
```

Figure 4.16 shows the input and the output of the Evaluator module. As input, the Evaluator receives the Concrete-Tree Expression from the previous module and the fetches the values of the Trust Opinions on Trust Relationships from the Trust Model that is provided as input to the TLEE. As previously explained, the trust opinions on the level of trust relationships are sent by the TAM as part of the Trust Model, and calculated by the TSM.

To recap, as we already explained earlier in this section, until this component the TLEE engine rewrites the expressions symbolically, on a level of trust opinion *variables*. If we refer to figure 4.15, then those opinions variables are the leaves of the tree. After 1) fetching the *numerical* values of the Trust Opinions on Trust Relationships calculated by the TSM, and 2) applying the concrete discount and fusion operators, the final, *numerical* Trust Opinion for the Trust Network, or in other words, the ATL is calculated. The ATL(s) is the output of the Evaluator and, therefore, the TLEE.

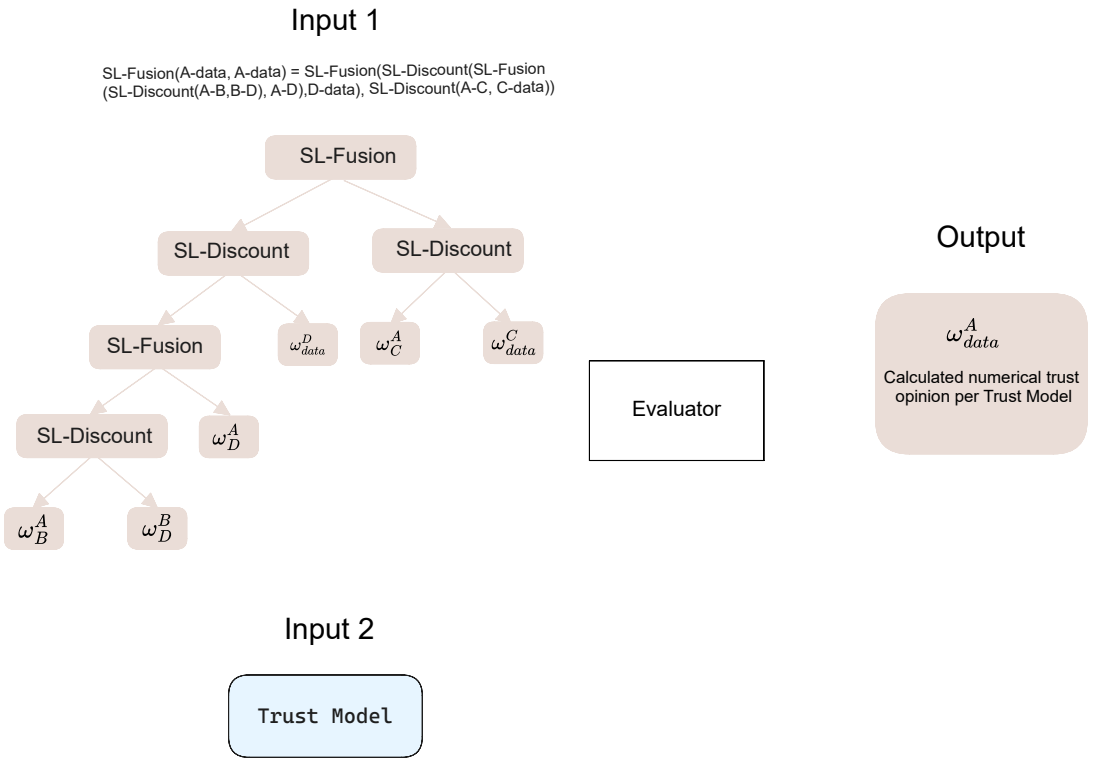


Figure 4.16: Evaluator.

In the following subsection, we provide more details on the formalisms behind different trust fusion and trust discount operators.

4.2 Trust Fusion and Discount Operators

As part of Subjective Logic theory, Jøsang [12] has proposed five *fusion operators*. In section 4.2.1, we explain each of the fusion operators, followed up with real-life, exemplary situations where the operators are applicable. Additionally, we provide the mathematical formulas for each of the operators. More details about the operators can be found in [12]. Additionally, part of prior works [13], Jøsang has proposed three operators for trust transitivity/discount. We provide a summary of those three operators in section section 4.2.1. As part of this section we also discuss several limitations of the proposed trust discounting operators by Jøsang. In response, as one of the many contributions that we have made as part of the CONNECT project, we have also proposed a novel trust discounting operator, which addresses some of the shortcomings that we have identified in the operators proposed by Jøsang, in regard to discounting trust on referral paths. We summarize the novel discounting operator in section 4.2.3. For further details about the novel discounting operator, refer to [14].

Finally, it should be noted that the trust fusion and discount operators are formally defined within the Trust Model, which serves as input to the TLEE. The selection of specific operators depends on the particular requirements and objectives of each use case scenario under examination.

4.2.1 Trust fusion operators

- **Belief Constraint Fusion** (BCF) [12] is suitable in situations where each agent expresses an opinion stating which variable values are the most correct, but in case of totally conflicting opinions the belief fusion is not allowed (i.e.,, an agreement between the expressed opinions is not possible). An example is when a group of friends try to agree on seeing a movie. If they share the same preferences for a movie, they can agree on watching that movie. However, if their preferences do not match then there is no agreement and they will not watch any movie together.
  The algebraic expression for the Belief Constraint Fusion operator is given as follows [12]:

$$\omega_X^{A\&B} : \begin{cases} b_X^{(A\&B)}(x) = \frac{\mathrm{Har}(x)}{(1-\mathrm{Con})} \\ u_X^{(A\&B)} = \frac{u_X^A u_X^B}{(1-\mathrm{Con})}, \\ a^{(A\&B)}(x) = \frac{a_X^A(x)(1-u_X^A)+a_X^B(x)(1-u_X^B)}{(2-u_X^A-u_X^B)}, & \text{for } u_X^A + u_X^B < 2, \\ a^{(A\&B)}(x) = \frac{a_X^A(x)+a_X^B(x)}{2}, & \text{for } u_X^A = u_X^B = 1. \end{cases}$$

  The term $Har(x)$ represents the relative harmony between constraints (in terms of overlapping belief mass) on x. The term Con represents the relative conflict between constraints (in terms of non-overlapping belief mass) between ω_X^A and ω_X^B.

$$Har(x) = b_X^A(x)u_X^B + b_X^B(x)u_X^A + \sum_{(x^A \cap x^B)=x} b_X^A(x^A)b_X^B(x^B),$$

$$Con = \sum_{(x^A \cap x^B)=0/} b_X^A(x^A)b_X^B(x^B).$$

- **Cumulative Belief Fusion** (CBF) [12] is when it is assumed that, as more and more (independent) sources are included, the amount of independent evidence increases. For instance, by applying this operator to a subscriber's historical location data, a mobile network operator can generate an opinion with progressively reduced uncertainty regarding the subscriber's frequented locations.
  Let ω^A and ω^B be source A and B's respective opinions over the same variable X. Let $\omega_X^{(A\diamond B)}$ be the opinion such that [12]:
  Case 1: For $u_X^A \neq 0 \vee u_X^B \neq 0$:

$$\begin{cases} b_X^{(A\diamond B)}(x) &= \frac{b_X^A(x)u_X^B + b_X^B(x)u_X^A}{u_X^A + u_X^B - u_X^A u_X^B} \\ u_X^{(A\diamond B)} &= \frac{u_X^A u_X^B}{u_X^A + u_X^B - u_X^A u_X^B} \\ a_X^{(A\diamond B)}(x) &= \frac{a_X^A(x)u_X^B + a_X^B(x)u_X^A - (a_X^A(x)+a_X^B(x))u_X^A u_X^B}{u_X^A + u_X^B - 2u_X^A u_X^B} & \text{if } u_X^A \neq 1 \vee u_X^B \neq 1 \\ a_X^{(A\diamond B)}(x) &= \frac{a_X^A(x)+a_X^B(x)}{2} & \text{if } u_X^A = u_X^B = 1 \end{cases}$$

  Case 2: For $u_X^A = u_X^B = 0$:

$$\begin{cases} b_X^{A\diamond B}(x) &= \gamma_X^A b_X^A(x) + \gamma_X^B b_X^B(x) \\ u_X^{A\diamond B} &= 0 \\ a_X^{A\diamond B}(x) &= \gamma_X^A a_X^A(x) + \gamma_X^B a_X^B(x) \end{cases}$$

where

$$\begin{cases} \gamma_X^A &= \lim_{\substack{u_X^A \to 0 \\ u_X^B \to 0}} \frac{u_X^B}{u_X^A + u_X^B} \\ \gamma_X^B &= \lim_{\substack{u_X^A \to 0 \\ u_X^B \to 0}} \frac{u_X^A}{u_X^A + u_X^B} \end{cases}$$

Then $\omega_X^{(A \diamond B)}$ is called the cumulatively fused opinion of ω_X^A and ω_X^B, representing the combination of the independent opinions of sources A and B.

- **Averaging Belief Fusion** (ABF) [12] is when it is assumed that including more sources does not imply that the extra evidence supports the final conclusion (i.e., dependence between sources is assumed). In ABF, a vacuous opinion contributes the same weight as every other opinion, therefore ABF has no neutral element and is idempotent. An example of this type of situation is when an examination committee tries to grade an student after having observed her dissertation.
  Let ω^A and ω^B be source A and B's respective opinions over the same variable X. Let $\omega_X^{(A \underline{\diamond} B)}$ be the opinion such that [12]:
  $Case\ 1 : u_X^A \neq 0 \vee u_X^B \neq 0 :$

$$\begin{cases} b_X^{(A \underline{\diamond} B)}(x) &= \frac{b_X^A(x) u_X^B + b_X^B(x) u_X^A}{u_X^A + u_X^B} \\ u_X^{(A \underline{\diamond} B)} &= \frac{2 u_X^A u_X^B}{u_X^A + u_X^B} \\ a_X^{A \underline{\diamond} B}(x) &= \frac{a_X^A(x) + a_X^B(x)}{2} \end{cases}$$

  $Case\ 2 : u_X^A = u_X^B = 0 :$

$$\begin{cases} b_X^{(A \underline{\diamond} B)}(x) &= \gamma_X^A b_X^A(x) + \gamma_X^B b_X^B(x) \\ u_X^{(A \underline{\diamond} B)} &= 0 \\ a_X^{A \underline{\diamond} B}(x) &= \gamma_X^A a_X^A(x) + \gamma_X^B a_X^B(x) \end{cases}$$

  where

$$\begin{cases} \gamma_X^A &= \lim_{\substack{u_X^A \to 0 \\ u_X^B \to 0}} \frac{u_X^B}{u_X^A + u_X^B} \\ \gamma_X^B &= \lim_{\substack{u_X^A \to 0 \\ u_X^B \to 0}} \frac{u_X^A}{u_X^A + u_X^B} \end{cases}$$

  Then $\omega_X^{(A \underline{\diamond} B)}$ is called the averaged opinion of ω_X^A and ω_X^B, representing the combination of the dependent opinions of A and B.

- **Weighted Belief Fusion** (WBF) [12] is suitable in cases where the source opinions should be weighted as a function of the confidence of the opinions. When the input opinions have equally confident argument opinions, the fusion is averaging. In case one of the opinions is confident and the other is uncertain, then the confident opinion contributes the highest weight, however the combined confidence does not increase. WBF is commutative, it considers a vacuous opinion as neutral element and is idempotent. An example of this type of situations is when, e.g. medical doctors express opinions about a set of possible diagnoses.
  Let ω^A and ω^B be source A and B's respective opinions over the same variable X. Let $\omega_X^{(A \widehat{\diamond} B)}$ be the opinion such that [12]:

$Case\ 1 : (u_X^A \neq 0 \vee u_X^B \neq 0) \wedge (u_X^A \neq 1 \vee u_X^B \neq 1)$

$$\begin{cases} b_X^{(A\hat{\diamond}B)}(x) = \frac{b_X^A(x)(1-u_X^A)u_X^B + b_X^B(x)(1-u_X^B)u_X^A}{u_X^A+u_X^B-2u_X^Au_X^B} \\ u_X^{(A\hat{\diamond}B)} = \frac{(2-u_X^A-u_X^B)u_X^Au_X^B}{u_X^A+u_X^B-2u_X^Au_X^B} \\ a_X^{A\hat{\diamond}B}(x) = \frac{a_X^A(x)(1-u_X^A)+a_X^B(x)(1-u_X^B)}{2-u_X^A-u_X^B} \end{cases}$$

$Case\ 2 : u_X^A = 0 \wedge u_X^B = 0$

$$\begin{cases} b_X^{(A\hat{\diamond}B)}(x) = \gamma_X^A b_X^A(x) + \gamma_X^B b_X^B(x) \\ u_X^{A\hat{\diamond}B} = 0 \\ a_X^{A\hat{\diamond}B}(x) = \gamma_X^A a_X^A(x) + \gamma_X^B a^B(x) \end{cases}$$

where

$$\begin{cases} \gamma_X^A = \lim_{u_X^A \to 0, u_X^B \to 0} \frac{u_X^B}{u_X^A+u_X^B} \\ \gamma_X^B = \lim_{u_X^A \to 0, u_X^B \to 0} \frac{u_X^A}{u_X^A+u_X^B} \end{cases}$$

$Case\ 3 : u_X^A = 1 \wedge u_X^B = 1$

$$\begin{cases} b_X^{(A\hat{\diamond}B)}(x) = 0 \\ u_X^{(A\hat{\diamond}B)} = 1 \\ a_X^{A\hat{\diamond}B}(x) = \frac{a_X^A(x)+a_X^B(x)}{2} \end{cases}$$

Then $\omega_X^{(A\hat{\diamond}B)}$ is called the weighted fusion opinion of ω_X^A and ω_X^B.

- **Consensus & Compromise Fusion** (CCF) [12] is suitable in cases where the input opinions are in conflict. CCF is designed to maintain shared beliefs from each source, and to transform conflicting beliefs into a compromise belief. If a consensus belief already exists, it is preserved, and compromise belief is formed when necessary. The resulting fused belief is uncertain in the presence of totally conflicting beliefs. CCF is idempotent, commutative and considers a vacuous opinion as the neutral element. This is helpful in situations when different experts generate opinions identifying different options, such as when doctors with different expertise suggest potential causes in a diagnostic process. The fused opinion reflects the opinion of all experts, and illustrates the group as a whole is certain about a certain set of potential causes.
  The Consensus & Compromise Fusion opinion is defined as the result of the following three computation phases: (1) consensus phase, (2) compromise phase, (3) normalization phase.
  Step 1: Consensus step.
  The consensus step simply consists of determining shared belief mass between the two arguments, which is stored as the belief vector b_X^{cons}:

$$b_X^{cons}(x) = min(b_X^A(x), b_X^B(x))$$

The sum of consensus belief denoted b_X^{cons} is expressed as:

$$b_X^{cons} = \sum_{x \in \mathcal{R}(X)} b_X^{cons}(X)$$

The residue belief masses of the arguments are

$$\begin{cases} b_X^{resA}(x) = b_X^A(x) - b_X^{cons}(x) \\ u_X^{resA} = b_X^B(x) - b_X^{cons}(x) \end{cases}$$

Step 2: Compromise step.
The compromise step redistributes conflicting residue belief mass to produce compromise belief mass, stored in b_X^{comp} X expressed by :

$$b_X^{comp} = b_X^{resA}(x)u_X^B + b_X^{resB}(x)u_X^A + \sum_{\{y \cap z = x\}} a_X(y/z)a_X(z/y)b_X^{resA}(y)b_X^{resB}(z) + \sum_{\{y \cup z = x\}\{y \cap z \neq \emptyset\}} (1 - a_X(y/z)a_X(z/y))b_X^{resA}(y)b_X^{resB}(z) + \sum_{\{y \cup z = x\}\{y \cap z = \emptyset\}} b_X^{resA}(y)b_X^{resB}(z)$$

Then the following quantities are computed:
Preliminary uncertainty mass:

$$u_X^{pre} = u_X^A u_X^B$$

Sum of compromise belief:

$$b_X^{comp} = \sum_{x \in \mathcal{P}(X) b_X^{comp}(x)}$$

In general, $b_X^{cons} + b_X^{comp} + u_X^{pre} < 1$, hence normalisation of b_X^{comp} is required:
Normalisation factor:

$$\eta = \frac{1 - b_X^{cons} - u_X^{pre}}{b_X^{comp}}$$

Because belief on X represents uncertainty mass, the fused uncertainty is

$$u_X^{A \diamond B}(x) = u_X^{pre} + \eta b_X^{comp}(X)$$

The compromise belief mass on X must then be set to zero, i.e. $b_X^{comp}(X) = 0$
Step 3: Merging consensus and compromise belief.
After normalisation, the resulting CC-fused belief is

$$b_X^{A \diamond B}(x) = b_X^{cons}(x) + \eta b_X^{comp}(x) \, \forall x \in \mathcal{R}(X)$$

The CC-fused opinion is then expressed as $\omega_X^{A \diamond B} = (b_X^{A \diamond B}, u_X^{A \diamond B}, a_X)$ This marks the end of the three-step process for consensus & compromise fusion.

4.2.2 Trust discount operators

The second operator that is fundamental to quantifying trust is the trust discount. Trust discounting innately is related to the concept of trust transitivity. We explained the concept of trust discounting in more detail as part of section 5.3.4. in [4]. Furthermore, it is critical to note that while [12] presents only a single trust discount operator (unlike the diverse fusion operators discussed earlier), subsequent work by Jøsang in [13] introduces multiple trust discounting operators with distinct properties. The following section elaborates on these operators in detail.

- **Uncertainty Favouring Trust Transitivity [13].** When agent A distrusts recommending agent B, it implies that A believes B doesn't know the truth on X, leading A to also not know the truth on X. This discounting operator is proven to be associative but not commutative, meaning the order of combining opinions matters. In paths with multiple recommending entities, opinion independence is assumed. Eq. 4.1 shows how to calculate the Uncertainty Favouring Trust discounted opinion.

$$\omega_X^{A:B} : \begin{cases} b_X^{A:B} & = b_B^A b_X^B \\ d_X^{A:B} & = b_B^A d_X^B \\ u_X^{A:B} & = 1 - (b_X^{A:B} + d_X^{A:B}) \\ a_X^{A:B} & = a_X^B \end{cases} \tag{4.1}$$

- **Base Rate Sensitive Trust Transitivity [13]** operator does not consider the base rate a_B^A when discounting, which may seem counter intuitive. This operator is equivalent to the trust discount operator from [12]. For example, in a scenario where a stranger seeks a car mechanic in a town known for honesty and asks the first person he meets to direct him to a good car mechanic. Since the stranger does not know this person he will have a fully uncertain trust opinion him ($b = 0, d = 0, u = 1$). However, the base rate will be high since it's known that citizens of this city are honest. Therefore this should impact somehow the discounted result. Eq. 4.2 shows how to calculate the Base Rate Sensitive Trust discounted opinion.

$$\omega_X^{A:B} : \begin{cases} b_X^{A:B} & = E(\omega_B^A) b_X^B \\ d_X^{A:B} & = E(\omega_B^A) d_X^B \\ u_X^{A:B} & = 1 - (b_X^{A:B} + d_X^{A:B}) \\ a_X^{A:B} & = a_X^B, \end{cases} \tag{4.2}$$

  where $E(\omega_B^A) = b_B^A + a_B^A u_B^A$.
  Nonetheless, this approach requires caution. For instance, if the stranger has high trust expectations but receives a recommendation from someone with uncertain beliefs, the resulting belief may seem counter intuitive. This issue could become more pronounced with longer trust paths. Therefore, a safety principle might be to apply base rate-sensitive discounting only to the last transitive link. In summary, the Uncertainty Favouring Trust discounting operator is safe and conservative, while the Base Rate-Sensitive operator can be more intuitive in some situations but requires careful application. Another solution to avoid this problem is to use only the uncertainty favouring operators and set the base rate to $\frac{1}{2}$ assuming that other information that might impact the base rate are already taken into account in the belief and disbelief.

- **Opposite Belief Favouring Trust Transitivity [13].** When agent A distrusts recommending agent B, it suggests A believes B consistently suggests the opposite of B's true opinion about the truth value of X. Consequently, A not only distrusts X to the extent that B suggests belief but also trusts X to the extent that B suggests disbelief because two disbeliefs combine to form belief in this scenario. This operator embodies the idea of "your enemy's enemy is your friend," applicable in certain situations. However, it should only be applied

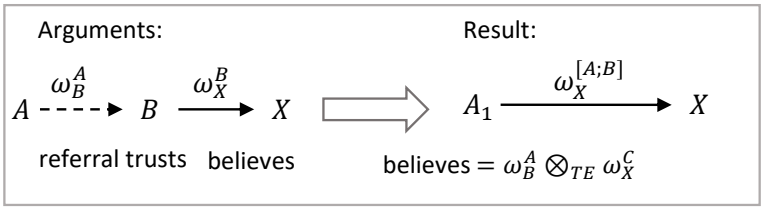


Figure 4.17: Trust discounting for Two-Edge Path [12, Ch.14.3]

when plausible.

$$\omega_C^{A:B} : \begin{cases} b_C^{A:B} & = b_B^A b_C^B + d_B^A d_C^B \\ d_C^{A:B} & = b_B^A d_C^B + d_B^A b_C^B \\ u_C^{A:B} & = 1 - (b_C^{A:B} + d_C^{A:B}) \\ a_C^{A:B} & = a_C^B \end{cases} \tag{4.3}$$

This operator may enable a reputation-based attack where a malicious actor with poor credibility deliberately asserts truthful information. Since the agent expects adversaries to lie, it may incorrectly reject valid claims based solely on the source's negative reputation.

4.2.3 Novel discount operator

During the TLEE development, We have identified that existing discounting operators often encounter challenges in discounting trust across multiple edges, particularly when confronted with referral trust opinions. Concretely, the existing operators for discounting trust that have been previously proposed in [12, 13], provide only solutions for trust discounting when the final edge of the network is always a functional or direct trust. However, based on practical scenarios encountered while working on the use cases from the CONNECT project, we identified the need to discount trust only on referral (or indirect) edges—a limitation that existing discounting operators do not address. In response, we present a novel trust discount operator for referral edges in a path. Our operator is designed to discount two referral edges in a manner that aligns with existing trust discounting principles. This advancement seeks to mitigate inconsistencies encountered when discounting multi-edge referral paths. Beyond discounting trust on only two edges, our new operator enables calculating trustworthiness of paths containing multiple referral edges within complex networks. Furthermore, as part of the evaluation of the novel discounting operator, we also established a relationship between path length and trustworthiness. Our findings highlight that longer referral paths tend to yield less trustworthy outcomes. Additional information regarding the evaluation is available in [14].

Summary of the existing Trust Discounting Operators

In his book [12], Jøsang proposes two trust discount operators for 1) Two-Edge Path and 2) Multi-Edge Path. The process of trust discounting for Two-Edge Path is shown in fig. 4.17. As the name suggests, this trust discounting operator ($\otimes_{TE}$) discounts the opinions on two edges: one referral trust ω_B^A from node A to node B, and one functional trust ω_X^B from node B to variable X.

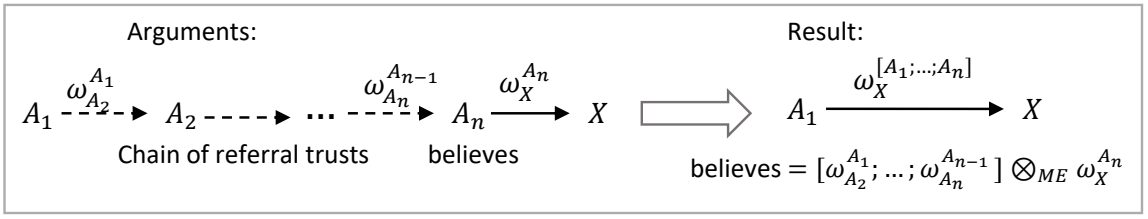


Figure 4.18: Trust Discounting for Multi-Edge Path [12, Ch.14.3]

The Two-Edge Path Discounting Operator ($\otimes_{TE}$) is defined as follows [12]:

$$\omega_X^{[A;B]} = \omega_B^A \otimes_{TE} \omega_X^B : \begin{cases} b_X^{[A;B]} & = P_B^A b_X^B \\ d_X^{[A;B]} & = P_B^A d_X^B \\ u_X^{[A;B]} & = 1 - (b_X^{[A;B]} + d_X^{[A;B]}) \\ a_X^{[A;B]} & = a_X^B \end{cases} \tag{4.4}$$

where P_B^A is the projected probability. The Two-Edge Path discount operator is equivalent to the *Base Rate Sensitive Transitivity* that Jøsang et al. propose in [13], which we explained in more detail in section 4.2.2.

On the other hand, the trust discounting for Multi-Edge Path describes how trust discounting is performed for longer trust paths, in situations when there are more than two adjacent edges, as shown in fig. 4.18. Namely, fig. 4.18 shows a trust path from node A_1 to variable X via an arbitrary number of intermediate nodes $A_2, A_3, \ldots, A_n$.

The derived opinion $\omega_X^{[A_1;\ldots;A_n]}$ on Multi-Edge Paths is defined as follows:

$$\omega_X^{[A_1;\ldots;A_n]} = [\omega_{A_2}^{A_1}; \ldots; \omega_{A_n}^{A_{n-1}}] \otimes_{ME} \omega_X^{A_n} : \tag{4.5}$$
$$\begin{cases} b_X^{[A_1;\ldots;A_n]} & = P_{A_n}^{A_1} b_X^{A_n} \\ d_X^{[A_1;\ldots;A_n]} & = P_{A_n}^{A_1} d_X^{A_n} \\ u_X^{[A_1;\ldots;A_n]} & = 1 - (b_X^{[A_1;\ldots;A_n]} + d_X^{[A_1;\ldots;A_n]}) \\ a_X^{[A_1;\ldots;A_n]} & = a_X^{A_n} \end{cases}$$

where $P_{A_n}^{A_1}$ is the projected probability of the referral trust path $[A_1; \ldots; A_n]$, computed as

$$P_{A_n}^{A_1} = \prod_{i=1}^{n} P_{A_{i+1}}^{A_i}. \tag{4.6}$$

We want to emphasise that the operators for Two-Edge Path Trust Discounting ($\otimes_{TE}$) and Multi-Edge Path Trust Discounting ($\otimes_{ME}$) are two different operators. Hence, we label them with two different symbols, as we also show in table 4.1. Additionally, please note that in order to apply these discount operators, the last edge should always be a functional trust edge.

Problem statement

For demonstration purposes, as an example, we take the Trust Network shown in fig. 4.19. The STN has four referral trust relationships or edges (ω_B^A, ω_D^B, ω_C^A, and ω_D^C) forming a chain, as can

Table 4.1: Trust discount operators

Operator Name	Symbol	Function
Two-Edge Path Eq. eq. (4.4), fig. 4.17	$\otimes_{TE}$	$\omega_B^A \otimes_{TE} \omega_X^B$
Multi-Edge Path Eq. eq. (4.5), fig. 4.18	$\otimes_{ME}$	$[\omega_{A_2}^{A_1}; \ldots ; \omega_{A_n}^{A^{n-1}}] \otimes_{ME} \omega_X^{A_n}$
Referral-Edge Path Eq. eq. (4.7), fig. 4.20	$\otimes_{RE}$	$((\omega_{A_2}^{A_1} \otimes_{RE} \ldots) \otimes_{RE} \omega_{A_{n-1}}^{A_{n-2}}) \otimes_{RE} \omega_{A_n}^{A_{n-1}}$

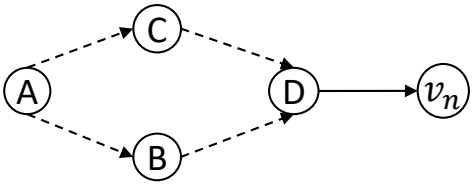


Figure 4.19: Exemplary Trust Model.

be seen in the figure. Additionally, vehicle D has a direct or a functional trust on its velocity v_n. In this example, the final goal is to calculate the opinion $\omega_{v_n}^A$ that for example, a vehicle A has on the velocity v_n even though the vehicle does not observe the velocity directly (i.e., it does not have a direct relationship with the velocity). This is done by first discounting the referral trust opinions ACD and ABD, and then fusing the obtained opinions.

As previously shown in eqs. (4.4) and (4.5), the existing trust discounting operators need the last edge to be a functional trust, which is not the case in ACD and ABD paths, where CD and BD are also referral trust. Although in the previous work Jøsang [12] differentiates and proposes trust discounting operators for Two-Edge and Multi-Edge paths, the case where the graph topology consists *only* of referral chains, has been left out of consideration in his theoretical contributions. To overcome this problem, one might be tempted to use the existing trust discounting operator for Two-Edge Path (Eq. eq. (4.4)) to discount AC and CD. However, this operator was not built for this purpose, primarily because the second opinion, in this case CD is expected to be a functional trust edge. Furthermore, if we want to discount two referral trust edges as part of a chain topology, as shown in fig. 4.19, the solution needs to be consistent with the trust discounting operator for Multi-Edge Path, which already considers discounting on multiple referral edges. Another solution could be to discount using only projected probability since this is the exact way that the Multi-Edge Path uses to discount referral edges (cf. $P_{A_n}^{A_1}$ in Eq. eq. (4.5)). This means that only the projected probabilities (see Eq. eq. (4.6)) are used instead of the opinions; therefore, resulting in a projected probability again. However, in our example in fig. 4.19, the node D requires fusing of the two previously discounted referral paths ACD and ABD, which is only possible when we have opinions (not probabilities).

In this work we aimed to build a trust discounting operator for two or more referral trust opinions (without having the last edge in the STN as a functional trust), since this is needed for some STN topology types, as we showed in fig. 4.19. Moreover, the new trust discounting operator for two referral trust opinions must provide consistent results with the already existing operators that Jøsang introduced for Two-Edge Path and Multi-Edge Path trust discounting.

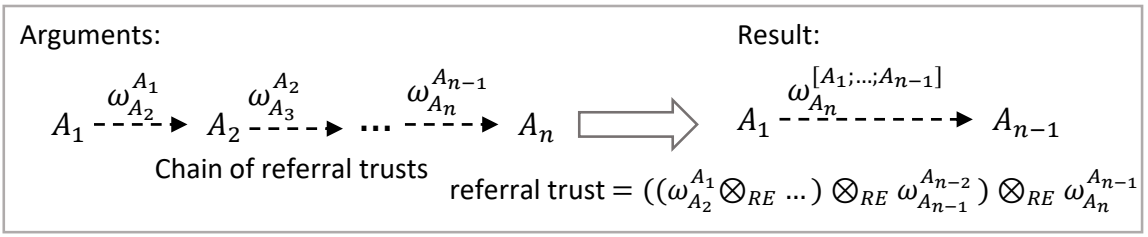


Figure 4.20: Trust Discounting for Referral-Edge Path

The proposed Referral-Edge Path Discounting Operator

As part of this section, we propose a new Referral-Edge Path Discounting Operator ($\otimes_{RE}$) to address the problem previously explained. The idea behind the new Referral-Edge Path Discounting Operator is derived from the principles of the Uncertainty Favouring Operator that was previously proposed in [13]. A notable distinction between our operator (see Eq. eq. (4.7)) and the uncertainty favouring operator is that we focus on ensuring the projected probability of the resulting discounted opinion (e.g.,, $P_C^{[A;B]}$) to be equal to the product of the projected probabilities of the two original opinions prior discounting (e.g.,, $P_C^{[A;B]} = P_B^A P_C^B$).

There are two ways to achieve this, either by adjusting the belief and the disbelief, or the base rate. We decided to adjust the base rate because the symmetry between belief and disbelief implies that altering one necessitates a corresponding adjustment to the other. Furthermore, if we refer to fig. 4.17, when discounting the referral trust edge AB and functional trust edge BX, we can observe that the base rate of the final discounted opinion $\omega_X^{[A;B]}$ is equal to the base rate of the functional trust edge. Therefore, adjusting the base rate of the referral trust will not impact the base rate of the ultimate derived opinion. We define the Referral-Edge Discounting Path Operator ($\otimes_{RE}$) as follows:

$$\omega_C^{[A;B]} = \omega_B^A \otimes_{RE} \omega_C^B : \tag{4.7}$$
$$\begin{cases} b_C^{[A;B]} & = b_B^A b_C^B \\ d_C^{[A;B]} & = b_B^A d_C^B \\ u_C^{[A;B]} & = 1 - (b_C^{[A;B]} + d_C^{[A;B]}) \\ a_C^{[A;B]} & = \frac{(b_B^A + u_B^A a_B^A)(b_C^B + u_C^B a_C^B) - b_B^A b_C^B}{1 - b_B^A (b_C^B + d_C^B)} \end{cases}$$

Our proposed operator is associative, and it works in the same way for discounting two or multiple referral edges (see fig. 4.20) without any inconsistencies.

4.3 TAM/TLEE Interface Explanation: `RunTLEE` Method

Overview. The `RunTLEE method` invokes the core function of the TLEE that performs trust evaluation based on a specified trust model. It can take as input both static and dynamic trust models, which are subsequently analyzed by different modules of the TLEE, as we have explained in more details in the earlier sections of this chapter. The TLEE processes the trust network structures of the Trust Models, which also include relationship data (i.e., trust opinions) to produce the final Actual Trust Level (ATL) for the Trust Model Instance received as input in this function call. In fig. 4.21, we show the method signature.

```
func (tlee *TLEE) RunTLEE(
    trustmodelID string,       // Trust Model ID
    version int,               // Version indicating value changes of the Trust Opinions
    fingerprint uint32,        // Hash of model structure (indicating structural changes in the T
rust Model)
    structure trustmodelstructure.TrustGraphStructure,
    values map[string][]trustmodelstructure.TrustRelationship
) (map[string]subjectivelogic.QueryableOpinion, error) {
    // Function body would go here
}
```

Figure 4.21: `RunTLEE` function call

Parameters.

1. `trustmodelID` (`string`). This parameter is the unique identifier for the Trust Model that the TLEE receives as input. For static trust model instance, it is a permanent identifier, whereas, for dynamic trust model instances it is the template identifier used by the TMM.
2. `version` (`int`). This parameter tracks changes in the numerical values of the opinions of the trust relationships in the in the Trust Model.
3. `fingerprint` (`uint32`). This parameter is the cryptographic hash of the trust model structure, and it changes whenever the graph topology changes. Its purpose is to enables detection of structural modifications in the trust model.
4. `structure` (`trustmodelstructure.TrustGraphStructure`). This parameter holds the structure of the Trust Model, specified as part of the TMM, including: Node relationships, Fusion operators and Discount operators, providing information about how trust should be computed and propagated.
5. `values` (`map[string][]trustmodelstructure.TrustRelationship`). Lastly, this parameter holds the numerical trust opinions for each of the trust relationships in the Trust Model. This values are calculated and updated by the TSM. As previously explained, the numerical values of the trust opinions are used in the very final step of the TLEE, concretely as part of the Evaluator component, when the ATL value is calculated. Until this component, the TLEE operates on a symbolic level, enabling run-time modifications of the numerical values of the opinions.

It is important to emphasize that during the design of the Trust Assessment Framework, the `version` and `fingerprint` parameters were deliberately decoupled, to utilize a specific feature of the TLEE, and optimize the engine's calculation and performance. Namely, the *Expression Synthesizer* is the most computationally expensive component of the whole TLEE. As previously explained, as part of the *Expression Synthesizer*, we build the expression. This expression is only changing (i.e., it needs to be re-computed) when the structure of the Trust Model is changing. As long as the structure of the Trust Model instance remains the same—even in dynamic models where only the numerical values of the trust opinions change—the TLEE will only trigger the Evaluator component. Navigating to the two different types of changes as part of the Trust Model instance, is enabled by both of these parameters.

Return values.

1. `Computed ATLs` (`map[string]subjectivelogic.QueryableOpinion`). Returns the ATLs, one for each sub-trust model, as part of the Trust Model Instance. The return value is a map where keys are the TMs identifiers and the values are their computed trust opinions.
2. `Error` (`error`). Returns any errors that occurred during execution. `Nil` if execution was successful.

TLEE multithreading

The current TAF implementation leverages multithreading by spawning multiple workers as *goroutines*, each responsible for invoking the TLEE with a distinct set of TM instances in parallel. Each worker independently instantiates its own TLEE, allowing multiple TLEE instances to run concurrently—one per worker. This approach ensures that the TLEE itself does not need to be multithreaded, as the concurrency is handled externally by the TAF-level workers. Since TMs may differ in instance, version, and fingerprint, isolating them per worker and per TLEE instance ensures thread safety and simplifies the system design.

4.4 Development Insights: From Implementation to Evaluation

4.4.1 Design and Implementation Evolution

In the first design of the TAF, which is documented in [2], the TLEE and the rest of the TAF components were designed without the central Trust Assessment Manager (TAM) component. In this design, the TLEE was receiving the Trust Models directly from the Trust Model Manager (TMM), and the numerical values of the opinions of the trust relationships, from the Trust Source Manager (TMS), indicating two input interfaces to the TLEE, one with the TMM and one with the TSM. In this design, the numerical trust opinions were kept decoupled from the Trust Model (precisely the trust model structure), identifying the need for some synchronization mechanisms for the structural changes of the network and the numerical changes of the trust opinions. Lastly, as part of the first TAF design, the TLEE had a single output interface. Using this interface, the outcome of the TLEE, i.e., the ATL for the whole Trust Network was sent to the Trust Decision Engine (TDE). With the change of the conceptual design of the Trust Model in deliverable [5], now encompassing not only the structure but also the opinion's numerical values, some of the issues that emerged from the initial TAF design were mitigated. Additionally, as part of this deliverable, we refined the TAF design by introducing the TAM, which, among other roles, now serves as a central communication bus for all TAF components. The solution introduces a singular input and output interface for all TAF components to communicate with the TLEE via the TAM. In alignment with the updated TAF design, we have specified a method with the designated signature of the interface between the TAF and the TLEE, the `RunTLEE` functional call, as shown in fig. 4.21. Consistent with the revised TAF architecture, we have formally defined the TAF-TLEE interface specification through the `RunTLEE` functional call, alongside the concrete signature, as shown in fig. 4.21 and described in section 4.3.

Throughout the project, there were continuously changing requirements, which in response, required continuous refinement in the design, the functional specifications and implementation of the TLEE. These requirements originated from three primary sources: identification of some

limitations in the theory of the SL, modifications to the conceptual design of other TAF components, and use case-driven necessities that mandated TAF adaptations. All of these categories of changes required subsequent modification of the TLEE implementation. One of these changes was the introduction and the implementation of the Sub-Trust Model Extractor. In the initial design of the TAF and the first version of the TLEE implementation, the team was still contemplating on how to consider the various propositions (i.e., variables in the leaves of the trust model), especially within the formal framework of the Subjective Logic. Once we have agreed that we consider these propositions as independent variables, we proceeded with implementing the Sub-Trust Model Extractor. The concrete functional specifications of this component are described in section 4.1. The second change was the implementation of the new trust discount operator for referral paths (Referral-Edge Path discounting), as part of the TLEE. We have provided the formal definition of the new trust discount operator in section 4.2.3, and have modified the TLEE implementation to identify when this operator needs to be applied instead of the discount operator that Jøsang proposes in his book [12]. We have done this by implementing a logic in the TLEE that provides a specific analysis of the structure of the Trust Model (i.e., the trust network), and identifies where the Referral-Edge Path discounting operator is necessary. And lastly, since Jøsang in [12] considers only a single trust discount operator, we have designed the Trust Model, and consequently, specified the TLEE/TAM interface and the TLEE implementation, in accordance with only a single discount operator. The implementation supported configurable trust fusion operators within the Trust Model while employing the standard Trust Discount operator by default. Hence, the most recent change in the TLEE was based on a newly identified requirement from the IMA use case, which required an option to specify different trust discount operators in the trust model.

4.4.2 Evaluation Evolution

In Chapter 8 of deliverable D6.1 [6], we present a comprehensive benchmarking methodology for evaluating the TLEE, along with detailed performance results. Our analysis focuses on the TLEE's capability to process varying levels of complexity inherent in trust models, expression construction, and evidence processing. The benchmarking framework systematically assesses computational efficiency, scalability, and robustness under different workload conditions. Results demonstrate the TLEE's ability to maintain consistent performance while accommodating increasingly complex trust relationships and evidence-based reasoning tasks. These findings validate the engine's design choices and provide measurable insights into its operational boundaries in real-world deployment scenarios.

The performance assessment of the TLEE in Deliverable D6.1 [6] focused exclusively on standalone TLEE functionality, analyzing core computational metrics such as inference latency and model-complexity scalability. In contrast, section 3.9 of this deliverable expands the evaluation scope to the integrated Trust Assessment Framework (TAF), where the TLEE operates as a component within the full system architecture.

Chapter 5

Federated TAF Concepts

In this chapter, we describe four concepts of the federated TAF, which were created within the CONNECT project. In Deliverable D3.2 [5], we developed three approaches of a federated TAF, in which different types of information are exchanged between the individual TAF instances. It was initially envisioned that the first two approaches would be implemented within the context of this work package while the third approach was left to future work. However, none of these approaches were appropriate for the use cases of the CONNECT project, where a federated TAF is used. As a result, a further approach of a federated TAF was created and implemented. This approach of the federated TAF (Case 4) will be evaluated in the context in the context of a use case in D6.2. This chapter provide an overview of all approaches to present the current status of the federated TAF concepts.

The trust assessment query interface of the TAF (see Chapter 3) enables the second concept outlined in this chapter (although limited to the scope of propositions), while the NTM-based federation implementation corresponds to the fourth concept of this chapter.

5.1 Case 1 - Request for an atomic opinion relative to a trust source

In the first case, the federated TAF architecture is leveraged by the requesting TAF A to gain access to the information provided by a remote source of trust evidence, aka trust source, accessible to the petitioned TAF B.

Assumptions Let TAF A be the requesting entity and TAF B be the petitioned entity. We assume that a secure communication channel exists between the containers hosting TAF A and TAF B; furthermore, as per the CONNECT architecture, we assume that TAF A and TAF B have attestation capabilities, and therefore, can produce trust evidence that can be exchanged to mutually assure the trustworthiness of the federated TAFs. This includes evidence about the integrity of the communication channel and authenticity of the sender.

Notation Consider the simple trust model depicted in the right hand side of Figure 5.1, where $\omega_X^B\big|_{\mathrm{TAF}_B}$ denotes the trust opinion of node B (trustor) on node X (trustee), as evaluated in TAF B. The two blocks on the side of the arrow between B and X represent two trust sources producing

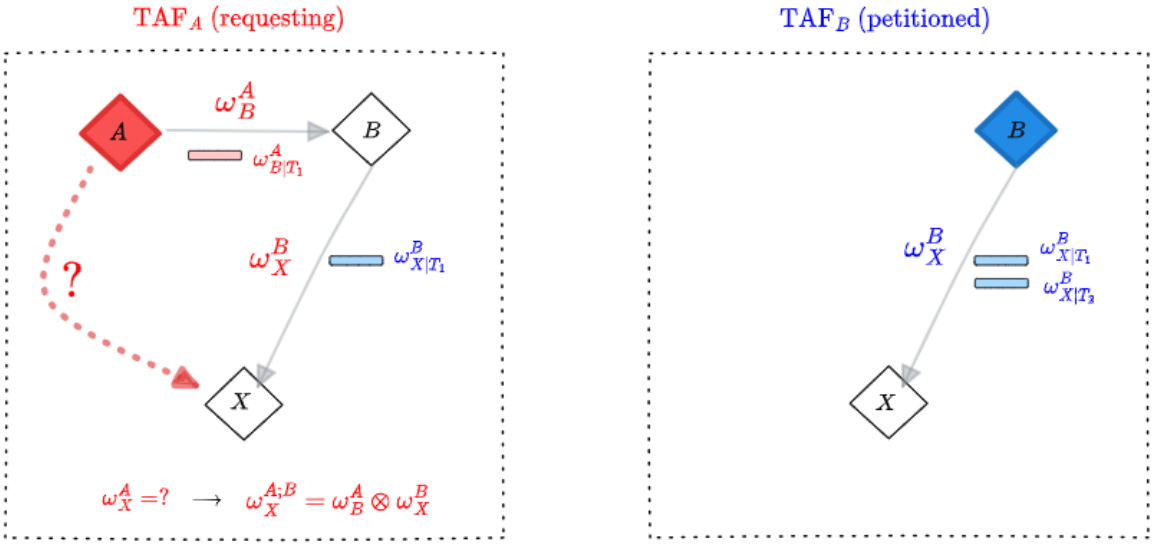


Figure 5.1: Case 1: Representation of the requesting TAF_A (left) and the petitioned TAF_B (right).

evidence relevant to the trust relationship. From these, the Trust Source Manager of TAF B evaluates the atomic opinions $\omega^B_{X|T_1}\big|_{\mathrm{TAF}_B}$ and $\omega^B_{X|T_2}\big|_{\mathrm{TAF}_B}$. The trust opinion $\omega^B_X\big|_{\mathrm{TAF}_B}$ is evaluated by TAF_B by taking both atomic trust opinions into account. Notice that if only one atomic opinion $\omega^B_{X|T_1}\big|_{\mathrm{TAF}_B}$ is available, then $\omega^B_X\big|_{\mathrm{TAF}_B} = \omega^B_{X|T_1}\big|_{\mathrm{TAF}_B}$.

Federation Refer to Figure 5.1, where trust opinions and atomic opinions evaluated in the requesting TAF_A are depicted in red and trust opinions and atomic opinions evaluated in the petitioned TAF_B are depicted in blue. TAF_A needs to complete an assessment which requires to evaluate $\omega^A_X\big|_{\mathrm{TAF}_A}$, but since it does not have access to any relevant trust source, it cannot evaluate it directly. However, TAF_A knows that an atomic opinion $\omega^B_{X|T_1}\big|_{\mathrm{TAF}_B}$ is available in the TAF_B. The TAF_A model hence triggers a federation request addressed to TAF_B, requesting $\omega^B_{X|T_1}\big|_{\mathrm{TAF}_B}$. As visible in Figure 5.1, TAF_A does not know of the existence of the atomic opinion $\omega^B_{X|T_2}\big|_{\mathrm{TAF}_B}$ at TAF_B.

TAF_B responds with the required atomic opinion $\omega^B_{X|T_1}\big|_{\mathrm{TAF}_B}$, which TAF_A uses to evaluate $\omega^B_X\big|_{\mathrm{TAF}_A} = \omega^B_{X|T_1}\big|_{\mathrm{TAF}_B}$. Moreover, TAF_B provides its attestation evidence to TAF_A, so that TAF_A has the atomic opinion $\omega^A_{B|T_1}\big|_{\mathrm{TAF}_A}$. As a consequence, TAF_A can evaluate $\omega^A_B\big|_{\mathrm{TAF}_A} = \omega^A_{B|T_1}\big|_{\mathrm{TAF}_A}$. Finally, TAF_A can combine $\omega^B_X\big|_{\mathrm{TAF}_A}$ and $\omega^A_B\big|_{\mathrm{TAF}_A}$ to provide $\omega^A_X\big|_{\mathrm{TAF}_A} = \omega^{A;B}_X\big|_{\mathrm{TAF}_A}$ using trust discounting.

5.2 Case 2 - Request for a trust opinion for a single trust relationship

In this section we consider the case where the federated TAF architecture is leveraged by the requesting TAF_A not simply for querying an atomic opinion relative to an unaccessible trust

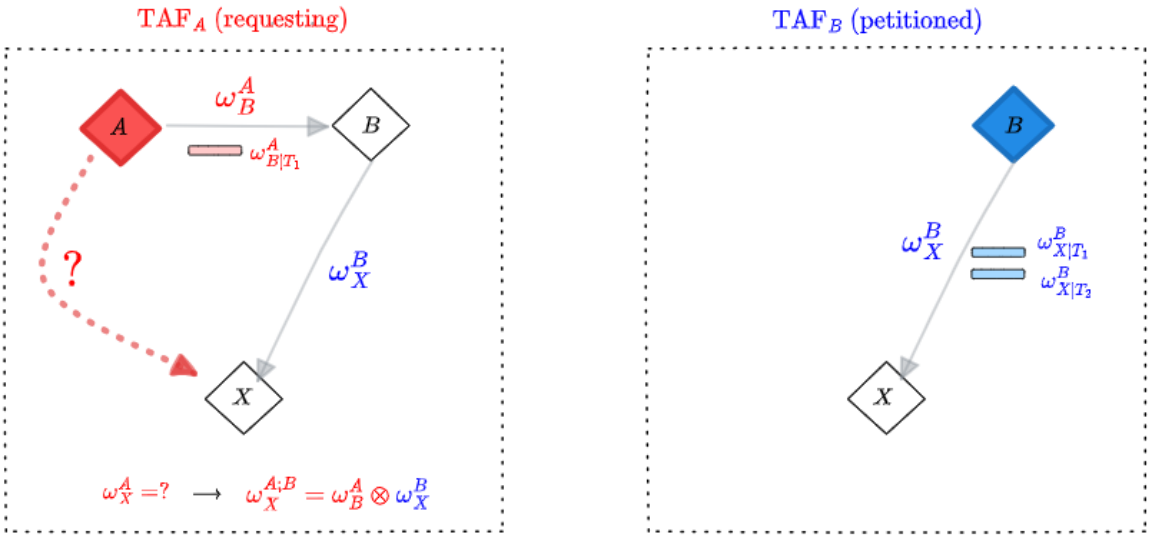


Figure 5.2: Case 2: Representation of the requesting TAF_A (left) and of the petitioned TAF_B (right).

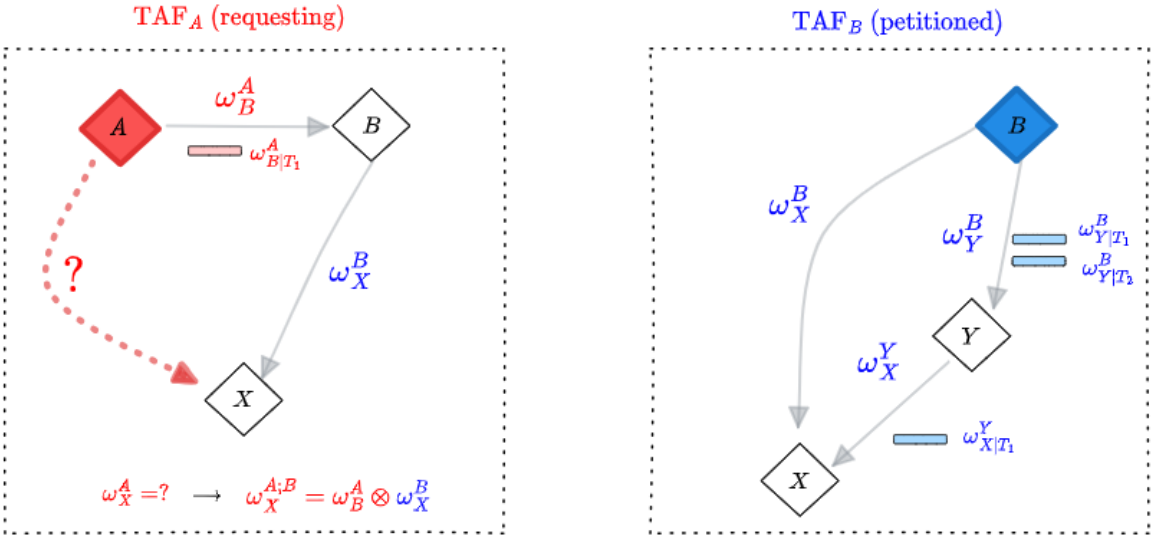


Figure 5.3: Case 2: Representation of the requesting TAF_A (left) and of the petitioned TAF_B (right).

source, but to require the result of a trust opinion computation performed by the petitioned TAF_B. This case is very similar to case 1, just the way how the opinion is created differs.

Assumptions We consider the same assumptions as in the previous case. As before, TAF_A is the requesting entity TAF_B is the petitioned entity.

Federation The TAF_A and TAF_B are depicted in Figure 5.2. As in Case 1, the TAF_A model needs to assess the opinion $\omega_X^A\big|_{\mathrm{TAF}_A}$, and it cannot evaluate it directly since it has not access to any relevant trust source. This time, TAF_A knows that TAF_B is able to assess the trust opinion $\omega_X^B\big|_{\mathrm{TAF}_B}$. Then TAF_A initiates the federation process,requesting $\omega_X^B\big|_{\mathrm{TAF}_B}$.

TAF_B receives the federation request. Notice that, by the TAF_B perspective, this is in principle equivalent to receiving a TAR for $\omega_X^B\big|_{\mathrm{TAF}_B}$ by an application. TAF_B responds with the required $\omega_X^B\big|_{\mathrm{TAF}_B}$, which it evaluates as fusion of the trust sources $\omega_{X|T_1}^B\big|_{\mathrm{TAF}_B}$ and $\omega_{X|T_2}^B\big|_{\mathrm{TAF}_B}$. Furthermore, TAF_B sends its attestation evidence, which constitute a trust source for TAF_A, which evaluates the atomic opinion $\omega_{B|T_1}^A\big|_{\mathrm{TAF}_A}$.

Finally, TAF_A is able to discount the received opinion $\omega_X^B\big|_{\mathrm{TAF}_B}$ by $\omega_B^A\big|_{\mathrm{TAF}_A}$, and to evaluate the required $\omega_X^A\big|_{\mathrm{TAF}_A} = \omega_X^{A;B}\big|_{\mathrm{TAF}_A}$.

Federation (variation of TAF_B) To better illustrate this case we consider the federation process in the same setting, with the variation of the TAF_B depicted in Figure 5.3. As before TAF_A is initiating the federation to require $\omega_X^B\big|_{\mathrm{TAF}_B}$.This time TAF_B evaluates $\omega_X^B\big|_{\mathrm{TAF}_B}$=$\omega_X^{B;Y}\big|_{\mathrm{TAF}_B}$, as it would do in receiving a TAR for $\omega_X^B\big|_{\mathrm{TAF}_B}$.

Trust model design requirements For the above process to take place the following requirements need to be fulfilled:

- The requesting TAF_A needs to know that the petitioned TAF_B is able to evaluate $\omega_X^B\big|_{\mathrm{TAF}_B}$;
- The requesting TAF_A and the petitioned TAF_B reference trust opinion $\omega_X^B\big|_{\mathrm{TAF}_B}$ in a unique way.

The previous requirements are satisfied if TAF_A and TAF_B share the notation and the semantics of the elements in Figure 5.2. Notice that the requesting TAF_A explicitly encodes the fact that $\omega_X^B\big|_{\mathrm{TAF}_B}$ is evaluated by TAF_B. Notice that this implies that TAF_A does not need to know how $\omega_X^B\big|_{\mathrm{TAF}_B}$ is evaluated by TAF_B, which is an advantage towards computation distribution across TAFs.

5.3 Case 3 - Distributed calculation of a trust opinion

In this section, we finally consider a scenario where the federated TAF architecture is leveraged to assess the same trust object, by TAF_A and by TAF_B. As a motivating example, imagine vehicle

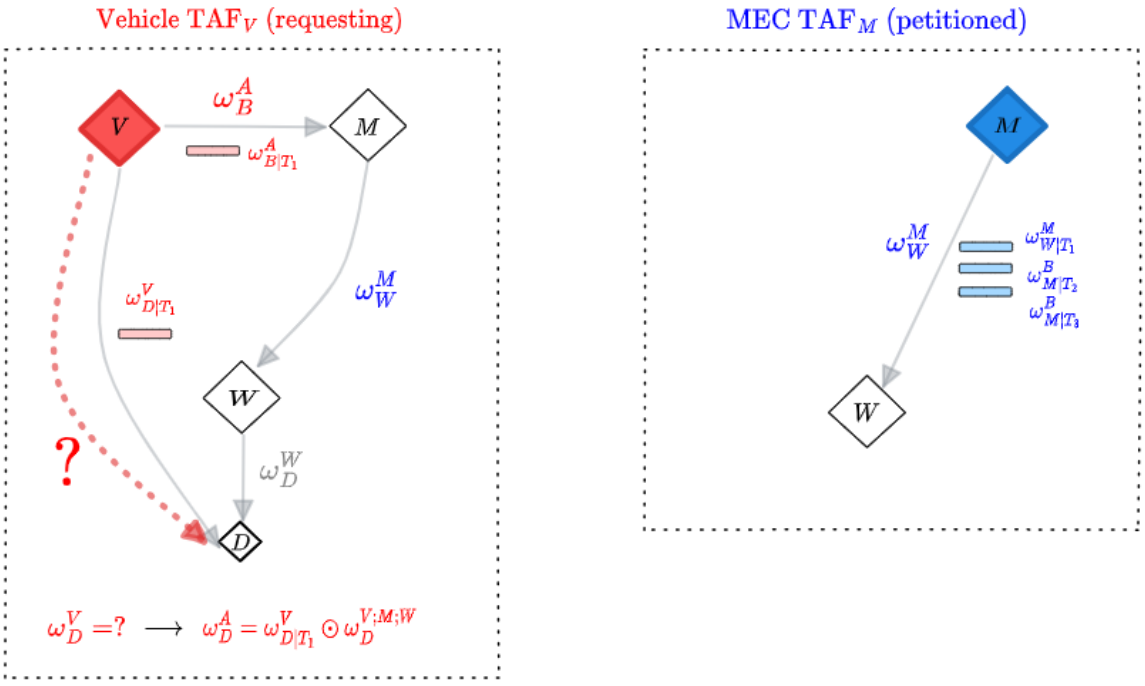


Figure 5.4: Case 2 Example: MEC-based V2X Node Trustworthiness Assessment Service

V and vehicle Y, in the same neighborhood, both receiving V2X data D from vehicle W. Both V and Y are interested in assessing the trustworthiness of the data D, as explained for the Vehicle TAF in Figure 5.4.

Both V and Y run the onboard misbehaviour detection service, and can produce trust evidence both relevant to the V2X data D (data-centric misbehaviour detection) and to the vehicle as source of V2X data W (node-centric misbehaviour detection). Both TAF_V and TAF_Y receive TARs to evaluate the ATLs $\omega_D^V\big|_{\mathrm{TAF}_V}$ and $\omega_D^Y\big|_{\mathrm{TAF}_Y}$, as detailed in Figure 5.5.

The interest in federating TAF_V and TAF_Y resides, in this case, in making the whole evidence available at both sides, which is beneficial because it should provide a reduction on the uncertainty on both ATLs. The most straightforward way to achieve this is by implementing federation instances so that all the atomic opinions from the trust sources can be shared. This is shown in Figure 5.6.

Sharing all the available atomic opinions would require several federation requests to take place (one for each shared atomic opinion). This has the drawback both of provoking a communication overhead (due to the multiple federation processes) and a potential increase in the complexity of both TAF_V and TAF_Y, which need to embed information about all the available trust sources. For these reasons, the federation solution that appears most favorable is the one depicted in Figure 5.7. In this case, only two federation processes take place (one from TAF_V towards TAF_Y, and one from TAF_Y towards TAF_V). Notice that the final ATLs $\omega_D^V\big|_{\mathrm{TAF}_V}$ and $\omega_D^Y\big|_{\mathrm{TAF}_Y}$ will have the same value only if both $\omega_Y^V\big|_{\mathrm{TAF}_V}$ and $\omega_V^Y\big|_{\mathrm{TAF}_Y}$ correspond to full belief, zero uncertainty opinions. In this case, TAF_V and TAF_Y may be understood to work on the same global trust model; and the federation principle allows a distributed evaluation of the required ATL, where half of the computation is performed by TAF_V and the other half by TAF_Y.

Note that there can be cases where evaluation of TAF_V evaluates back to TAF_Y and this might lead to (prohibited) circular evaluations that are not identifiable in any single trust model. Therefore, we only allow forward and no backward references, i.e., if TAF_V references TAF_Y then if

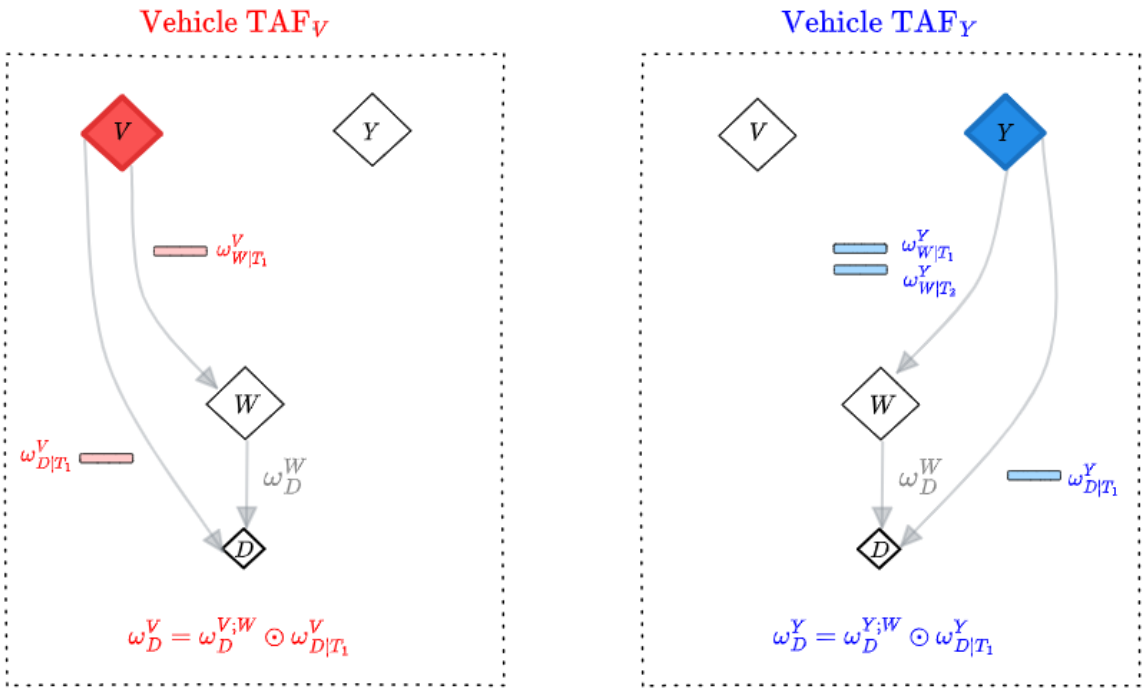


Figure 5.5: Example: two vehicles. Standalone case.

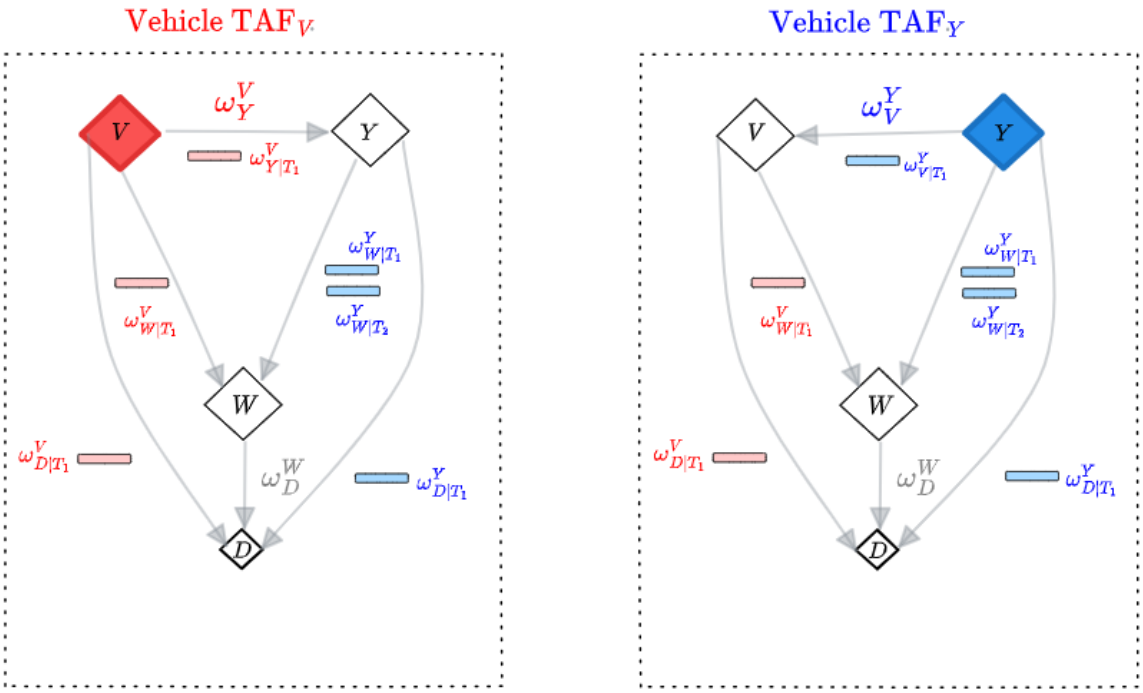


Figure 5.6: Example: two vehicles. Federation, sharing of the atomic opinions from the trust sources.

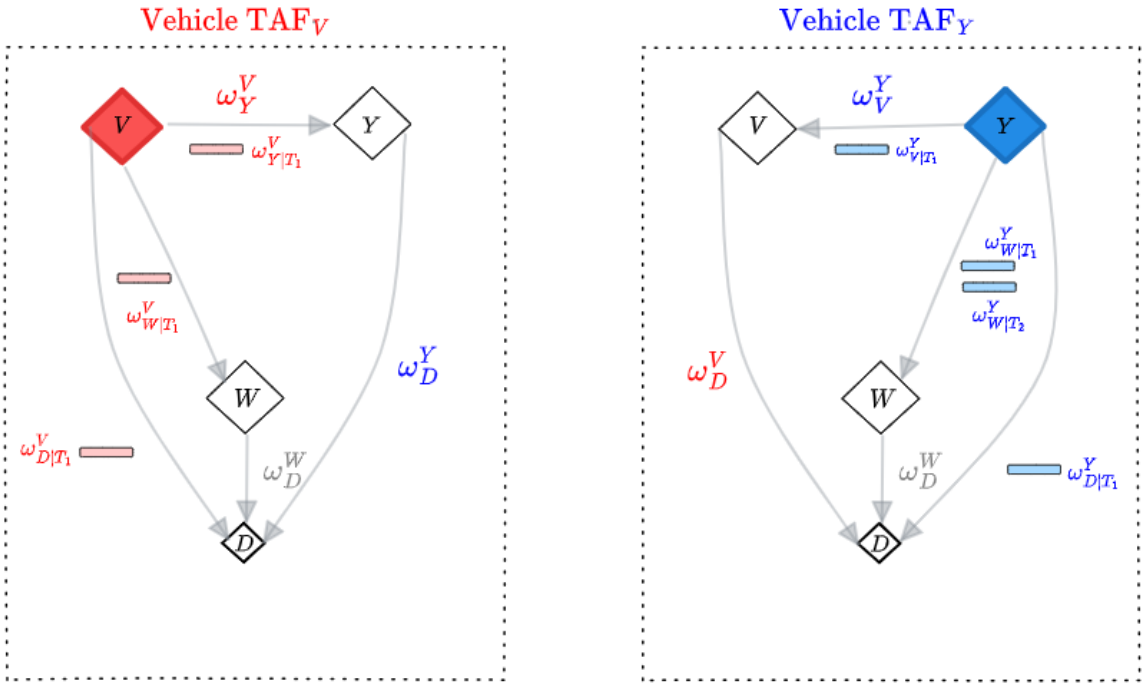


Figure 5.7: Example: two vehicles. Federation, sharing of trust opinions.

evaluation of a trust object in TAF_Y encounters a link back to TAF_V, the evaluation would be aborted and fail. A more complex algorithm to avoid cycles can be envisioned.

5.4 Case 4 - Pushing actual trustworthiness level (ATL) updates

Next up, we cover a more specific variant of the second case. This case was created as an additional approach of the federated TAF beyond what was included in Deliverable D3.2 and was implemented in the context of the CONNECT project. Here, we only focus on the actual trustworthiness levels as the only trust relationships shared between federated nodes. Furthermore, this variant uses a push-based communication approach instead of a pull-based approach.

Assumptions Again, as in the second case, TAF_A is the receiving entity and TAF_B is the sending entity. TAF_A is considering the trustworthiness of certain entities in its trust models and requires TAF_B to provide their derived opinions (ATLs) on them, without knowing internal details of TAF_B's trust models.

Federation TAF_B hosts a trust model in which certain ATLs might be of interest for other TAF instances. TAF_B maintains a mutlicast group of other TAFs to feed ATL update to. After TAF_A has registered at TAF_B, TAF_B sends change notifications to TAF_A once the ATLs have changed. By limiting the queryable entities to ATL values (leaf nodes), the internals of the used trust models used by the providing TAF_B stay hidden.

Trust model design requirements The outlined approach are based on the following requirements:

- The subscribing TAF_A needs to know that TAF_B is able to provide relevant ATLs (irrespective of the specific trust models);
- The subscribing TAF_A is using a trust model that relies on node trustworthiness messages (NTMs) as a trust source, hence TAF_B becomes a trust source for TAF_A when sending trust notifications.

Chapter 6

Trust Assessment Framework inside a TEE

6.1 Motivation

The CONNECT TAF framework introduces trust-aware decision-making for managing CCAM functions throughout their lifecycle. For instance, if a safety-critical CCAM function is deployed on an ECU that begins to exhibit signs of faulty behavior, the ATL value associated with that ECU is expected to decrease significantly. This drop would then trigger the enforcement of an appropriate security control strategy—such as migrating the function to another ECU within the vehicle. From this example, it becomes obvious that it is of paramount importance to provide the necessary safeguards when it comes to the TAF deployment. In fact, this is also reflected in the trust requirement FR_TR_5 documented in D2.2 [7] by which the deployed TAF instance needs to operate under resilience and robustness assurances.

To this extent, it becomes obvious that the CONNECT TAF is a critical service that participates in continuous monitoring and enforcement of security controls and policies in the system. Hence, it needs to be treated as a Trusted Component within the in-vehicle architecture and, as such, it needs to be deployed with the necessary assurances attesting to the correctness of its state during runtime. This opens the investigation of placing the entire TAF functionality within a Trusted Execution Environment as illustrated in the overarching CONNECT Architecture [7].

In the first release of the overarching CONNECT framework, focus is put on the evaluation of the TAF service as a standalone element outside of any trusted execution environment. From this evaluation, it is clear that the 60-70 milliseconds required to derive a trust decision showcase the potential to support the safety-critical CCAM applications presented in [6] and the defined runtime barrier of 200 milliseconds.

As we progress toward the second release of the CONNECT framework, a key question arises concerning the overhead introduced by the Trusted Execution Environment and its impact on the overall runtime performance of the TAF. In the following section, we conduct an initial evaluation of the performance overhead associated with different types of TEEs, using the *Gramine LibOS* runtime as documented in [8]. Specifically, this process of ”graminizing” the TAF application involves the specification of the Gramine Manifest ile that describes all the software stack (including the TAF application) and configuration that needs to be executed inside the enclave. To do so, we define a trust model template designed to incrementally stress the TAF’s computational logic. Based on this model, we assess the additional runtime cost introduced by three TEE configurations: (1) a software-based TEE utilizing Gramine Direct capabilities, (2) a hardware-backed

TEE by integrating the graminized TAF application with an Intel SGX-enabled host system that provides a full hardware root of trust, and (3) a hardware-based TEE through the deployment of the TAF service inside a TDX Trusted VM.

6.2 Experimental Setup

Going beyond the microbenchmarking in the context of the IMA use case [6], for this evaluation we want to define a scenario that will gradually increase the complexity with respect to the internal operations that need to be performed within the TAF component. Consequently, in this section we consider a simple trust model deployed on a TAF agent running on a MEC infrastructure (or a Road-Side Unit).

In this context, the TAF agent sequentially receives a new trustworthiness claim from a vehicle in its vicinity. According to this scenario, each vehicle transmits a single set of evidence and then stops interacting with the TAF. This triggers the TAF to instantiate a new trust model instance to keep track of the trustworthiness of both the vehicle and the data that it transmits to the MEC. The trust model instance describing the relationships for a single vehicle V_1 is presented in Figure 6.1. For the entire lifespan of the experiment, all the trust model instances are maintained; the Time-To-Live parameter for the trust model instances is intentionally set to high number (i.e., 1 hour) so that no vehicle is considered detached and all trust model instances are persisted inside the TAF instance.

The received trustworthiness claims used to quantify the relevant trust opinions are slightly different for the purposes of this evaluation (Table 6.1. For the trust opinion of the analyst node A on the integrity of the vehicle V_1, $\omega^A_{V_1}$, attestation evidence is collected proving the correct configuration of the OBU system (i.e., running the correct software stack). For the trust opinion on the integrity of the data, $\omega^A_{C_1}$, relevant data traces are used. The extracted evidence claims from the data traces show indications that the transmitted data are sent successfully without any cyclic redundancy check (CRC) errors. In addition, the data traces include evidence pertaining to the resilience of the communication channel. To provide more evidence on the runtime execution of the in-vehicle network, traces regarding the isolation of the device applications are provided. Finally, a data trace includes a network telemetry metric, namely the round-trip time corresponding to the transmission of the data. Of course, as applied in the evaluations of D6.1 [6], the internal trust opinion, $\omega^{V_1}_{C_1}$, is assumed to have a trust opinion of full belief.

Concerning the execution of the scenario, we are running the TAF agent on a host machine with a single Intel® Xeon® processor D-1736NT CPU @ 2.70GHz with 8 cores and 256 GB RAM. To simulate the transmission of trustworthiness claims from different vehicle nodes each time, we have developed a simple script that initially configures the TAF agent (i.e., using a TAS_INIT_REQUEST to configure the trust model template and a TAS_SUBSCRIBE_REQUEST to subscribe for new trust assessment reports). Then, with a frequency of one second, the script sends a single trustworthiness claim each time representing a different vehicle identifier. To further stress the TAF computation logic, the trust model is developed so that upon receiving a new message, the new trust model instance is created, and then the TLEE is invoked to recalculate the ATL value for all the trust model instances that are available in memory. The script is executed until there are 100 vehicles simulated (and thus 100 trust model instances maintained inside the Trust Model Manager component).

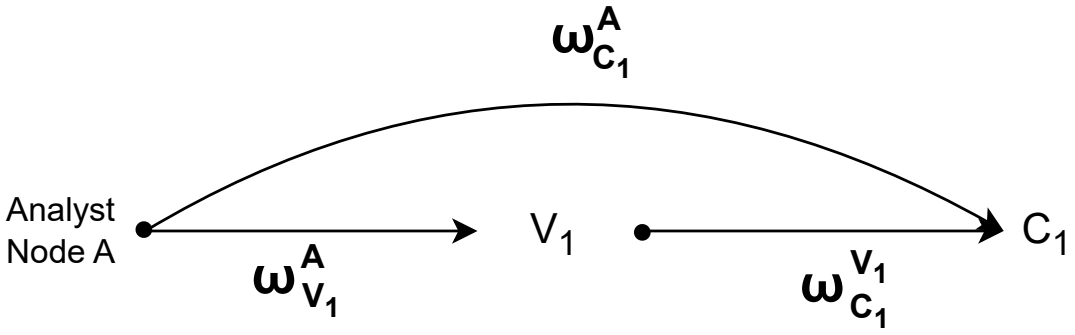


Figure 6.1: Simplified trust model for experimentation in different TEE environments

Opinion	Default $\omega(\mathbf{b}, \mathbf{d}, \mathbf{u})$	Evidence	Trust Property
$\omega^A_{V_1}$	$(0, 0, 1)$	Binary Integrity Measurement List	Software Stack Configuration Integrity
$\omega^A_{C_1}$	$(0, 0, 1)$	CRC Checks/Sequence numbers	Communication Integrity
		Netflow headers	Communication Resilience
		Isolation level	Source Integrity
		Round-Trip time	Communication Availability

Table 6.1: Trust model information for organization A: Trustworthiness evidence and trust quantification weights.

6.2.1 Baseline TAF execution

To seamlessly evaluate the TEE overhead in the TAF runtime execution, we containerize the TAF executable and manage it through the Docker runtime environment. Listing 6.1 presents the docker-compose.yml configuration file that is used to deploy the TAF as a docker service (lines 1-10) and the necessary Kafka (lines 12-39) and Zookeeper (lines 41-57) services to interact with the TAF service. By running the "*docker compose up -d*" command, all the services are spawned and we are ready to run the evaluation script and collect our measurements. The evaluation is executed with three concrete scenarios: i) all 100 vehicles report positive data, ii) 30% of the vehicles report failed CIV attestation appraisals, and iii) 70% of the vehicles report failed CIV attestation appraisals. With this we want to capture the runtime overhead that the trust quantification function may infer to the overall runtime assessment. This is due to the fact that upon receiving a failed attestation, the rest of the quantification function logic is omitted and the trust opinion is immediately set to full disbelief.

```
1  taf:
2    image: ubi/taf:1.0-perf-level-of-isolation
3    container_name: mec_taf
4    entrypoint: ["sh", "-c", "TAF_CONFIG=/app/res/taf.json /app/
       out/main"]
5    volumes:
6      - ./taf/main:/app/out/main:ro
7      - ./taf/taf.json:/app/res/taf.json:ro
8    depends_on:
9      - kafka
10   network_mode: host
```

```
11
12  kafka:
13    image: confluentinc/cp-kafka:7.6.0
14    container_name: "kafka"
15    hostname: kafka
16    environment:
17      KAFKA_BROKER_ID: 1
18      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
19      KAFKA_LISTENERS: DOCKER_INTERNAL://0.0.0.0:9092, LOCALHOST
           ://0.0.0.0:9091, EXTERNAL://0.0.0.0:9093
20      KAFKA_ADVERTISED_LISTENERS: DOCKER_INTERNAL://kafka:9092,
           LOCALHOST://127.0.0.1:9091, EXTERNAL://192.168.2.14:9093
21      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: DOCKER_INTERNAL:
           PLAINTEXT,LOCALHOST:PLAINTEXT, EXTERNAL:PLAINTEXT
22      KAFKA_INTER_BROKER_LISTENER_NAME: DOCKER_INTERNAL
23      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
24      KAFKA_LOG_RETENTION_HOURS: 48
25      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
26      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
27      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
28    ports:
29      - 9093:9093
30      - 9091:9091
31    volumes:
32      - kafka:/var/lib/kafka/data
33    depends_on:
34      - zookeeper
35    logging:
36      options:
37        max-size: "10m"
38        max-file: "10"
39    restart: unless-stopped
40
41  zookeeper:
42    image: confluentinc/cp-zookeeper:7.6.0
43    container_name: "zookeeper"
44    hostname: zookeeper
45    environment:
46      ZOOKEEPER_CLIENT_PORT: 2181
47      ZOOKEEPER_TICK_TIME: 2000
48    ports:
49      - 2181:2181
50    volumes:
51      - zookeeper-data:/var/lib/zookeeper/data
52      - zookeeper-logs:/var/lib/zookeeper/log
53    logging:
54      options:
55        max-size: "10m"
```

```
56          max-file: "10"
57      restart: unless-stopped
58
59  volumes:
60    kafka:
61      name: demo_kafka
62    zookeeper-data:
63      name: demo_zookeeper-data
64    zookeeper-logs:
65      name: demo_zookeeper-logs
```

Listing 6.1: Dockerfile for experimentation

6.2.2 TAF as a Gramine application (Sw- and Hw-based TEE)

When transitioning to a software-based TEE, there is additional pre-processing that needs to take place prior to the TAF service deployment. Specifically, in the context of enclave applications using Gramine LibOS, it is essential that the enclave Manifest is specified. The Manifest is basically an application-specific configuration text file that specifies the environment and resources for running an application inside Gramine. The Manifest template file that is used for describing the TAF enclave is presented in Listing 6.2. Once specified, this Manifest template is subjected to the Sealing process, where it is being signed by a Sealing Authority (e.g., the Security Administrator operating the CONNECT Cloud infrastructure) to derive the final Manifest file. This signed proof of the configuration specifics of the TAF enables the secure launch of the graminized application; i.e., the TAF cannot secure launch as a service without the proper configuration files and packages.

By inspecting Listing 6.2, we can observe that in the manifest template we specify the path to the expected TAF configuration file (lines 8, 17) and the URI for the actual TAF executable file (line 16). Once subjected to the sealing process, the hash of each file is computed, meaning that they cannot be tampered with prior to their execution inside the enclave. In addition, inside the manifest we can specify performance-related settings such as the stack size of each thread in each enclave (line 20; set to 2MB) and the entire memory required for the TAF enclave to operate (line 25; set to 2GB).

```
1  # Copyright (C) 2023 Gramine contributors
2  # SPDX-License-Identifier: BSD-3-Clause
3
4  libos.entrypoint = "/app/out/main"
5  loader.log_level = "{{ log_level }}"
6  loader.entrypoint = "file:{{ gramine.libos }}"
7  loader.env.LD_LIBRARY_PATH = "/lib:/lib:{{ arch_libdir }}:/usr/{{
       arch_libdir }}"
8  loader.env.TAF_CONFIG="/app/res/taf.json"
9
10 # loader.argv = ["/app/out/main"]
11
12 fs.mounts = [
13   { path = "/lib", uri = "file:{{ gramine.runtimedir() }}" },
```

```
  { path = "/etc/hosts", uri = "file:/etc/hosts" },
  { path = "/app/res", uri = "file:/app/res" },
  { path = "/app/out/main", uri = "file:/app/out/main" },
  { path = "/app/res/taf.json", uri = "file:/app/res/taf.json" },
]

sys.stack.size = "2M"
sys.enable_extra_runtime_domain_names_conf = true

sgx.debug = true
sgx.edmm_enable = 'false'
sgx.enclave_size = '2G'
sgx.max_threads = '32'

sgx.trusted_files = [
  "file:{{ gramine.libos }}",
  "file:{{ gramine.runtimedir() }}/",
  "file:/etc/hosts",
  "file:/app/out/main",
  "file:/app/res/taf.json"
]

sgx.allowed_files = [
  "file:/app/res/cert/tchkeys_public_key.pem",
]
```

Listing 6.2: Gramine Manifest Template to generate TAF as an enclavized application

As already mentioned, from the sealing process the output is a set of files describing all the integrity checks for the expected configuration characteristics of the TAF graminized application and the authenticity check showing that this manifest is signed by the expected signing authority. In order to run the TEE-backed TAF service using the Docker runtime, we need to update the command that launches the TAF process (i.e., entrypoint command) and pass the sealed manifest files to the container environment. Listing 6.3 lists the updated description of the TAF service in the docker-compose.yml deployment configuration file. Compared to the plain TAF service, we can see that in order for the enclavized application to work inside the containerized environment, we need to expose the host the SGX drivers through the devices in lines 5-6 of Listing 6.3. In addition to that, lines 7-8 provide the necessary custom profile to run gramine-direct applications inside the docker containers, useful for evaluating the software-based TEE flavor of the TAF service. Finally, a key differentiator between the two TEE-based TAF instances is the entrypoint command. For the software-based TEE version, the TAF service is deployed using the entrypoint in line 9, while for the full-fledged, SGX-rooted graminized application the TAF is deployed by replacing the entrypoint with the commented line 10 (Listing 6.3).

```
  taf:
    image: ubi/taf:1.0-perf-level-of-isolation
    container_name: mec_taf
    devices:
      - "/dev/sgx_enclave:/dev/sgx_enclave"
      - "/dev/sgx_provision:/dev/sgx_provision"
```

```
    security_opt:
      - seccomp=./docker.json
    entrypoint: ["sh", "-c", gramine-direct taf] # For Sw-based
       TEE
    #entrypoint: ["sh", "-c", gramine-sgx taf] # For Hw-based TEE
    volumes:
      - ./taf/taf.manifest.sgx:/app/taf.manifest.sgx:ro
      - ./taf/taf.sig:/app/taf.sig:ro
      - ./taf/taf.manifest:/app/taf.manifest:ro
      - ./taf/main:/app/out/main:ro
      - ./taf/taf.json:/app/res/taf.json:ro
      - ./taf/hosts:/etc/hosts:ro
    depends_on:
      - kafka
    network_mode: host

# The remainder of the file remains as is
```

Listing 6.3: Dockerfile for experimentation (TAF as a graminized application)

6.2.3 TAF inside a Trusted VM (TDX)

For the final evaluation phase, we have launched an Ubuntu-based virtual machine operating within a Trusted Domain Extensions (TDX) environment. The VM is configured as a trusted execution environment (TEE), inside its own Trust Domain (TD). This provides the necessary confidentiality and integrity of the application and its data, even in the presence of a potentially compromised host or hypervisor.

Once the Trusted VM is configured and deployed, we deploy the TAF agent as a typical go application, as we do in the baseline scenario. For the instantiation of the TDX VM, we did not specify custom resource parameters; instead, we relied on the default CPU and memory configurations provided by QEMU during VM launch. This setup demonstrates a minimal and reproducible deployment approach suitable for evaluating TDX-based security guarantees.

6.3 Performance Evaluation and Take-away messages

As mentioned in the experimentation setup, the scenario is executed three consecutive times:

- **All positive traces**: All 100 vehicles report positive traces about the vehicle's correct state and the transmitted data.
- **30% failing vehicles**: 30 out of the 100 vehicles report negative traces about the vehicle's correct state due to an invalid CIV reports.
- **70% failing vehicles**: 70 out of the 100 vehicles report negative traces about the vehicle's correct state due to an invalid CIV report.

The following figures showcase the TEE overhead on the TAF execution considering the aforementioned scenarios. By comparing Figures 6.2 - 6.4, we quickly observe that the actual content

of the trustworthiness claims do not cause any impact on the TAF runtime. In fact, this is an expected behavior considering the simple operations selected in the trust quantification functions allowing the mapping of concrete evidence to subjective logic opinions in constant time (see Chapter 9). That being said, we will focus on Figure 6.2 for the interpretation of the overhead posed by a TEE environment.

From the plot of Figure 6.2, it is clear that the SGX-based poses a non-negligible overhead on the overall TAF execution. Recall that the monitored execution refers to the duration of the TAF to process the incoming trustworthiness claim, quantify the corresponding trust opinions stemming from the evidence and eventually re-calculate the ATL values for all trust model instances that are persisted in-memory. One critical think to notice is that only in the case of the SGX-enabled enclave, the increase in the in-memory storage requirements (i.e., due to the need of storing more and more trust models) leads to an gradual increase in the overhead that the TEE poses in the overall TAF runtime performance.

Based on the previous observation, Table 6.2 provides additional insight on the additional overhead posed by the two TEE environments. Specifically, it is clear that the average overhead identified in this scenario is slightly below 40%. At the same time, regardless of the additional computational overhead (i.e., increase in the number of vehicles and trust models that need to be maintained) the additional overhead posed by the software-based TEE version has a standard deviation of 9.31%, indicating that the overhead is small and generally below 10%.

Finally, one key takeaway message is related to the runtime performance of the TAF service when executed inside a Trusted VM deployed in a TDX-capable machine. Of course, having executed the TAF instance as a typical GoLang executable - i.e., not managed as a docker container - on a completely different machine does not allow the direct comparison with the aforementioned measurements. However, it demonstrates that the adoption of latest trusted computing extensions could potentially unlock the secure and trustworthy deployment of core application- and trust- related functions, while also respecting the runtime constraints posed by the safety-critical CCAM ecosystem. Specifically, by adopting such TDX-capable infrastructure at the MEC layer may unlock the secure task offloading of critical functions (e.g., inference of video stream data in the context of slow moving detection objects) or the federation of TAF agents achieving a wider perception of the target environment.

Table 6.2: Average and Standard Deviation of Overhead Measurements

Overhead Metric	**SW-based TEE Overhead**	**HW-based TEE Overhead**
Average (AVG)	6.59%	38.78%

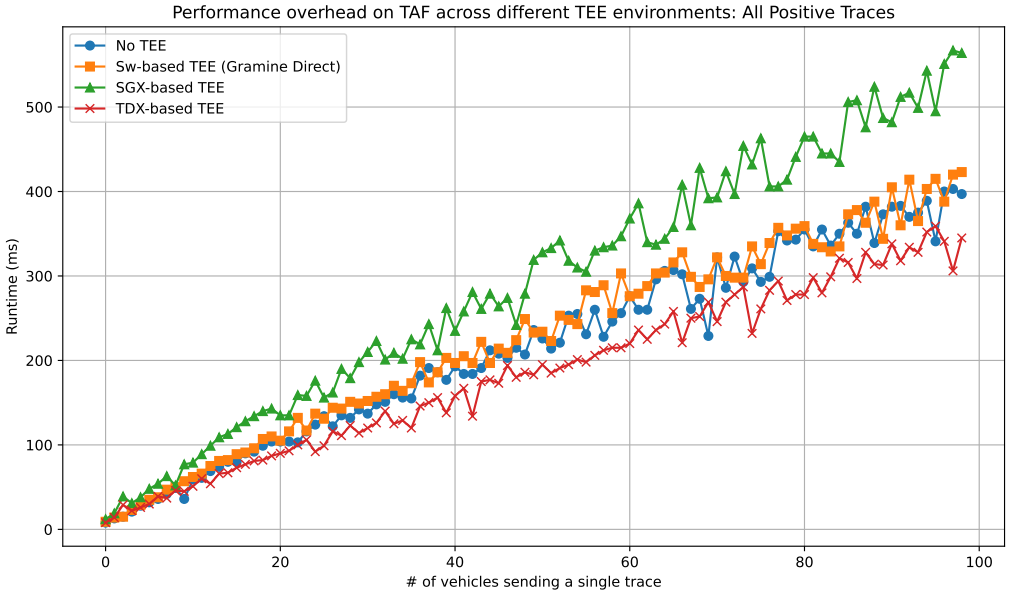


Figure 6.2: Overhead of placing TAF inside a TEE environment (All positive traces)

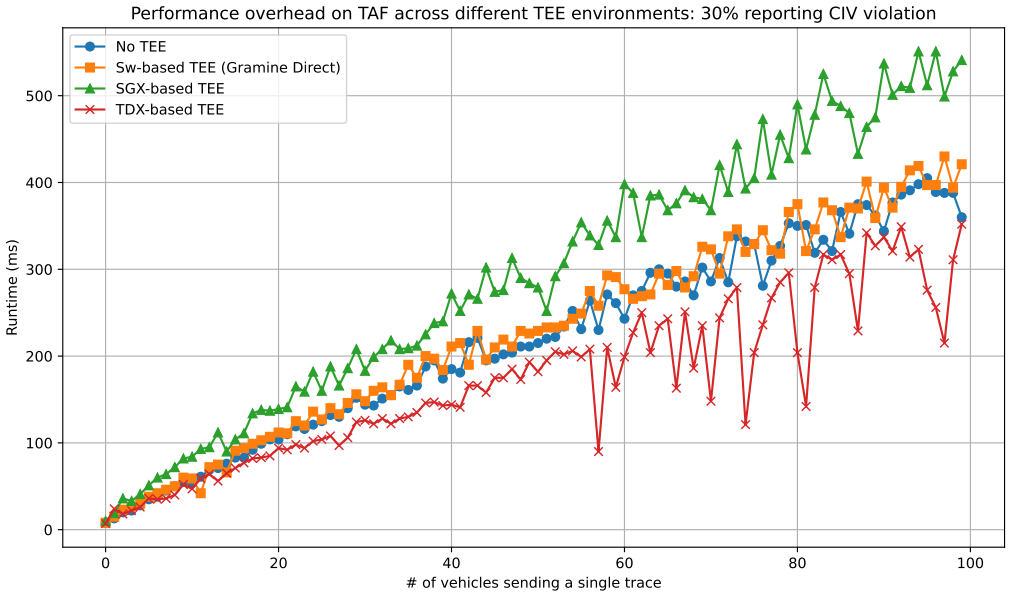


Figure 6.3: Overhead of placing TAF inside a TEE environment (30% of the vehicles report CIV violation)

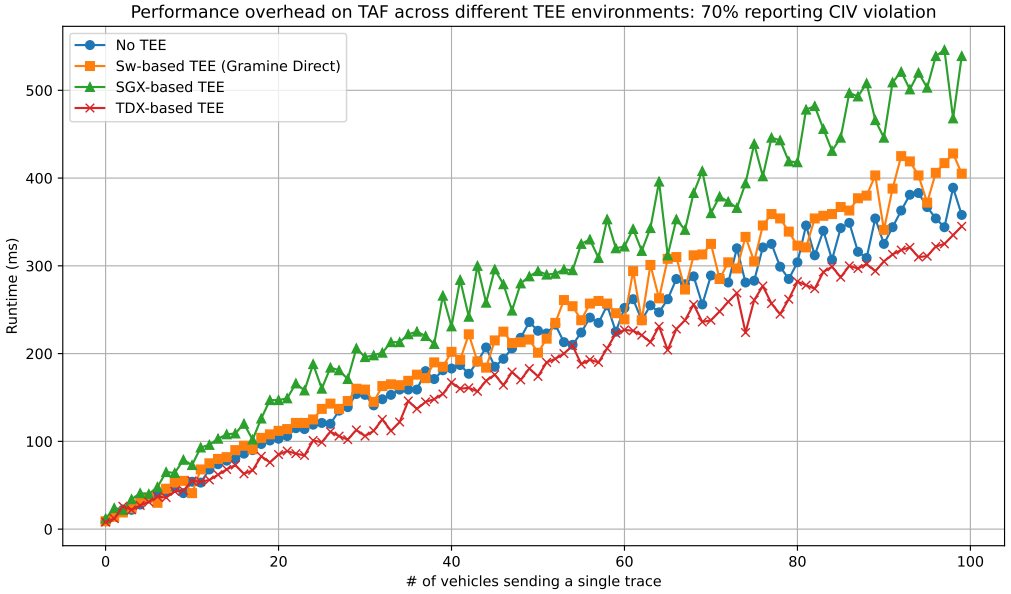


Figure 6.4: Overhead of placing TAF inside a TEE environment (70% of the vehicles report CIV violation)

Chapter 7

Digital Twin

7.1 Motivations for a Digital Twin

Traditionally, a Digital Twin is an architectural pattern where the state of a physical entity is replicated in some way in the digital space. This replication can leverage various technologies like sensors to observe the physical entity, data connections to components inside the physical entity, simulation models which represents the entity's behaviour or 3D/2D models representing the entity. Digital Twins can be used in a range of different use cases, most commonly related to system monitoring, prediction of maintenance, change management or remote operation.

In CONNECT, we want to use such a Digital Twin to support offloading the processing of Trust Assessment Requests from a TAF inside a vehicle to a MEC server. One rationale behind this use case is that Trust Models and Trust Sources may become so large and complex that the computations performed to obtain the Trust Levels will require too much computing power for the processing capabilities available inside a vehicle. With the CONNECT architecture also taking advantage of the capabilities of Edge Computing, we consider it possible to perform those computations in a much less restricted environment while staying close enough to the vehicle in order to achieve acceptable latencies.

So in CONNECT, a digital twin of a trust assessment framework (termed TAF-DT) represents a remote replica of a vehicle's TAF, including all of its state, like trust models, trust sources, and inference capabilities which runs in a MEC device. This is similar but not identical to the definition of digital twins given above, as we do not create a model of a physical entity, but rather replicate a model of trust relationships in a second location.

The offloading capability offered by a TAF-DT could be especially worthwhile in the context of a federated TAF when a MEC server running a TAF-DT for a vehicle could query that vehicle's TAF (or rather its digital twin) without actually communicating with that vehicle. Instead, it now becomes possible to query the state of other vehicles, or trigger computations involving their trust models, without imposing additional load on those vehicles or the network. This would provide numerous advantages, for example, it lessens delays caused by higher priority computations needing to be run on a vehicle when a federated request is received or reduces the exposure of vehicles to denial of service via flooding them with federated requests.

On the other hand, it also creates new challenges to be solved, foremost the replication and synchronisation of the TAF-DT with the original TAF (also called main TAF).

In this chapter, we analyse the TAF-DT in terms of challenges, and requirements, and also pro-

vide generic architectural foundations. We also provide an analysis of how components could be distributed in a Digital Twin deployment and the specification of a synchronization protocol including the associated messages. Finally, we present the results of a preliminary evaluation of the overheads introduced by using a Digital Twin to offload trust computations in the CONNECT architecture, and an outlook on future developments.

7.1.1 Challenges

In this section, we identify the main challenges to design a TAF-DT. While we will not be able to tackle all of them in CONNECT due to limited resources and time, we still aim to provide pointers on how to resolve them.

The core challenge will be to keep the TAF-DT's state consistent with the main TAF, ensure availability of both TAFs, while still allowing some partitioning tolerance. These three properties are interlinked and the CAP (consistency, availability, partition tolerance) theorem shows that only two of them can be guaranteed at the same time, so some compromises will have to be made.

- **Consistency of state:** While strict consistency may come with an unacceptable burden in latency and network overhead, the main TAF and TAF-DT should be eventually consistent, i.e., always converge to the same state. Also the drift of state between both should be limited. As ensuring perfect synchronization between the twin and the physical counterpart is not something we can expect to do easily, to address this challenge we plan to study how trust opinions behave when introducing small changes in the underlying trust model.
- **Availability:** A stable communication link between main TAF and TAF-DT must be maintained while respecting the intermittent nature of automotive communication. While TCP provides many of the required properties, it is not particularly suited for vehicular communication. Another problem for availability will be handovers where the vehicle moves from one cell to another. Pseudonym changes would be another challenge, however, for our practical design and implementation, this will be disregarded to contain complexity.
- **Latency:** Ensuring timely responses and communication between TAFs is a precondition to enable consistency and maintain latency requirements of applications. While this is already challenging for the standalone TAF, federated TAF and TAF-DT will have additional sources of latency. For the TAF-DT this might be caused mainly by synchronization delays. On the other hand, a MEC server querying a local TAF-DT instead of a remote TAF in a vehicle might also bring advantages in terms of latency. Note that in CONNECT, we will only be able to address some aspects of latency, as we do not have full control of the cellular network and telecommunication operator infrastructure.
- **Confidentiality of TAF:** Because the TAF-DT replicates the state of a vehicle, it also contains confidential information about this vehicle and its trust model, so we must make sure that this state does not leak outside of the TAF-DT execution environment. To achieve this, we plan to rely on trusted execution environments investigated in CONNECT in the form of Intel SGX and GRAMINE.
- **Integrity of Computation:** The TAF-DT will be tasked with trust computations on behalf of the vehicle, but it will run outside of the control of said vehicle. Any entities querying the TAF-DT must be able to trust that the TAF-DT itself is trustworthy and its computations cannot be manipulated, for example, by a maliciously manipulated MEC server. To this end, we also plan to leverage trusted execution environments.

Depending on the targeted use cases, a Digital Twin can be implemented in different ways. For example when focusing on Asset management (e.g. for maintenance prediction and equipment monitoring) the bulk of it would be time series made from telemetry data captured by sensors on the physical counterpart. In other use cases, for example if the digital twin itself produces data, it could also contain a simulation model, a 3D representation of the physical counterpart etc..

As our TAF-DT represents a very special notion of a digital twin, we come up with specific challenges and requirements.

As a result, we should correctly identify the scope of the digital twin use case and derive appropriate requirements in order to take appropriate design decisions and select the appropriate technologies to implement it.

7.1.2 Requirements

To fulfill the objectives highlighted previously we define a number of requirements on the TAF-DT

Execution of the TAF computations related to a specific TAF instance in an edge node	
Description	The TAF-DT must be able to transfer requests issued locally in a vehicle to a remote instance of the TAF and execute it there.
Rationale	This requirement implements the basic offloading functionality.

TAF computations in the edge node are based on a replica of the TAF instance state	
Description	The two TAF instances must be able to communicate and synchronize their internal state with sufficient quality.
Rationale	In order to achieve sufficiently comparable outcomes on trust computations, the state of the standalone TAF in the vehicle and the TAF-DT must remain synchronized. Strict consistency will very likely violate latency and other QoS requirements of applications. Therefore, a weaker form of consistency (like eventual consistency) is aimed for. Furthermore, we assume that a event-based update mechanism will be used that delivers events which change the state in parallel to both nodes. The update mechanism can also be opportunistic in that it uses spare communication bandwidth for synchronization.

Communication with other TAF instances inside MEC	
Description	A TAF-DT instance must be able to interact with other TAF instances (in particular those inside the same MEC server) in order to query their opinions or publish its own.
Rationale	This requirement enables functionalities necessary to use replicated TAF instances in a federation use case.

Confidentiality of the TAF internal state inside the edge node	
Description	An edge node must not be able to read or infer the internal state of the replicated TAF instances it hosts.
Rationale	The TAF-DT replicates the internal state of a component of the vehicle. It is the responsibility of the vehicle to allow access to part of its state. Therefore the TAF-DT must not disclose state beyond what the admitted TARs would reveal anyways.

Integrity of the TAF internal state inside the edge node	
Description	An edge node must not be able to modify the internal state of the replicated TAF instances it hosts.
Rationale	The TAF-DT replicates the internal state of a trusted component of the vehicle. The TAF-DT must ensure that it accurately reflects this state.

7.1.3 Implementation in CONNECT

Evaluation of results

In CONNECT, we are mainly interested in the ability of the edge to complement interactions with vehicles. The TAF-DT is mainly pursued as an illustration of these interactions as it can be queried either from a vehicle or the edge. The main research questions to be addressed by the TAF-DT in CONNECT are the following:

- **What are the synchronization constraints between the trust model of the vehicle and of the digital twin which should be considered to produce useful results?** These constraints may refer to a minimum frequency for pushing updates from the vehicle to the twin, whether it is necessary to synchronize the whole state at once or if it can be done incrementally. These results will inform us on the viable solution space for implementing an offloaded trust management solution. The trade-off between consistency of state between the two entities and added latency needs to be investigated through simulation studies.
- **Which strategies can be used to minimize the amount of data transfers to synchronise the twin and the vehicle?** For example, if we can synchronize the trust model partially, how to choose which parts will be transferred? Or can we let the twin build parts of the trust model autonomously and resolve disagreements later? This may lead to identify research opportunities to explore further.
- **Is it feasible to offload the computations of trust opinions for a vehicle to a digital twin?** Based on the results of exploring the previous questions, and taking into account the state of the art regarding the performances of 5G networks, we should be able to compare the requirements for our digital twin solutions to what should be achievable. We also plan to demonstrate the offloading capability in the Slow Moving Traffic Detection use case to provide concrete feedback on the deployment.

As a result, to assess the benefit provided by the TAF-DT, we focus our experiments on the latency overhead added when keeping the state consistent. This divergence will be measured mainly according to the synchronisation rate. The baseline to compare with is the case without TAF-DT where the MEC or an entity sending a TAR have to resort directly to a standalone TAF.

Limitations

In this experiment, we only replicate the TAF and not the whole vehicle as a more comprehensive digital twin would do.
A limitation of our approach will be that performance analysis and measurements will have to resort to a simulation toolchain based on virtual machines, and not be integrated and measured in a real vehicle and edge node. This is partly due to resource constraints, but also due to replicability and controllability of the experiments. The CONNECT consortium does not include

any telecommunication operator which could host the edge node as part of their infrastructure, so any performance evaluation would miss the actual network performance part.

7.2 Architecture of TAF-DT

7.2.1 Digital Twin reference architecture

Standards are currently being developed to assist in leveraging the Digital Twin Pattern in a system architecture. Most notably ISO/IEC 30188 aims to define reference architectures for a variety of use cases. These reference architectures define viewpoints to help different stakeholders design, implement, build or use the system. In the following, we will focus on the Foundational and Functional viewpoints as we are in the early design phases.

The Foundational viewpoint covers concerns such as ”What is the Digital Twin?”, ”What are the implications of building this system as a Digital Twin?”, ”What are the benefits of using a Digital Twin?”, as well as an overview of what is being twinned and how the lifecycles of the twinned object and its digital representation interact together. This viewpoint is roughly covered by the previous section on the motivations behind this Digital Twin.

The Functional viewpoint is defined according to the following taxonomy of base functions: Define, Execute, Monitor, Measure, Analyse, Control and Optimise. In the Design phase of the digital entity lifecycle, the proposed functions to consider are the following:

Base function	Functions
Define	• Model design • Digital thread design • Digital twin system design
Execute	• Better understanding and selection of tools, techniques and data
Monitor & Measure	• Visual design • Simulation of digital twin implementation process
Analyse	• Better understanding and analysis of physical entities
Control	• Reduce system cost
Optimise	• Realize design optimization

Table 7.1: Functions of a Digital Twin in the Design phase

7.2.2 Functional architecture

To further define the Functional viewpoint of the TAF-DT architecture, we go back to the following figure highlighting the main functions in deliverable D2.1.

The elements defined in this figure are as such:

Target and assistance functions

For the Target function, the Vehicle TEE trust management is the standalone TAF, i.e. the function which computes trust levels based on a trust model and trust sources.

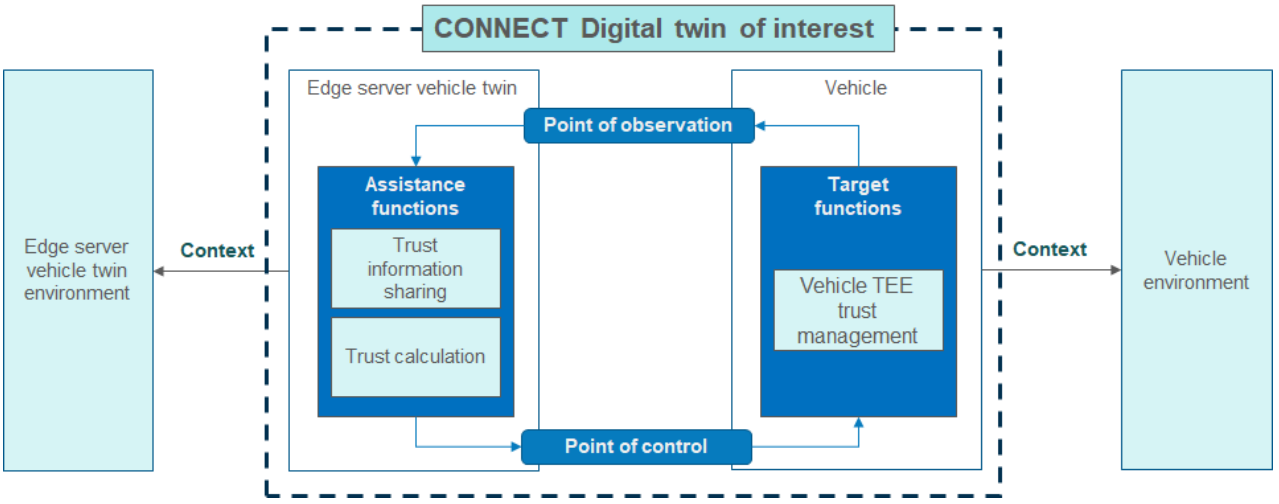


Figure 7.1: Main functions of the TAF-DT

For the Assistance functions, Trust calculation corresponds to the offloading capability and Trust information sharing to exchange capabilities, for example to the Federated TAF.

Points of interaction

Interactions between the twin and the physical counterpart can occur in both direction:

- Physical to Twin:
  - ✓ Synchronisation module integrated inside local TAF. This module is used to synchronise changes in the trust model from a vehicle to its twin.
  - ✓ Sensors connected to MEC infrastructure. Used to gather information from V2X messages which could also be fed into the digital twin.
- Twin to Physical:
  - ✓ Trust computation results. These computations are triggered from requests issued by the vehicle or a remote query.

Support functions

On top of the main functions of a digital twin, we need other functions to support the deployment, execution and ensure security of the TAF-DT:

- Management by MEC:
  - ✓ Create and Destroy twin instances. The MEC infrastructure is in charge of managing the containers in which the digital twin will be hosted, to deploy them inside trusted enclaves and securely destroying them when they are no longer needed.
  - ✓ Route requests to the correct node. The vehicle does not know in advance where or how the digital twin will be deployed so the MEC will have to provide a mechanism either to route transparently requests addressed to a well-known contact point or to provide the specific contact addresses after deployment.

  - ✓ Provide simulated network between twin instances. To support communication between TAF instances, for example in federated use cases, the containers should be able to establish communication channels.
- Security Properties:
  - ✓ Confidentiality and Integrity of twin internal state provided by Gramine/SGX
  - ✓ Authentication between twin and vehicle provided by the mechanisms decribed in D4.2.

7.2.3 Allocation of TAF components

In this section we evaluate the difficulty and potential gains in offloading different parts of the TAF as part of the digital twin.

We consider the following criteria:

- The main goal of the TAF-DT being offloading, components doing heavy computation would be the one which gain most to be hosted in a digital twin.
- Synchronizing state will be one of the main challenges in the TAF-DT. So we want to minimize the amount of data which needs to be synchronized, as part of the state of the component. Inputs and outputs may also be taken into consideration.

From these two criteria, we derive the following characteristics:

- Computation complexity: Evaluates if the components perform heavy or light computation. The main goal of the digital twin is to enable task offloading in the MEC and this characteristic highlights which component would have the most gain to be offloaded.
- State dependency: Evaluates if the computations made by the component depend on some state maintained during its execution or if it is stateless. This characteristics are relevant because any state should be synchronized between the vehicle and the digital twin. Higher state dependency makes a component harder to put in the digital twin because of the amount of data which must be synchronized.
- State volatility: Evaluates the frequency at which the state of the component changes. This characteristic shows how a component would constrain state synchronisation between the vehicle and the twin. A higher volatility implies more stringent performance requirements for the synchronization mechanism.
- I/O complexity: Evaluates the complexity of the data which passes through the component's interfaces. This characteristic takes into account raw data size and complexity of the data structures handled. It is somewhat correlated with State dependency because previous inputs or outputs can be retained by the component as state. A higher I/O complexity on a component which is an interface of the digital twin can make it harder to implement the twin if the connection is unreliable.

By using these characteristics, the ideal candidate for being offloaded in the digital twin is a component which has high computation complexity, low state dependency, low state volatility and low I/O complexity. However, a high state dependency may be counteracted by a low state volatility because then it would not need to be updated as often.
The following table summarizes how each component of the TAF scores for these characteristics:

Component	Computation complexity	State dependency	State volatility	I/O complexity
TMM	medium	high	high	high
TSM	low	high	high	high
TLEE	high	none	none	high
TDE	low	low	low	low

Table 7.2: Complexity of integration in TAF-DT

From these results, we see that the TMM and TSM are the hardest components to replicate on the digital twin, managing both complex and volatile states while not being computationally intensive. Meanwhile, the TLEE does not need to manage an internal state but performs a lot of computations, the main issue being its dependency to complex inputs from the TMM.

An ideal configuration of components may be to replicate the vehicular TMM, TLEE and TDE in the digital twin and not replicating the TSM, relying instead on a shared TSM among replicas to listen for evidences available from the V2X network and only synchronising evidences coming from source inside the vehicle. This should be coupled with a caching mechanisms that caches trust source state in the digital twin and which gets updated with an event-based mechanism that guarantees eventual consistency but not strict consistency. This will allow the TAF-DTs view on trust sources to lag behind to a certain but limited extend, but will make sure that the TAF-DT is guaranteed to catch up as new updates arrive. What needs to be avoided is a systematic drift of the TAF-DTs state from its twin's standalone TAF's state. Furthermore, we need to investigate how this drift will affect application quality when relying on a TAR sent to a TAF-DT instead of the original TAF. We expect here that some deviation is tolerable, as updates of Trust Sources will, depending on the type trust source, also happen with a certain delay in the standalone TAF.

7.2.4 Additional messages specification

To support the replication of a Trust Model state across the local TAF and TAF-DT, an additional external interface needs to be added to the TAF, alongside corresponding messages to manage the twin lifecycle and propagate updates.

Trust Model Initialization

The `TAS_DT_TMI_INITIALIZE` message is sent from the local TAF to the TAF-DT once a new trust model instance has been spawned locally at the local TAF. This message makes the TAF-DT to spawn the same trust model instance remotely. The initialization message contains the following information:

- the identifier of the instance in form of a fullTmiID (full TMI identifier) that includes the trust model template to be used
- sessionParams: optional parameters with which the session has been spawned (e.g., application-specified weights)
- initializeParams: optional parameters which the trust model instance should use (e.g., to bootstrap the initial trust graph)

Trust Model Update

The TAS_DT_TMI_UPDATE message is sent from the local TAF to the TAF-DT whenever a local update to a trust model instance occurs. The state of a trust model instance is determined by it's initial state as well as a sequence of update operations applied to that trust model instance. If two instances have the same initial state and receive the same sequence of identical updates, they have the same eventual state. This mechanism is used for each trust model instance when replicating changes from the local TAF to the digital twin. Updates are strictly serialized in a sequential order and have version number that reflects the number of applied updates. Each update yields an increment of the version number of the state of the trust model instance.

For efficiency reasons both within the TAF and when replicating the state between the TAF to its replicas, micro-batching of updates can occur. If a TAS_DT_TMI_UPDATE message contains more than one update, it should first apply all available updates (ordered by the version numbers) before submitting the trust model instance to further processing (i.e., calling the TLEE/TDE). This allows to bundle smaller updates for network efficiency (i.e., fewer messages with less overhead) and for processing efficiency (i.e., less TLEE calls for intermediate states with updates pending).

Digital Twin ATL Update

The TAS_DT_ATL_UPDATE message is used by a TAF-DT to send the calculated ATL results back to the actual TAF. This message represents an (asynchronous) response to a previous TAS_DT_TMI_INITIALIZE or TAS_DT_TMI_UPDATE messages sent from the local TAF to its replica. The message specifies the trust model instance as well as the version of the trust model instance to which the update refers to.

Trust Model Deletion

The TAS_DT_TMI_DELETE message is used by the local TAF to tell the TAF-DT that the specified trust model instance is no longer used and should be removed in the digital twin as well. When this message is sent, the TAF-DT should not expect any further updates for this trust model instance. Also, once this message has been processed by the TAF-DT, the local TAF will not expect any additional TAS_DT_ATL_UPDATE messages from the TAF-DT, as deleting the trust model instance also removes the necessity for the replica to calculate ATL updates.

7.3 Evaluation

7.3.1 Description of scenario

The TAF-DT evaluation is done over two main scenarios involving the same overall architecture. The components needed are:

- Two TAF instances, one for the local TAF and the other for the TAF-DT.
- A trust source used to feed evidences and trigger trust computation on both TAF instances.
- A communication channel common to both the local TAF and TAF-DT

Two different machines are also needed to take into account that the TAF-DT is running in an edge node, separate from the local TAF. One will host the local TAF and the trust source, the other will host the TAF-DT.

For this evaluation, the focus is on showing whether a TAF-DT introduces noticeable delays or latencies to trust computations. The following scenarios do not showcase actual synchronization of the trust models through the mechanisms described previously as we were not able to implement them in the TAF during the CONNECT project. However, the test scenarios introduce the edge constraints by physically separating the TAF-DT from the evidence generation, forcing the evidences to go through the network to reach the Digital Twin. Also, in the CONNECT architecture, edge nodes execute trust-related operation in trusted enclaves, which adds more potential for overhead. As such, another requirement on the machine hosting the TAF-DT is to use a trusted enclave environment to load the TAF-DT.

The base evaluation scenario is as follows:

1. Preamble: Local TAF, TAF-DT and Trust Source are operational on their respective host. Both local TAF and TAF-DT have loaded the same Trust Model.
2. An application subscribes to trust notifications to both local TAF and TAF-DT simultaneously
3. As a result of the subscription, both local TAF and TAF-DT subscribe to evidences from the Trust Source
4. The Trust Source produces evidences sent to both local TAF and TAF-DT
5. Local TAF and TAF-DT produce notifications to the application

The main metric considered in this scenario is the time elapsed between the production of the trust evidence triggering a notification by a TAF, and the production of the actual notification. This measurement showcases the end-to-end processing time of each TAF.

Two variations on this base scenario are used, depending on the type of notification requested by the application:

- By subscribing to the ATL, the TAF sends a notification each time an ATL in the Trust Model changes. By providing a sequence of evidence which always produces changes in the Trust Model, this scenario can measure the end-to-end processing time for each evidence. The parameters of this scenario are the frequency at which evidences are sent by the Trust Source and its duration.
- By subscribing to a Trust Decision, the TAF only sends a notification when an ATL crosses an RTL. This scenario allows to control how many evidences the TAF processes before sending a notification. As a result it shows if delays occur internally as a result of previous evidences processing. The parameters of this scenario are the frequency at which evidences are sent by the Trust Source and the number of evidences to send before sending the one triggering a notification.

7.3.2 Setup

The evaluation setup for the TAF-DT is as such:

- The TAF-DT is hosted in a container configured to use a Gramine enclave
- The Trust Source is an AIV service

- A Kafka broker is deployed alongside the local TAF and serves as the communication channel
- Both TAFs will use the VCM model as described in the attestation use case

The measurements are based on the timestamp attached to each record when they are received by the Kafka broker, also known as Log Append time. This represents the earliest time the message can be processed by its recipient, regardless of when it is actually read by the application. This allows to measure the end-to-end processing time from when the broker receives the Trust Event up to when it receives the trust levels. It also removes the need to synchronize clocks across the different hosts.

The local TAF, Kafka Broker and AIV service are hosted in the same physical machine. The TAF-DT is hosted on a machine located in a different physical location. The measured ping time between both hosts is around 10ms. The TAF-DT is also executed in a container configured with a gramine enclave using the gramine direct emulation layer. This setup is intended to simulate the edge configuration with the main services and the local TAF inside the vehicle and the TAF-DT located in a remote server at the edge.

In both scenarios, the main sequence of messages exchanged is represented by the following sequence diagram:

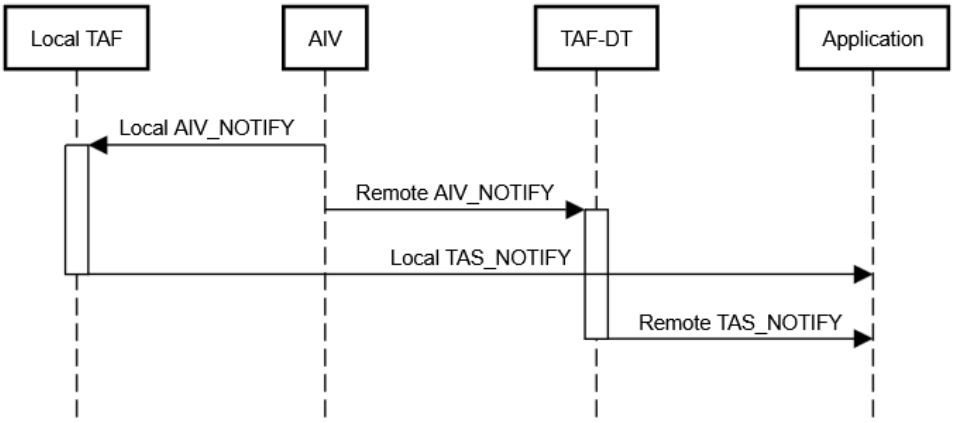


Figure 7.2: Digital Twin test sequence

7.3.3 Results

ATL Notify

For the ATL subscription setup, measurements have been made across three runs with an interval of 1s, 250ms and 25ms respectively, over a duration of 5 minutes. The AIV service sends an `AIV_NOTIFY` to both the local TAF and the TAF-DT at the set interval.

The measurement used in this scenario is the end-to-end processing duration by each TAF. Referring to the sequence diagram, this corresponds to the difference between the timestamp of an `AIV_NOTIFY` and its subsequent `TAS_NOTIFY` for both the local TAF and TAF-DT.

The results are presented in Table 7.3 to provide specific numbers characteristics of the measures, and Figure 7.3 to provide an overview of the entire run of measurements.

Interval	1s		250ms		25ms	
	local	twin	local	twin	local	twin
Average	6.41	35.5	5.85	42.2	5.58	30.5
Standard deviation	0.880	13.4	1.34	17.4	1.02	22.5
p20	6	22	5	25	5	19
p50	6	37	6	49	5	24
p99	9	73	8	76	8	117

Table 7.3: Summary of ATL Notify test results (ms)

These results show that the frequency at which the trust events are sent does not have a significant effect on the time the TAF processes them as it stays stable around 6ms for the local TAF and 35ms for the TAF-DT. As expected, because of network transit the TAF-DT processing is longer. This scenario implies that at least one round trip is made between the Kafka broker, located with the AIV and local TAF, and the TAF-DT so a minimum of 10ms additional time would be expected. In practice the processing time is much higher than this as it is also more jittery. Deviation is around 40% of the average processing time for the TAF-DT compared to less than 20% for the local TAF. However, as shown by the plots in Figure 7.3 and the p20 figure is also consistent with the expected additional time, showing that the lowest measured processing times for the TAF-DT are around 20ms.

The 99th percentile figures also tend to show that the lower the interval, the higher the processing time will get. This is also evidenced by Figure 7.3 which shows an increased chance for a random spike in processing time to occur. However, the processing time is usually in line with performance expectation, with p99 staying below 100ms except for the 25ms interval test.

Trust Decision

For the Trust Decision setup, the measurement used is not the same processing time as for the ATL notify test, but the difference between the timestamps of the `TAS_NOTIFY` messages sent by the TAFs. This is because this scenario aims to show the delay which should be expected from using the TAF-DT compared to the local TAF. Because the processing in the TAFs are triggered by two different messages, there could be a hidden advantage for the TAF which gets the first `AIV_NOTIFY`. However it has been verified that the timestamps for each message are the same at a milliseconds resolution, with some deviation of a few milliseconds.

Interval	1s		250ms			25ms		
State duration	10	50	10	50	100	10	50	100
Average	43.2	26.4	62.8	49.4	49.1	27.5	29.7	26.1
Standard deviation	11.3	7.92	15.1	13.2	8.86	20.3	37.2	18.3
p50	45	21	67	53	51	22	22	20
p99	64.84	36.88	82.82	72.5	59	66	197.52	52.61

Table 7.4: Summary of Trust Decision results (ms)

Similarly to the previous test, the notification delay for the Trust Decision has been measured while using the following intervals between messages sent by the AIV Trust Source: 1s, 250ms, 25ms. Additionally, in order to evaluate the effects of changes in a trust decision being less frequent compared to the number of trust events processed by a TAF, the Trust Decision state has

Figure 7.3: Message processing time (ms) over a period of 5 minutes (Orange: DT, Blue: Local)

been maintained for 10 or 50 messages before switching with each of these intervals, which is presented as State duration in Table 7.4. Another scenario with the state maintained for 100 messages has also been tested with the 250ms and 25ms intervals, but not the 1s interval because of the run time required to get a significant number of notification with these parameters.

The main visible take away from these results is that sending messages more frequently seems to reduce the notification delay while the deviation rises slightly. This is in line with the results shown in the ATL notify test. These results also show a trend of getting higher delays for a decision when trust levels change more frequently. However, as the 99th percentile values show, they can also be heavily impacted by random excursions.

Conclusions

From these two scenarios, we can conclude that the TAF function does not seem to provide significant obstacles to being placed in a Digital Twin. The differences in processing time appear to be mainly caused by the network and do not depend on the rate at which messages are exchanged. The main issue which can be shown is that response times are less stable through the Digital Twin, going from variations on the order of the milliseconds to tens of milliseconds. This leads to relative processing times which can be up to more than 5 or 6 times higher than local processing. However these variations, and the associated delays in reaction time, remain under 100ms in most of the observed test conditions (with both network nodes located 10ms apart in distinct physical locations). This represents an acceptable level of responsiveness, in line with the KPIs expressed for the TAF of 200ms latency for safety-critical CCAM functions. The occurrence of excursions caused by random network latencies can, however, be problematic but should be mitigated by adapting the synchronization rate, as the tests with a 25ms interval between trust events were the ones most impacted by these excursions, or with specific guarantees from the network operator which should be in place in an edge scenario.

Another trade-off which seems to appear in the trust decision test is that updating the state more frequently slightly reduces the average delay introduced by the Digital Twin but, similarly to what is observed with processing time in the ATL notify test, leads to an overall bigger spread in the observed delays.

7.4 Outlook and further works

The work described in this section is a preliminary step in order to produce more involved digital twins for the TAF. We have provided a specification for synchronizing a TAF instance with another in a remote node. Furthermore, we provided a preliminary performance evaluation showing that the delays inherent to this approach are compatible with the constraints of the TAF execution.

The main limitations of this work are the following:

- Due to time constraints in the project, the synchronization protocol described in this chapter could not actually be implemented. The evaluation relies on an alternative approach, relying on broadcast of trust events which main drawback is that the remote TAF has to initiate direct contact with the trust sources.
- The links between the nodes in the evaluation go through the Internet instead of the edge. Additional work is needed to better characterize how the quality of the network link, and the

ability to run the TAF-DT in the edge, which implies more guarantees in terms of latency and stability, instead of a standard cloud server affects the results.

To elaborate on this work, the network link characterization is the obvious way to better understand how it affects the TAF behaviour. Another possibility is also to evaluate a different repartition of components, especially the message broker. In the experiment, we used the broker already present along the local TAF and trust source to handle all messages but the TAF-DT could also use a dedicated broker, hosted in the MEC.

Additionally, while we were not able to integrate both together, a Digital Twin should be a good complement to the Federated TAF concept. By leveraging TAF-DTs on the MEC, federated queries would not need to be handled by vehicle TAFs and avoid imposing an additional burden.

Chapter 8

Risk Assessment Framework

8.1 Overview and updates on second release

The realization of the CONNECT Trust Assessment Framework is based on the careful and accurate specification of all the trust relationships between the components that comprise the target CCAM function under evaluation. At its core, this specification relies on a thorough risk analysis that needs to take place both during the design phase of the target CCAM function but also throughout its life-cycle so get an up-to-date view on the inflicted risk of the system. In fact, this can be observed by the fact that both the ATL configuration (Chapter 9) - i.e., the weights in the evidence quantification functions - and the RTL equations (Chapter 10) are based on the maximum risk that is identified in the system. Hence, the execution of a proper threat analysis and risk assessment in the target environment constitutes a core enabler in the realization of the trust decision engine, linking together the context behind the RTL and ATL values.

The importance of accurately characterizing the risk associated with the target system within the broader TAF pipeline is underscored by the structure of the risk assessment lifecycle. According to the Threat Analysis and Risk Assessment (TARA) framework [1], the Security Administrator must first provide a detailed component diagram that models the CCAM function under evaluation. This step involves identifying all relevant assets and their inter-dependencies (i.e., component diagram), as well as specifying all plausible attack paths that could lead to one or more forms of system damage. Through this process, concrete risk scenarios associated with the target CCAM function are systematically derived.

Based on these risk scenarios, the Security Administrator must then determine an appropriate risk treatment strategy — deciding which risks require mitigation through the enforcement of security controls and which may be accepted. This decision ultimately leads to the specification of the accepted (residual) risk level under which the target CCAM function is permitted to operate. Within this context, RTL constraints represent the threshold values — such as the minimum acceptable belief level in subjective logic terms — that define acceptable runtime conditions. Concurrently, the ATL values computed at runtime offer an evidence-based estimation of the actual trust level, enabling the Security Administrator to determine — within a defined margin of uncertainty — whether the system continues to operate within the bounds of the accepted risk level.

This introduces the need for a CONNECT RA Engine that not only provides the base functionalities to perform the necessary risk assessment tasks, but also enables the realization of the ATL and RTL calculations and consequently the entire CONNECT Trust Assessment Framework. In

what follows, we present the second iteration of the Risk Assessment Tool, developed as part of the ongoing effort to improve threat modeling and risk evaluation methodologies within the CONNECT framework. Building on the foundation established in the initial version of the CONNECT RA Engine presented in [5], this updated release introduces significant improvements in both functionality and integration capabilities, with a particular focus on the enablers of the RTL calculations presented in Chapter 10.

8.2 Functional Specifications

This section summarizes the list of functional specifications presented in [5] and reports the progress that has been made in the second release of the CONNECT RA Engine. Table 8.1 summarizes the full list of requirements and highlights the progress made on the pending points.

First, FR_RA_6 refers to the ability of the CONNECT RA Engine to carry out an attack path assessment and identify cascading attacks across the target topology. To this extent, one of the core features of this second version is the realization of the Attack Path Calculator component which is responsible for processing the available threat analysis knowledge and provide a list of possible attack paths that would allow an adversary to pivot from one asset in the topology to another - i.e., in the in-vehicle topology this may refer to leveraging a vulnerability in one ECU to eventually attack the main in-vehicle computer. Conceptually, this level of automation introduced by this requirement alleviates some of the threat analysis required in the context of a full-blown risk assessment process, adhering to the standardized Threat Analysis and Risk Assessment methodology [1]. Specifically, one of the heavy processes assigned to the TARA practitioners is the thorough identification of attack paths that unlock adversarial actions on target vehicle behaviors and introduce damage scenarios with operational, safety, financial and/or privacy impact. The results provided by the Attack Path Calculator component may help the practitioners to prepopulate the list of attack paths with concrete attack scenarios and update/enhance the provided list with additional cases based on their knowledge and expertise. Section 8.4 showcases the technical details behind the Attack Path Calculator component and provides a walkthrough on the steps that are followed in order to derive the list of attack paths, ready to be transferred to the TARA threat analysis process.

Secondly, according to FR_RA_10, it is essential for the CONNECT RA Engine to be able to consume various Indicators of Compromise (IoC) reported by the operational environment in the context of a CCAM topology. One relevant example of such an IoC stems from the attestation appraisals produced as part of the Attestation and Integrity Verification (AIV) component. Essentially, based on the requirement, upon receipt of negative/failed attestation evidence the CONNECT RA Engine should be able to update the risk graph and eventually re-assess the risk posture of the target environment. To this extent, this second release is shipped with two possible ways that enable this dynamic re-evaluation of risk. On the one hand, authenticated and authorized OEMs and Security Administrators can access the graphical user interface (GUI) of the CONNECT RA Engine and update the risk graph based on new information from the available IoCs. Of course, this task may involve additional pre-processing where the indicators - e.g., runtime traces proving that an ECU is at an incorrect state - need to be evaluated in order to identify whether the indicator originates from an already considered or a zero-day vulnerability. At the same time, integration with well-known repositories that provide up-to-date information on threats and vulnerabilities — such as the National Vulnerability Database —enables the periodic and dynamic recalculation of the risk graph for the target environment. This occurs both when

new entries are published and when existing vulnerabilities are re-evaluated in terms of criticality (e.g., when a vulnerability initially assessed as low-risk is later upgraded to high or critical severity due to newly discovered exploitation methods or real-world attacks). Section 8.6 provides all the interfaces implemented in the CONNECT RA Engine, including those to update the threat analysis of the TARA process and to perform a revised risk analysis.

In addition, as illustrated in FR_RA_11, one of the core enhancements of the second release of the CONNECT RA Engine is the development of the necessary interfaces for realizing the calculation of the RTL thresholds (Chapter 10). Specifically, Section 8.5 presents the concrete steps that need to be followed by a Security Administrator in the CONNECT RA Engine GUI in order to conduct the necessary TARA process and eventually calculate the RTL constraints. Of course, the TARA-related information available to a Security Administrator is also completed with the (failed) attestation traces that are exposed by the CONNECT Distributed Ledger Technology (DLT). A detailed reporting of these DLT interfaces is presented in [10].

Finally, the last requirement that is taken into account as part of the second release has to do with the harmonization of the different risk assessment methodologies - namely the CVSS-based and the TARA methodologies. Specifically, each of the two methodologies considered as part of the CONNECT RA Engine has a concrete risk quantification equation that converts the threat analysis knowledge into a score per risk scenario. Given that as part of the standardized TARA framework there is not a single, normative risk quantification equation that fits all needs, the aim is to express the available risk scores with a common scale. This unlocks an additional set of experiments that could potentially showcase the robustness of the RTL equations and shed a light on the interplay between the risk assessment pipeline and the relaxation (or strengthening) of the RTL constraints (see Section 8.7).

ID	Func	Information Flow		Implementation Status
		I want to <Action>	so that <Reason>	
1	FR_RA_1	model various types of assets	I can capture all the entities that characterize the CCAM continuum	Done (1st Release)
2	FR_RA_2	capture the interdependencies among the various assets	I can model the component diagram and data flows that characterize the asset graph	Done (1st Release)
3	FR_RA_3	visualize the entire asset cartography	I can execute the necessary threat analysis for the risk assessment methodologies	Done (1st Release)
4	FR_RA_4	capture different trust properties in the service graph chain	I can have a better overview of the relationships and nodes that need to be assessed per trust property	Done (1st Release)
5	FR_RA_5	have access to the latest open intelligence information available related to threats and vulnerabilities	I can associate them with the monitored assets and evaluate their impact on the risk assessment	Done (1st Release)
6	FR_RA_6	identify the attack paths comprising vulnerabilities that allow the propagation of attacks across multiple assets	I can capture the cascading effects that enable an adversary to pivot from one asset to another through the exploitation of the associated vulnerabilities	**Done (2nd Release)**

ID	Func	Information Flow		Implementation Status
		I want to <Action>	so that <Reason>	
7	FR_RA_7	perform risk assessment periodically as well as asynchronously	I can have the latest risk values for the entire topology right after an incident takes place	Done (1st Release)
8	FR_RA_8	associate assets with security controls that address - fully or partially - specific risks	I can enable the calculation of impact in the risk analysis when optimal sets of mitigation mechanisms are in place	Done (1st Release)
9	FR_RA_9	define a set of security controls for the various assets in the topology into a mitigation strategy	I can re-perform the risk analysis and evaluate its effectiveness in the overall risk of the topology	Done (1st Release)
10	FR_RA_10	re-calculate the overall risk of the topology (representing a service graph chain) when new vulnerabilities may be discovered from any monitored trustworthiness evidence (e.g., evidence from a "*failed*" attestation report)	I can re-calculate the RTL and communicate it across the TAF instances	**Done (2nd Release)**
11	FR_RA_11	provide an automated and seamless risk assessment analysis	I can achieve the maximum level of automation on the calculation of both ATL and RTL	**Done (2nd Release)**
12	FR_RA_12	converge outputs from both qualitative and quantitative risk quantification models	I can provide a more comprehensive risk assessment for the asset graph of a CCAM function, tailored to the set of trust properties of interest	**Done (1st Release)**

Table 8.1: Functional specifications of the CONNECT Risk Assessment Engine

8.3 CONNECT RA Conceptual Analysis

This section presents the internal architecture of the CONNECT Risk Assessment (RA) framework. Specifically, it starts from a high-level overview of the overall CONNECT RA framework. Subsequently, it presents the CONNECT RA architecture, highlighting the core updates of the internal components in the second release and how the interactions between the components are realized.

As presented in the initial version [5], the CONNECT RA Engine is built on top of UBITECH's OLISTIC risk assessment platform. Initially, it supported a generic CVSS-based methodology, but it has since been extended to accommodate the full TARA framework. In this second release, the CONNECT RA Engine offers two key advancements: (i) the addition of new components — namely, the Attack Path Calculator and the Attack Feasibility Recommendation Engine — which support various aspects of the TARA threat analysis process; and (ii) enhancements to existing components to more accurately model the threat analysis information within the TARA context.

8.3.1 High-level Flow of Actions

Figure 8.2 illustrates the latest architectural details of the CONNECT RA Engine. With the latest updates implemented both in the context of the Attack Path Calculator and the interfacing related to the RTL calculations, Figure 8.1 presents the positioning of the CONNECT RA Engine in the overarching CONNECT framework. Working in tandem with the RTL Calculations (see Chapter 10), the CONNECT RA Engine is capable of collecting all the threat analysis collected from the relevant OEMs and Security Administrators and deriving the risk graph of the target (e.g., in-vehicle) topology based on the TARA framework. As depicted in Figure 8.1, the risk analysis starts with the a new Request For Risk Assessment (RFRA). This can be triggered either manually (e.g., through an action performed by a security Admin) or automatically (e.g., through a periodic, scheduled task). This request (Step 1) invokes the CONNECT RA Engine to aggregate all of the information collected through the threat analysis and provided by the Security Administrator (e.g., asset topology, associated threats, attack paths and threat scenarios) and quantify the risk score for each risk scenario that is specified. This task is not executed only once; according to the TARA framework once all the risk scenarios have been identified and their risk is quantified, the security Admin shall decide on the treatment strategy for each entry in the report (see Section 8.3.3). When deciding to mitigate any of the risk scenarios, the security administrator shall specify the security control that addresses a particular risk and quantify the decrease in its attack feasibility. Hence, the collection of the results for all the iterations of the TARA process - encompassing also the security controls in place - provides a holistic view of the risk posture of the entire target topology (Step 2).

Based on the filtering capabilities of the CONNECT RA Engine, it is possible to group risk scenarios per security property (e.g., integrity, confidentiality) and per asset or data item. This capability unlocks the realization of the RTL calculations (Step 3) for a specific context as presented in (Chapter 10). Depending on the context that needs to be evaluated in the CONNECT Trust Assessment Framework, the relevant RTL values are stored on the CONNECT DLT (Step 4), as part of the trust policies that need to be enforced by the in-vehicle TAF instance (Step 5). During the operational phase, on-going trust assessments and runtime traces provide additional knowledge to the Security Administrators (Step 6 - 10). This information provides rich indicators of compromise that could potentially lead to a revision in the runtime risk characterization of the target environment, which in turn will be reflected in a revised set of RTL values for the affected security properties and trust objects.

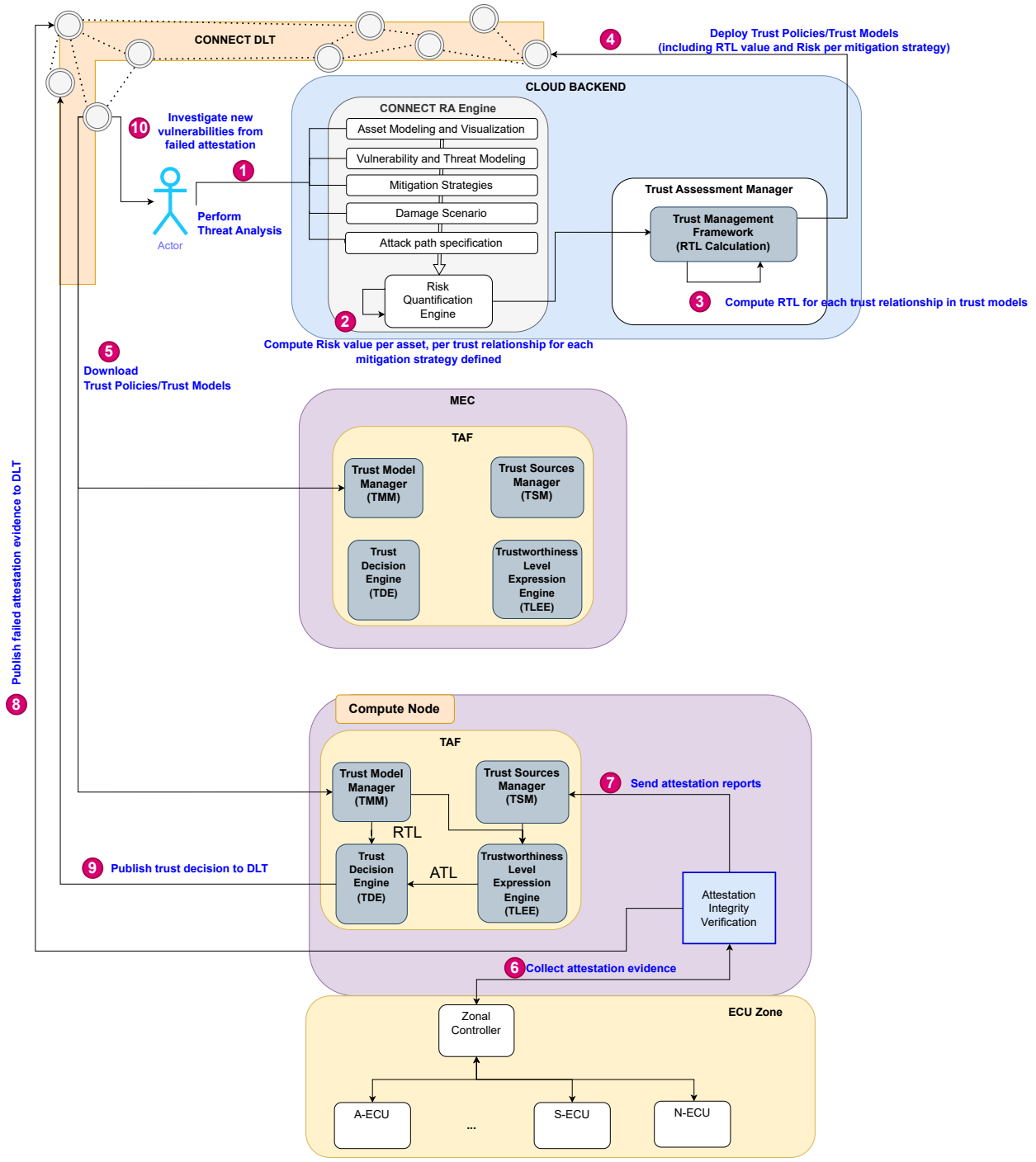


Figure 8.1: Risk Assessment Engine in the CONNECT Architecture

8.3.2 Component Analysis

Figure 8.1 presents the main aspects of the threat analysis (e.g., Asset Modeling and Visualization, Vulnerability and Threat Modeling) as part of the CONNECT RA framework. These capabilities are offered by the various internal components of the CONNECT RA Engine (Figure 8.2). This section provides an architectural blueprint of all the elements that comprise the CONNECT RA Engine, including a description per sub-component and a report on the latest updates that have been implemented as part of this second version.

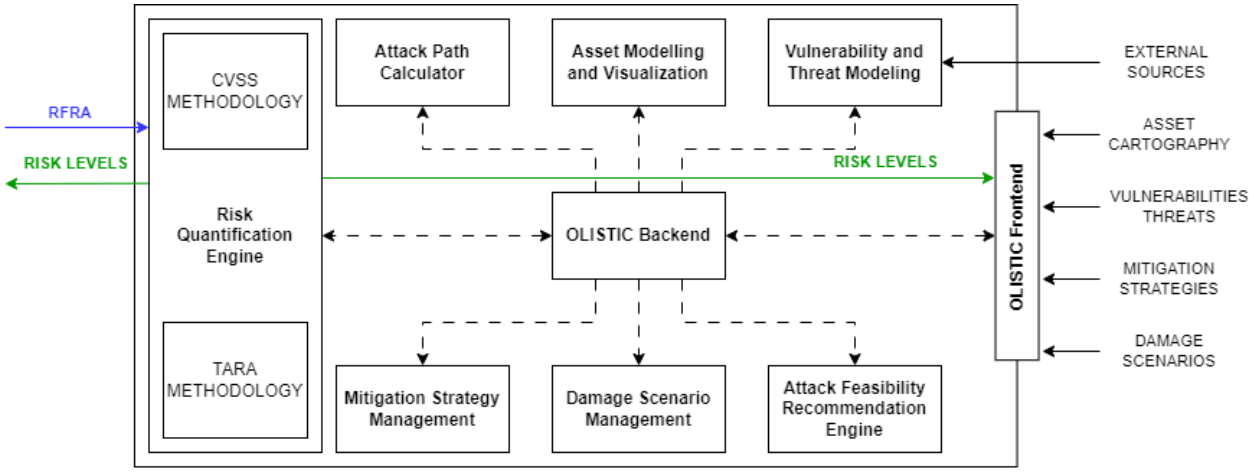


Figure 8.2: CONNECT RA Architecture

OLISTIC Backend and Frontend

The OLISTIC backend offers the necessary APIs to orchestrate all the backend operations of the engine and works in synergy with the frontend to offer the necessary functionalities to the security analyst. The backend component is also responsible for performing the access control rules that allow authorized users to use the available APIs. Finally, it provides the necessary Publish/Subscribe interfaces through Kafka clients to enable the asynchronous consumption of security incidents reported by other CONNECT components. The OLISTIC frontend offers an interactive dashboard which is used for visualizing the digital representation of the cyber-physical environment.

In the context of the second release, a few critical updates have taken place both in the backend and the frontend logic of the CONNECT RA Engine. First of all, the backend API services have been updated to expose the necessary endpoints that provide all the risk scores for the RTL belief and disbelief equations (Chapter 10). This includes not only the implementation of new endpoints but also the supporting of concrete filtering capabilities for isolating those risk scenarios that are relevant for particular trust propositions (e.g., querying for all the risk scenarios that are relevant for the main on-board unit and have to do with compromising its integrity). In addition to that, the OLISTIC Frontend is also enhanced with new capabilities that allow the Security Administrator to provide more information pertaining to the threat analysis in the context of the TARA process. In the remainder of this subsection, more details on the graphical user interface enhancements are presented per component of the CONNECT RA Engine.

Asset Modelling and Visualization Component

This component is responsible for modelling the list of assets that comprise the monitored CCAM target environment. This component enables a Security Admin to model the target component diagram that represents a CCAM function whose trustworthiness is intended to be assessed. In this context, the term asset needs to be as abstract as possible encompassing any tangible entity (e.g., hardware equipment like ECUs and in-vehicle sensors) and intangible entity (e.g., data items, data flows or software services). Consequently, this enables the association of threats, attack paths, and security controls in each asset in the target topology.

As part of the second release of the CONNECT RA Engine, this component has been sub-

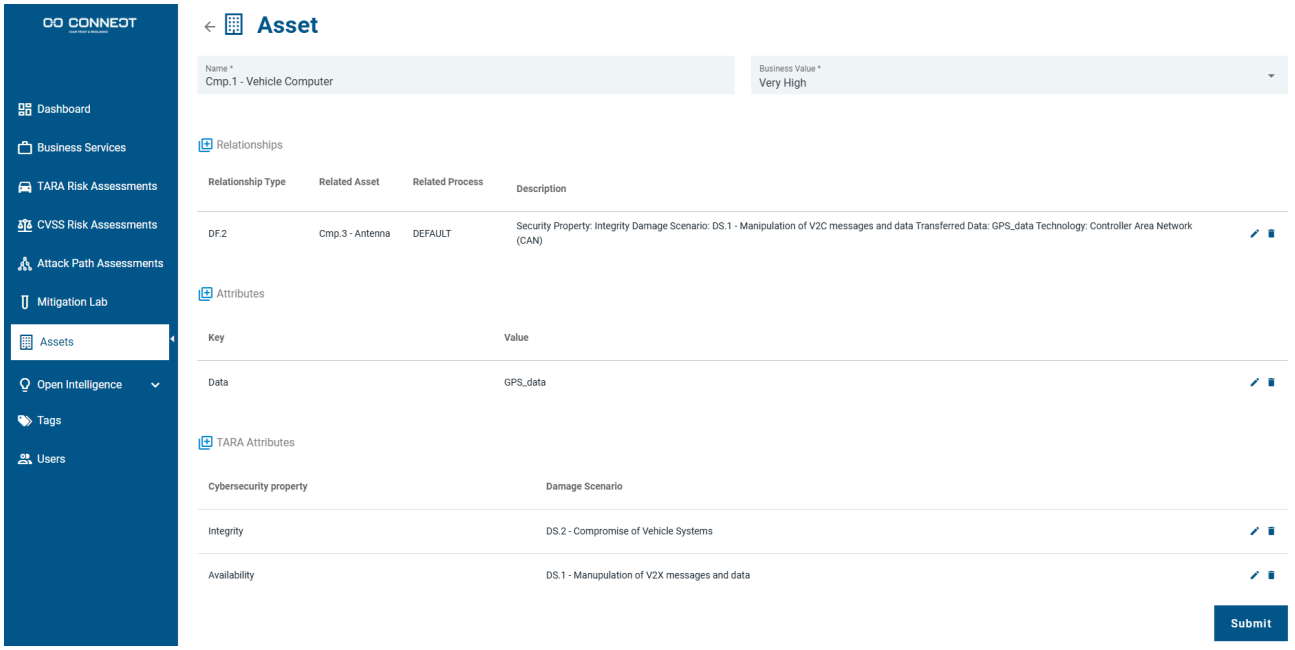


Figure 8.3: Asset and relationships data modelling

jected to important updates to allow the specification of fine-grained information per asset and the inter-dependencies between them. Figure 8.3 demonstrates the level of information that can be provided by the security administrator. Building on the first release, this component allows users to define custom asset relationships to better represent the various data flows between assets. It also enables the specification of crucial metadata, such as the technologies used to establish links between assets.

Vulnerability and Threat Modelling Component

The main goal of this component is to enable the data modeling of vulnerabilities and threats that are either specified based on the knowledge and expertise of Security Administrators or retrieved by well-known cyber threat intelligence platforms such as the NVD. Consequently, even from the first release, the Vulnerability and Threat Modelling Component has been equipped with the logic to capture vulnerability characteristics and threat knowledge based on well known data models, namely Common Vulnerability Scoring System and STRIDE respectively.

As part of the second release, refactoring efforts have focused on the re-design of the vulnerability data model so as to be able to express multiple CVSS profiles for a single vulnerability entry (see Figure 8.4). This will allow for the expression of vulnerability characteristics coming from different versions of the already adopted versions of CVSS - namely v2 and v3.1. At the same time, this will facilitate the future support of the latest v4.0 specification.

Damage Scenario Management Component

The definition of damage scenarios is pivotal in determining the risk associated with threat scenarios within the TARA methodology. The damage scenario management component facilitates the fine-tuning of each identified damage scenario's impact enabling Security Admin to refine their risk assessment analysis. According to the TARA specification, each damage scenario

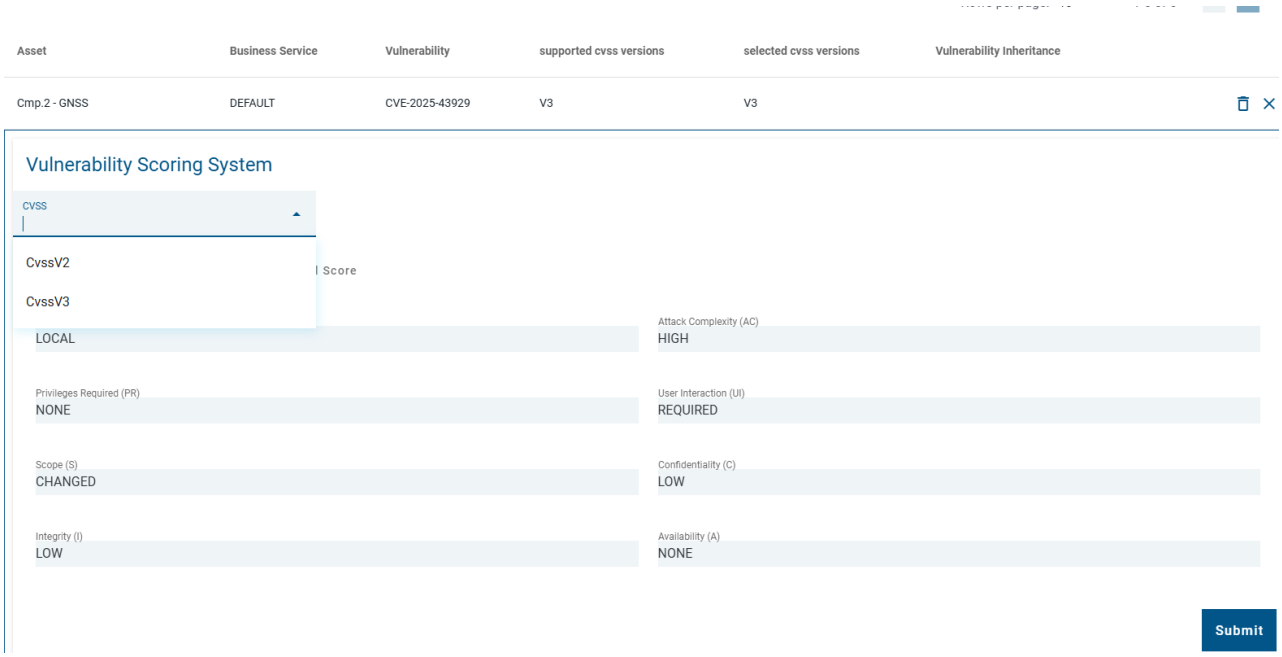


Figure 8.4: Extendability of vulnerability profiles

can be characterized by its impact in four concrete dimensions, namely Financial, Operational, Safety, and Privacy impact. For each risk assessment process, Security Administrators are able to select the target dimension of the impact of each damage scenario to be included in the risk quantification equation.

As part of the second release - and given that all the interfaces are already specified and implemented in the first release - enhancements are made in the various graphical user interface to accommodate the different pieces of information related to damage scenarios. Figure 8.5 showcases the creation of a new damage scenario in the CONNECT RA Engine, while Figure 8.6 illustrates the availability of additional information per damage scenario such as the associated threat scenarios.

Attack Path Calculator

One of the most challenging tasks as part of the TARA threat analysis is the specification of all the possible attack paths that can lead to a threat scenario in the target CCAM function that could eventually result in a damage scenario for the entire vehicle or its passengers. The Attack Path Calculator component aims to relax part of the burden of this non trivial task. Based on the available knowledge for each asset, and given the inter-dependencies across the target topology, it is able to provide a set of possible attack paths from the already identified vulnerabilities that are associated with each asset.

The Attack Path Calculator component, implemented as an expert system using the Drools engine, analyzes the characteristics of vulnerabilities associated with each asset. Leveraging a predefined set of rules that specify the conditions under which a vulnerability can be exploited to move from one asset to another, it identifies potential cascading attacks that an adversary could execute. Details on the design and implementation of this new component are presented in Section 8.4.

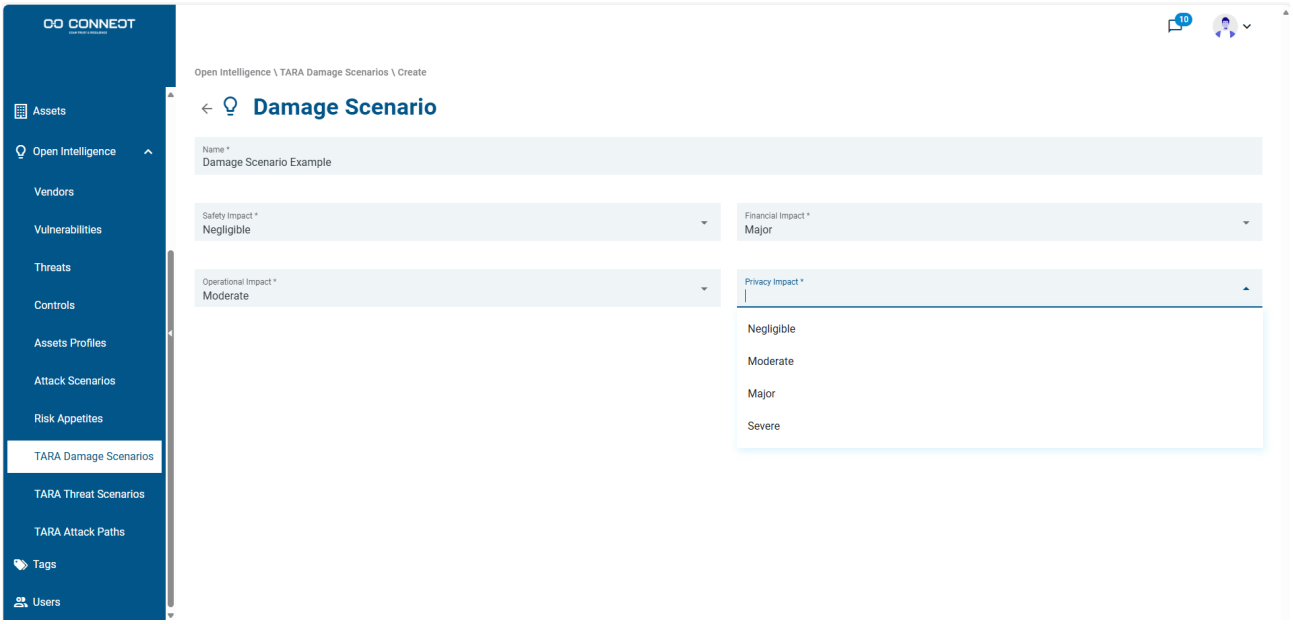


Figure 8.5: Creating a new damage scenario entry

Attack Feasibility Recommendation Engine

Once the attack paths are available, it is necessary to provide the means to identify their attack feasibility and eventually their overall risk. In the context of TARA, the specification of the attack feasibility rating for an attack path is based on five key parameters that can be specified by the Security Admin as illustrated in Figure 8.7. On another note, when it comes to the overall risk level of the identified attack path, a new equation is defined based on the Individual Risk Level (IRL) equation specified in [5]. Specifically, the Cumulative Risk Level (CRL) of an attack path is identified as:

$$CRL(Asset_1, Asset_i, \ldots, Asset_n) = IRL(Asset_1) \times IRL(Asset_i) \times \ldots \times IRL(Asset_n)$$

$$\begin{aligned} \text{where} \quad IRL(Asset_i) &= IRL(Asset_1, Vulnerability_j, Threat_k) \\ &= ThreatLevel \times VulnerabilityImpact \end{aligned}$$

The Attack Feasibility Recommendation Engine is implemented as part of the Attack Path Calculator and the implementation details are presented in Section 8.4.

Risk Quantification Engine

The risk quantification engine is responsible for consuming the threat analysis performed either automatically or manually by the security administrators and compute risk assessment risk values for the monitored asset topology. This component is designed in a modular fashion to allow the inclusion of diverse risk assessment methodologies. Prior to the second version of the CONNECT RA Engine, both the CVSS-based and the TARA methodologies were implemented. From that point onward, the goal of the risk quantification engine is to provide the necessary reporting capabilities so that it allows the derivation of the RTL values defined in Chapter 10.

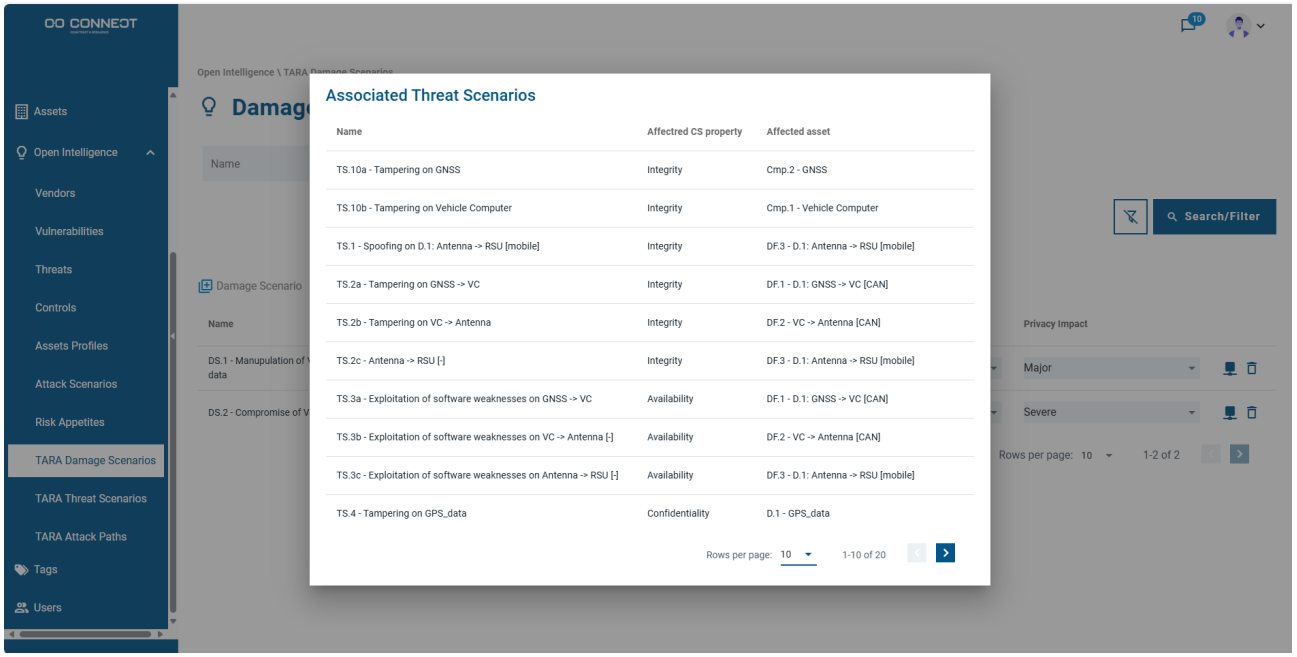


Figure 8.6: Displaying the threat scenarios associated with a particular damage scenario

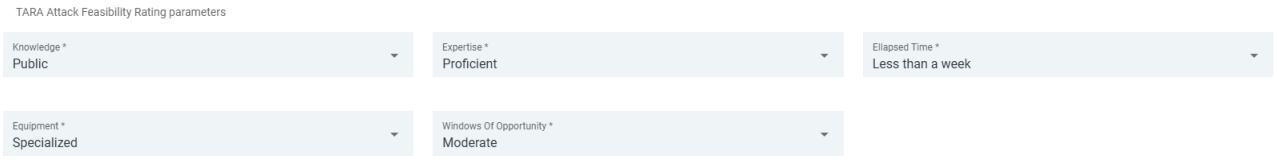


Figure 8.7: Attack feasibility rating parameters

In the context of the second release, the goal is to showcase how the available information from the TARA risk assessment (iterative) process can be leveraged to calculate both the belief (Section 10.1.1) and the disbelief (Section 10.1.2) equations. Nevertheless, the CONNECT RA Engine can be easily extended to include the necessary information for enabling the calculations of the Uncertainty threshold (Section 10.1.3). Overall, Section 8.5 presents how the risk quantification engine results are produced and made available for the RTL calculations to take place. Subsequently, Section 8.7 discusses how the selection of different risk quantification equations may impact the resulted RTL constraints.

Mitigation Strategy Component

The Mitigation Strategy Component supports the modeling of all types of controls. It also allows security administrators to signal the effectiveness of each security control by configuring the various parameters that contribute to the overall risk level for an asset or a trust relationship. An initial version of this component is available in the second release of the CONNECT RA Engine. Upon selection of a control scenario (set of controls to be applied to the asset topology), the Security Admin is able specify the reduction of the likelihood and impact of the relevant threat scenarios. Eventually, this control scenario leads to the recalculation of the overall risk graph. For instance, the Security Admin can see the overall impact of a security control — such as the application of a software patch — on associated risk scenarios and, ultimately, on the corresponding risk values (Figure 8.8).

Figure 8.8: Impact of applying a security control (e.g., software patch applied in the in-vehicle computer) in the number of risk scenarios considered.

8.3.3 CONNECT RA Framework

Based on the second release of the CONNECT RA Engine, this section takes a deep dive into the flows of actions that are realized inside the CONNECT RA architecture. In [5] the presented sequence diagram focused solely on the realization of a single risk assessment instance. In what follows, we position the sequence of actions by always taking into consideration how the resulting outcomes enable the calculation of the RTL constraints. In brief, the sequence diagrams presented below showcase how a security administrator is able to run the CVSS-based methodology and use the inferred knowledge to enhance the threat analysis performed as part of the TARA risk framework. Eventually, as part of the risk treatment decision the security administrator may decide to enforce security controls for the risk scenarios that are not accepted and need to be mitigated. A mitigation strategy comprising of one or more security controls is called a control scenario. The enforcement of a control scenario may lead to a revised risk quantification process that eventually will minimize the identified risk to accepted levels.

By delving into the sequence of actions, Figure 8.9 presents the revised CVSS-based methodology, with the inclusion of the Attack Path Calculator component and the derivation of the CRL risk score for the identified attack paths. Specifically, once the Security Administrator has provided the necessary threat analysis (asset topology and relevant threats and vulnerabilities), they are able to initiate a new Request For Risk Assessment (RFRA). This triggers the risk quantification engine to fetch the component diagram from the asset modeling and visualization engine, and the relevant attack scenarios that allow the derivation of the IRL score. Specifically, from the vulnerabilities and their CVSS characteristics it is possible to derive the vulnerability impact ([5]) while the associated threats are assigned a threat likelihood factor that can be parameterized by the Security Administrator. Once the IRL scores are quantified per risk scenario on a target asset, the Attack Path Calculator is invoked to automatically infer the attack paths based on a set of rules that dictate the conditions under which a vulnerability can be exploited by an adversary to carry out a cascading attack that pivots from one asset to another (see Section 8.4). In each attack path, one vulnerability is considered at a time (e.g., there may be multiple attack paths that lead

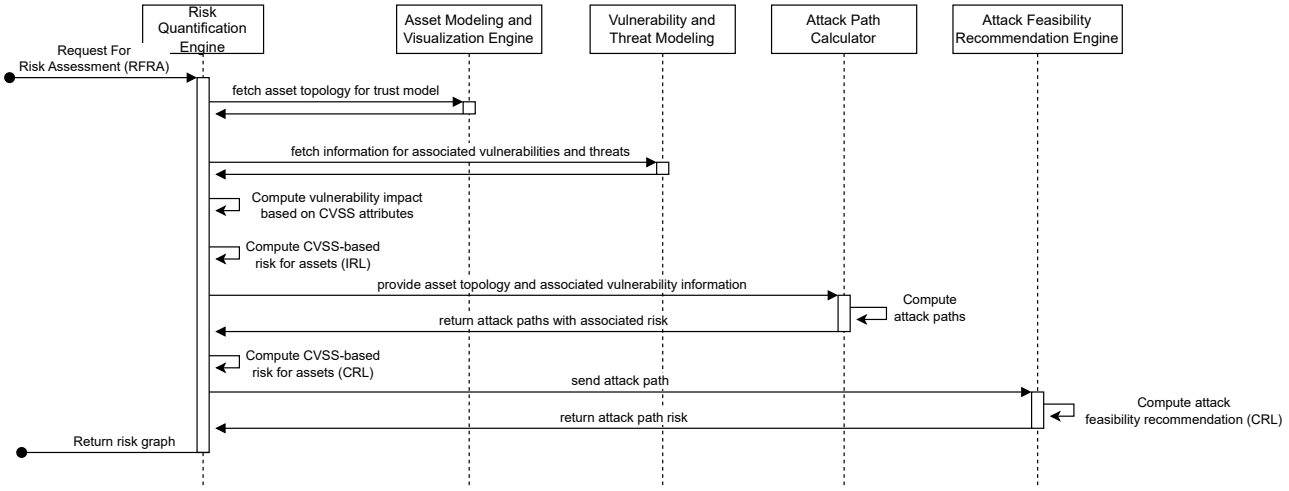


Figure 8.9: CVSS-based methodology with IRL and CRL calculations

to a cascading attack from an in-vehicle sensor to the OBU). Hence, it is possible for the Attack Feasibility Recommendation Engine to assign a CRL score for each attack path that is identified. All the collected risk results are finally aggregated and sent back to the Security Administrator.

The aforementioned CVSS-based methodology offers automated mechanisms that may facilitate the collection or even enhance the threat analysis required by the TARA framework. This is depicted in the first two steps of Figure 8.10. As part of this threat analysis, the Security Administrator provides the detailed specification of the attack steps for each attack path and identifies the Attack Feasibility Rating in the context of TARA. Subsequently, the Security Administrator initiates a TARA risk assessment process to perform their risk analysis as specified in [1]. As this may happen multiple times throughout the lifespan of the target environment, we define the term ”TARA iteration” which describes the single execution of the TARA steps. After specifying the target component diagram characterizing the CCAM function under evaluation, the Security Administrator specifies the relevant damage scenarios and their associated impact through the Damage Scenario Management component. With all this threat analysis, the Risk Quantification Engine is able to quantify the risk based on the equations defined and implemented in [5]. The risk scenarios are then made available to the Security Administrator. In the context of the risk treatment decision process, it is necessary to decide the management strategy of the identified risk scenarios: Accept, Avoid, Reduce, Transfer. When deciding to reduce the risk of one or more risk scenarios, it is necessary that the Security Administrator specifies the set of controls - i.e., control scenario - that enables the mitigation actions. The impact of an enforced control scenario is typically reflected in a reduced attack feasibility or the complete elimination of previously identified attack paths. However, it is important to note that introducing a control scenario may also lead to new attack vectors within the system. Additionally, the simultaneous enforcement of multiple controls — when part of a broader control scenario — can result in diminished effectiveness due to dependencies or conflicts between them. To account for these complexities, the Security Administrator may need to re-run the TARA process iteratively until the residual risks are reduced to an acceptable level.

Once the Security Administrator reaches to the aforementioned stable state where the risk of the system has arrived to an accepted level, it is possible to proceed with the RTL calculations of Chapter 10. Specifically, as also shown in Table 8.17 below, the Risk Quantification Engine exposes an API to query for risk scenarios and their associated risk values across different TARA iterations where different control scenarios are taken into consideration. This enables the RTL

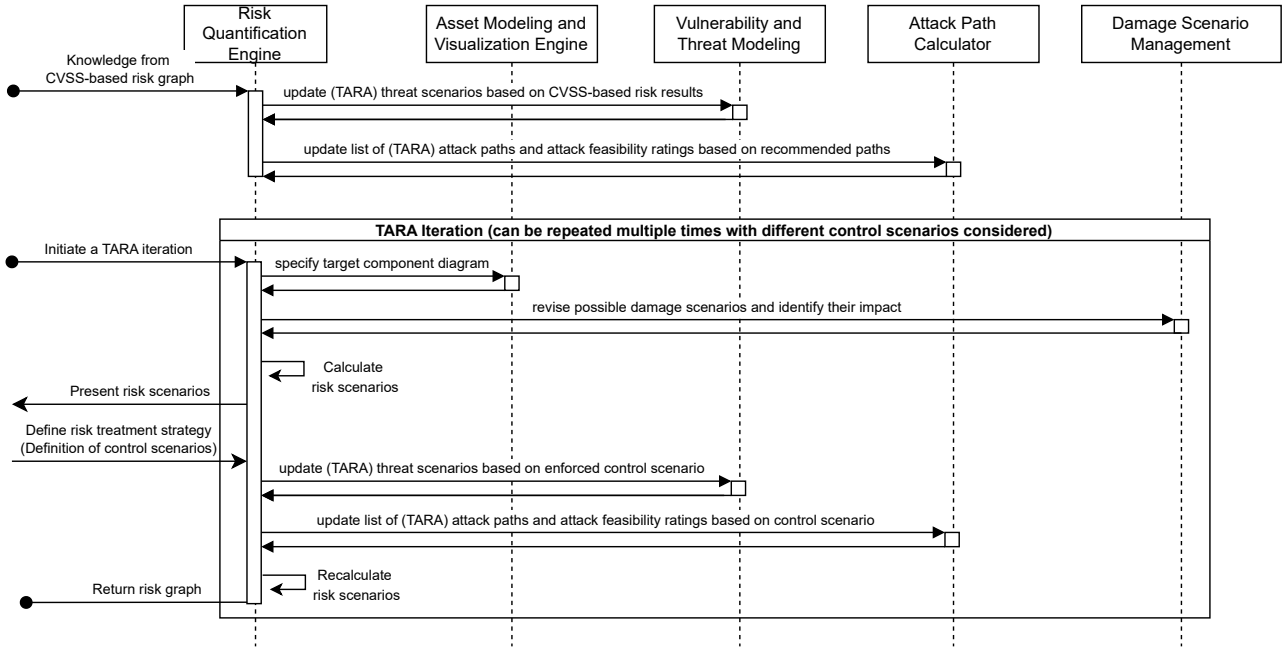


Figure 8.10: TARA iterative process encompassing the enforcement of control scenarios in the risk quantification process

Calculator to identify the maximum residual risk within the system and subsequently derive RTL constraints based on belief and disbelief values. To further illustrate the querying capabilities that unlock the RTL calculations, Section 8.5 presents an indicative example of two consecutive TARA iterations showcasing the filtering capabilities that result in RTL constraints for a specific context (i.e., security property and target trust object).

8.4 Streamlining TARA with Automated Attack Path Calculation

This Attack Path Calculator sub-component enhances the CONNECT RA engine capabilities with respect to analyzing the risk for a particular asset through the identification of possible attack paths that may target that asset. The attack path calculation analysis does not require any additional information apart from the one used in scope of the risk assessment analysis. Specifically, it takes into consideration the asset topology (assets and their relationships) as well as the vulnerabilities that are associated with each asset. As presented in this section, with this information the attack path calculation component can derive a set of attack paths that allow an attacker to infiltrate the infrastructure and hop from one asset to another by exploiting one associated vulnerability at a time.

Obviously, not all vulnerabilities give the power to an adversary to pivot from one asset to another. Thus, it is important that a security administrator defines the rules that dictate the conditions based on which a vulnerability enables a malicious user to hop between assets. These rules can be then fed to an expert system which will combine the rules with the existing topology information and derive the identified attack paths. This analysis is implemented using the Drools engine [15], a business-rule management system with a forward-chaining and backward-chaining inference-based rules engine, allowing fast and reliable evaluation of business rules and complex event

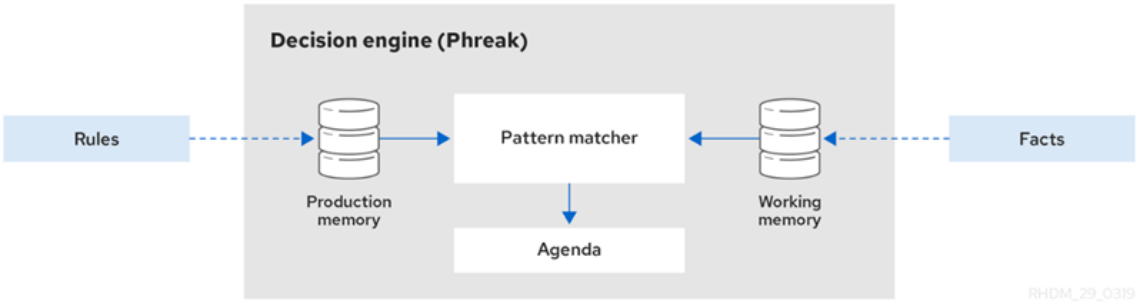


Figure 8.11: The Drools Engine internal architecture

processing. As shown in Figure 8.11, Drools engine is fed with two types of information: rules and facts.

Within the context of the attack path calculation, the rules are static information that describe the conditions for an asset relationship be a valid link of an attack path. These rules are expressed in the Drools Rule Language (DRL), are stored in .drl files and can be configured by an administrator during the launch of the CONNECT RA engine. A snapshot of such a file is presented in Table 8.2. It uses the CVSS attributes to define the conditions of a valid attack path edge. In parallel, the facts are loaded on runtime and describe the asset topology information including the associated vulnerabilities per asset. With this information the Drools engine can run multiple rounds to identify the available attack paths. For each round, the Drools engine uses the input data to derive multiple intermediate facts that are fed as input to the subsequent round. When no additional fact is produced for a specific round, the Drools Engine terminates its execution, and the resulting facts are placed in the Drools Agenda. From there the Attack Path Calculator component can retrieve the calculated attack paths, store them in the NoSQL database and report them back to the user.

Rule Id	Rule Description	Rule expressed in DRL

1	Input Validation Rule	`rule "rule-ValidationMandatoryInputCheck"`
		`when`
		`not(exists(APAsset(entryPoint==true) && APAsset(target==true) && APAttacker()))`
		`then`
		`insert(new ExceptionStructural("At least one entry point one target point and an attacker profile should exist"));`
		`End`

Rule Id	Rule Descripion	Rule expressed in DRL
2	Rule for creating an attack path for vulnerabilities that require Network access	`rule "rule-PropagationFromEntryPointsRemote" when $assetto:APAsset(isEntryPoint() == true) $attacker:APAttacker() $vuln: APVulnerability(vector == "NETWORK" || vector == "ADJACENT_NETWORK", AttackerSkillType. isAttackerCapableBasedOnComplexity(complexity, $attacker.skill)) from $assetto.vulnerabilities not(exists(APPath(fromAssetId==APPath. REMOTE_ADVERSARY_ID, toAssetId==$assetto. getId() , vuln==$vuln))) then insert(new APPath($vuln, APPath. REMOTE_ADVERSARY_ID , $assetto.getId())); end`
3	Rule for creating an attack path for vulnerabilities that don't require Network access	`rule "rule-PropagationFromEntryPointsLocal" when $assetto: APAsset(isEntryPoint() ==true) $attacker: APAttacker() $vuln: APVulnerability(vector=="LOCAL", AttackerSkillType. isAttackerCapableBasedOnComplexity(complexity, $attacker.skill)) from $assetto.vulnerabilities not(exists(APPath(fromAssetId==APPath. LOCAL_ADVERSARY_ID, toAssetId==$assetto. getId(), vuln==$vuln))) then insert(new APPath($vuln, APPath. LOCAL_ADVERSARY_ID , $assetto.getId())); end`

Rule Id	Rule Descripion	Rule expressed in DRL

4	Rule for expanding an attack path for vulnerabilities that require Network access	`rule "rule-PropagationFromPivotingLocal"`
		`when`
		`$path: APPath($assetfromid: toAssetId)`
		`$assetfrom: APAsset(id==$assetfromid)`
		`$dependency: APDependency(fromAssetId==`
		`$assetfrom.id,`
		`$assettargetid: toAssetId)`
		`$assetto:APAsset(id==$assettargetid)`
		`$attacker:APAttacker()`
		`$vuln: APVulnerability(vector == "LOCAL",`
		`AttackerSkillType.`
		`isAttackerCapableBasedOnComplexity(`
		`complexity, $attacker.skill)) from`
		`$assetto.getVulnerabilities()`
		`not(exists(APPath(fromAssetId==$assetfrom.`
		`getId(), toAssetId==$assetto.getId(), vuln`
		`==$vuln)))`
		`then`
		`insert(new APPath($vuln, $assetfrom.getId() ,`
		`$assetto.getId()));`
		`end`

5	Rule for expanding an attack path for vulnerabilities that don't require Network access	`rule "rule-PropagationFromPivotingRemote"`
		`when`
		`$path: APPath($assetfromid:toAssetId)`
		`$assetfrom: APAsset(id==$assetfromid)`
		`$dependency: APDependency(fromAssetId==`
		`$assetfrom.id , $assettargetid:toAssetId)`
		`$assetto:APAsset(id==$assettargetid)`
		`$attacker:APAttacker()`
		`$vuln : APVulnerability(vector == "NETWORK"`
		`|| vector == "ADJACENT_NETWORK",`
		`AttackerSkillType.`
		`isAttackerCapableBasedOnComplexity(`
		`complexity, $attacker.skill)) from`
		`$assetto.getVulnerabilities()`
		`not(exists(APPath(fromAssetId==$assetfrom.`
		`getId(), toAssetId==$assetto.getId(), vuln`
		`==$vuln)))`
		`then`
		`insert(new APPath($vuln, $assetfrom.getId() ,`
		`$assetto.getId()));`
		`end`

Rule Id	Rule Descripion	Rule expressed in DRL

6	Rule for obtaining the final attack chain	`rule "rule-ChainGeneration-EntryPoint"`
		`when`
		`$path:APPath(fromAssetId==APPath.REMOTE_ADVERSARY_ID)`
		`not(exists(APChain(getEntryPointPath()==$path)))`
		`then`
		`insert(new APChain($path));`
		`end`

Table 8.2: Rules for obtaining attack chains, expressed in DRL format

Apart from managing the Drools sessions for calculating the attack paths of the topology, the Attack Path Calculator component is responsible for managing the entire lifecycle of the attack path assessment tasks within the CONNECT RA engine. It is worth mentioning that during the calculations of the Drools engine, one key parameter that is taken into account is the attacker's capabilities (Figure 8.12). This is a parameter that allows the Drools engine to decide whether a vulnerability can be exploited based on the attacker's skills that are quantified in a 5-tier scale: Very Low, Low, Medium, High, Very High. Each attack path corresponds to a different combination of vulnerabilities that are used by an adversary to pivot through these assets. Finally, in scope of the Attack Path Assessment, a risk value is assigned to each one of these attack paths (Figure 8.13), following the rules defined in Table 8.2.

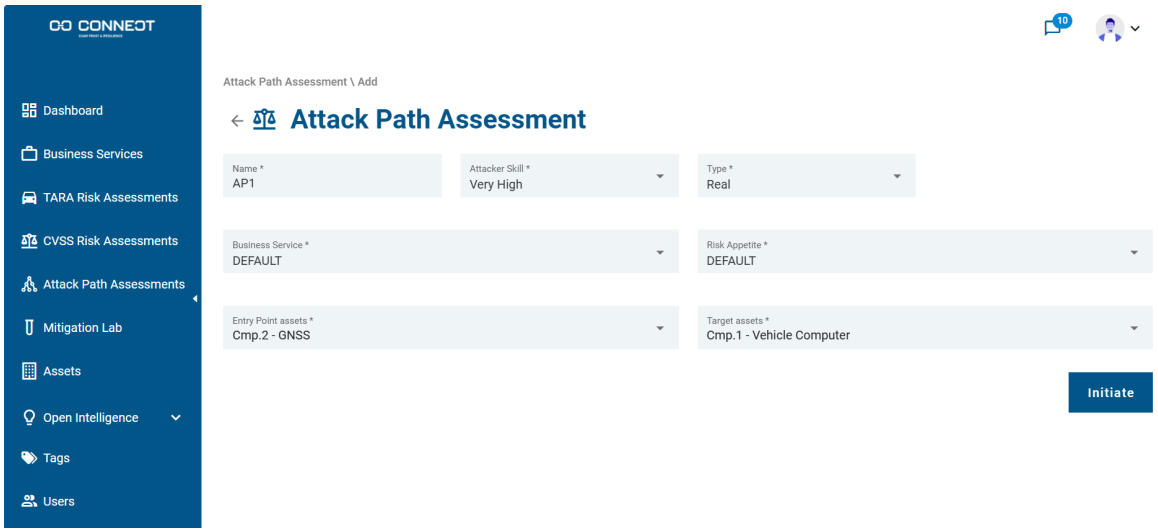


Figure 8.12: Initializing a new Attack Path Assessment

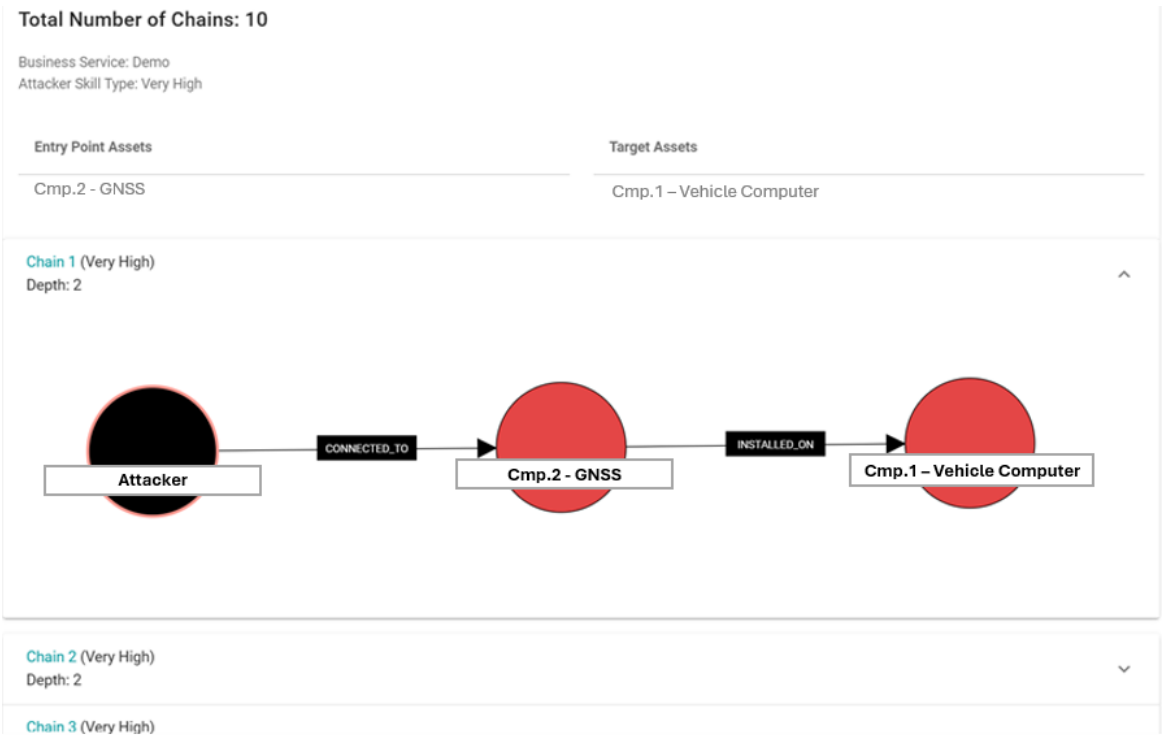


Figure 8.13: Visualizing results of Attack Path Assessment

8.5 Transitioning from TARA to RTL

Section 8.3.3 presents the systematic operations of the CONNECT RA Engine through detailed sequence diagrams. To better illustrate the availability of risk-related information that supports RTL calculations, this section showcases the TARA process from the perspective of a Security Administrator interacting with the CONNECT RA graphical user interface. For this purpose, we adopt the illustrative example presented in [5]. This example involves a simplified in-vehicle topology representing the vehicle function of receiving GNSS data and making it accessible to the vehicle's onboard computer.

Following also the steps of the sequence diagram in Figure 8.10, the Security Administrator logs in the CONNECT RA Engine and specifies the component diagram corresponding to the GNSS data transmission item function. Figure 8.14 presents the visualization of all the assets and their interdependencies that characterize the target CCAM function. As can be observed, in this component diagram both software, hardware and data flow entities are defined, allowing for a fine-grained identification of attack paths.

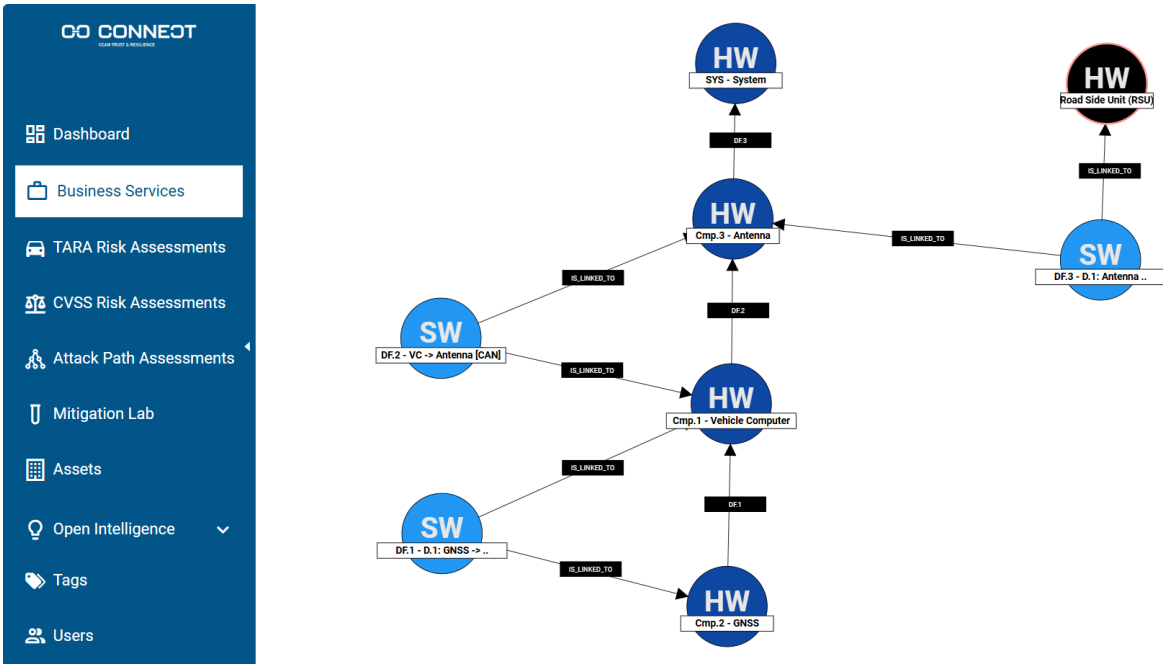


Figure 8.14: Step 1: Specify target component diagram

Secondly, navigation to the ”TARA Damage Scenario” tab (left sidebar), allows the specification of the relevant damage scenarios that are associated with this item function. From Figure 8.15 it is clear that the Security Administrator is able to specify the name of each damage scenario and the corresponding impact in terms of Safety, Economical, Operational and Privacy aspects.

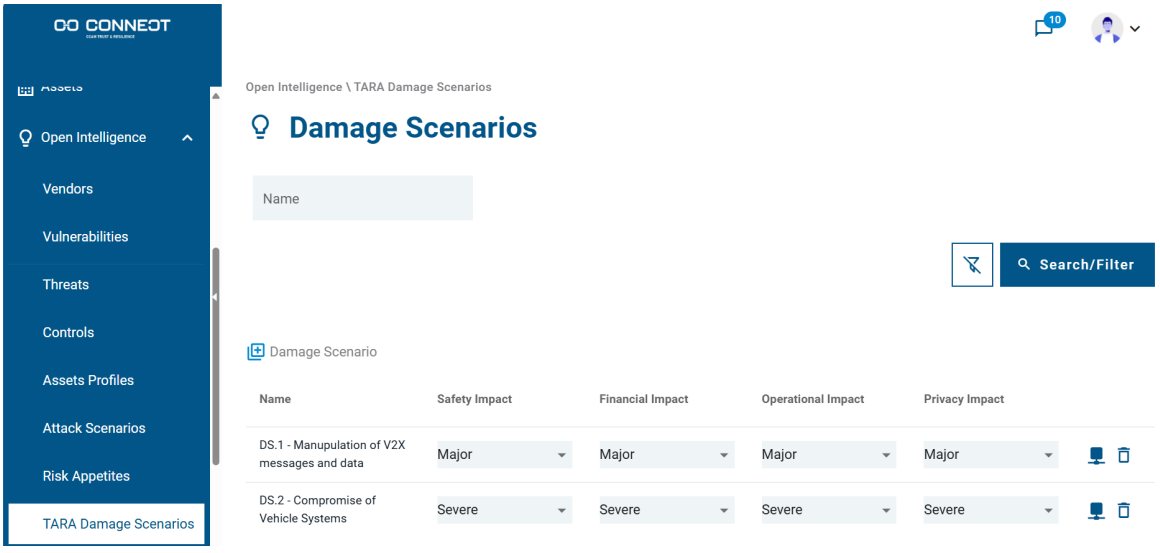


Figure 8.15: Step 2: Specify TARA damage scenarios

Subsequently, the Security Administrator is able to identify the concrete threat scenarios that can affect any of the assets in the component diagram. Specifically, in Figure 8.16 it is clear that each threat scenario can be associated with a particular asset (e.g., Tampering on GNSS sensor or Spoofing on the data flow from the sensor to the vehicle computer) as well as the target cyber-security property that it compromises. Eventually, this association is intrinsically linked with the threat scenarios that affect a specific trust property. Eventually, only the relevant threat scenarios

will contribute to the calculations of the corresponding RTL values for a particular trust property as presented in Chapter 10.

Figure 8.16: Step 3: Specify TARA threat scenarios

In the forth step, Figure 8.17 it is demonstrated how to define the attack paths for the TARA risk assessment process. Each path may consist of multiple intermediate steps that help realize a more complex and cascading attack scenario. The attack feasibility of each scenario is reflected in the reporting of the attack scenarios and can be dynamically configured from the five values specified in [1], namely:

- Knowledge
- Expertise
- Elapsed Time
- Equipment
- Window Of Opportunity

From the second right most button in Figure 8.17, the Security Administrator is able to inspect all the intermediate steps that are necessary for a particular attack path and evaluate the intermediate attack feasibility ratings per attack step.

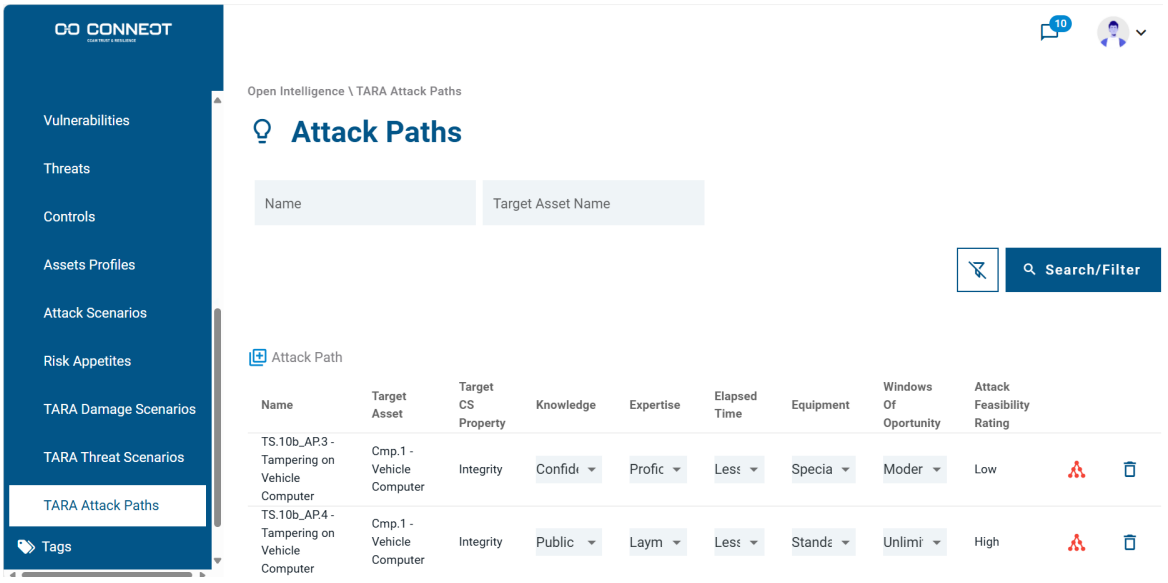


Figure 8.17: Step 4: Specify TARA attack paths

Before performing the risk quantification process, it is also possible to define the number of security controls to be considered in the risk assessment iterations. Figure 8.18 showcases the ”Controls” tab, where users are able to manage the list of available controls and associate them with specific attack paths that aim to mitigate.

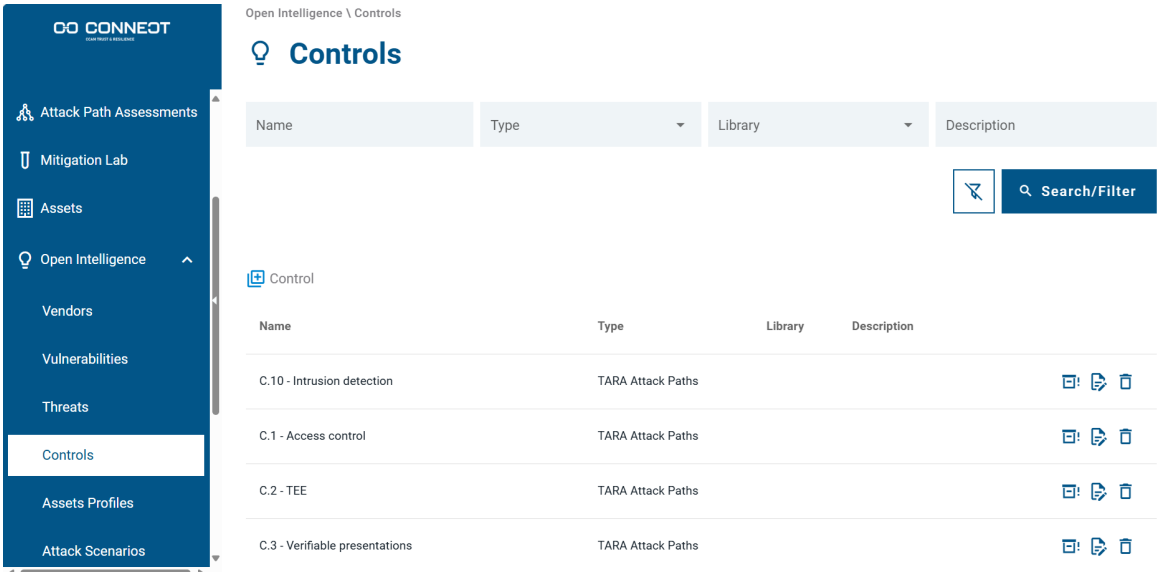


Figure 8.18: Step 5: Specify TARA security controls

In the first iteration that is showcases in Figure 8.19, the Security Administrator calculates the baseline risk; that is the risk scenarios and their associated scores assuming no security control in place. At this step the Security Administator has concluded the first iteration of the TARA risk assessment process by deciding the risk treatment per risk scenario.

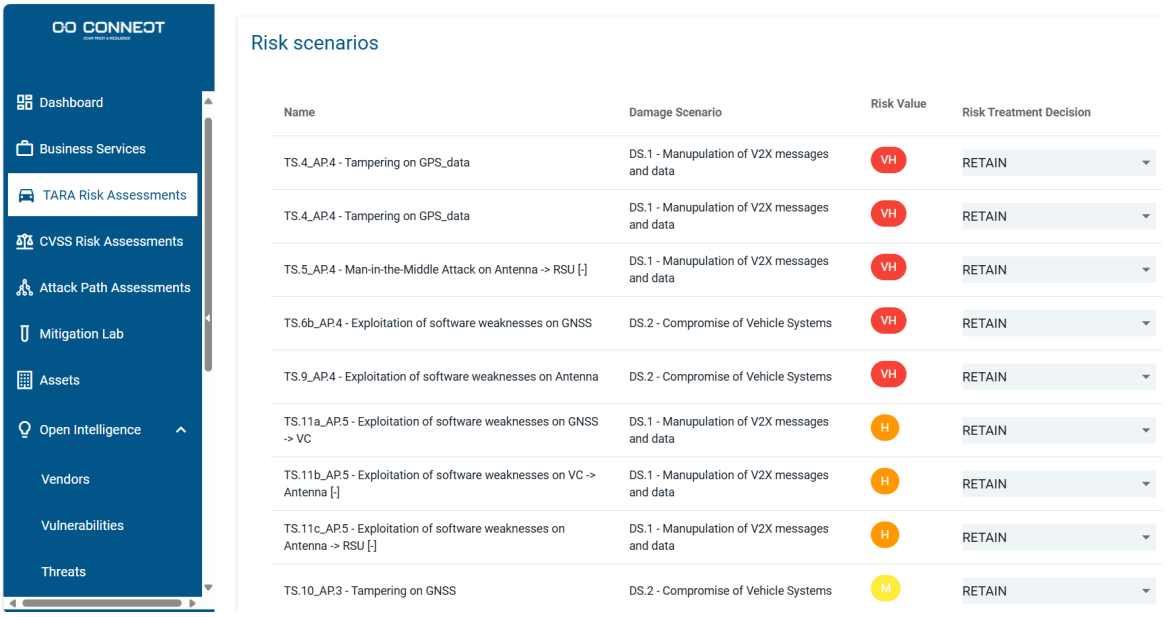


Figure 8.19: Step 6: Results of a risk assessment with no security controls enforced

Figure 8.20 showcases the management panel of each iteration where the users can instantiate new iterations - by taking into consideration different control scenarios - and monitor the progress of the tasks. In this example, it is clear that the second iteration refers to a control scenario where access control mechanisms are in place.

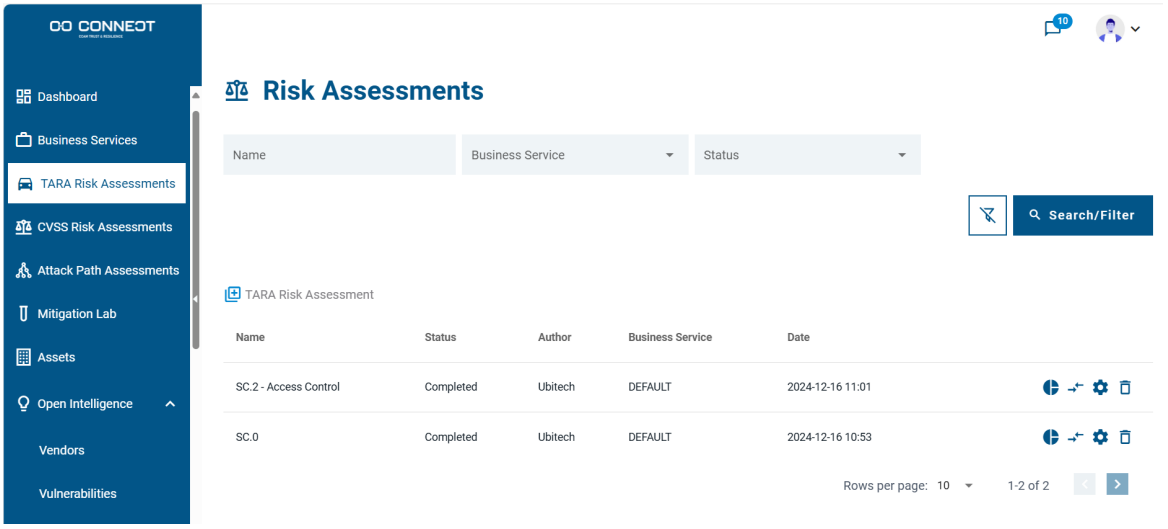


Figure 8.20: Step 7: List of risk assessment tasks: One with no controls applied and one with a single security control (Access control mechanisms enforced).

The availability of multiple TARA iterations enables the comparison of the different control scenarios. Specifically, as depicted in Figure 8.21, this comparison helps the Security Administrators to determine the impact of each control scenario in the overall risk posture of the topology. As can be seen from the figure, this panel offers a set of filtering capabilities which allows for narrowing down those risk scenarios that are relevant for a particular context. This is crucial both for the derivation of the RTL constraints but also for the optimal selection of the weights to be specified as part of the ATL calculations (see Chapter 9). For instance, assuming that one wants to derive

the RTL constraints pertaining to the in-vehicle integrity, the filtering options would allow the three first risk scenarios but exclude the impact of the Antenna - Road Side Unit risk scenario as it is not in the scope of the trust proposition to be evaluated.

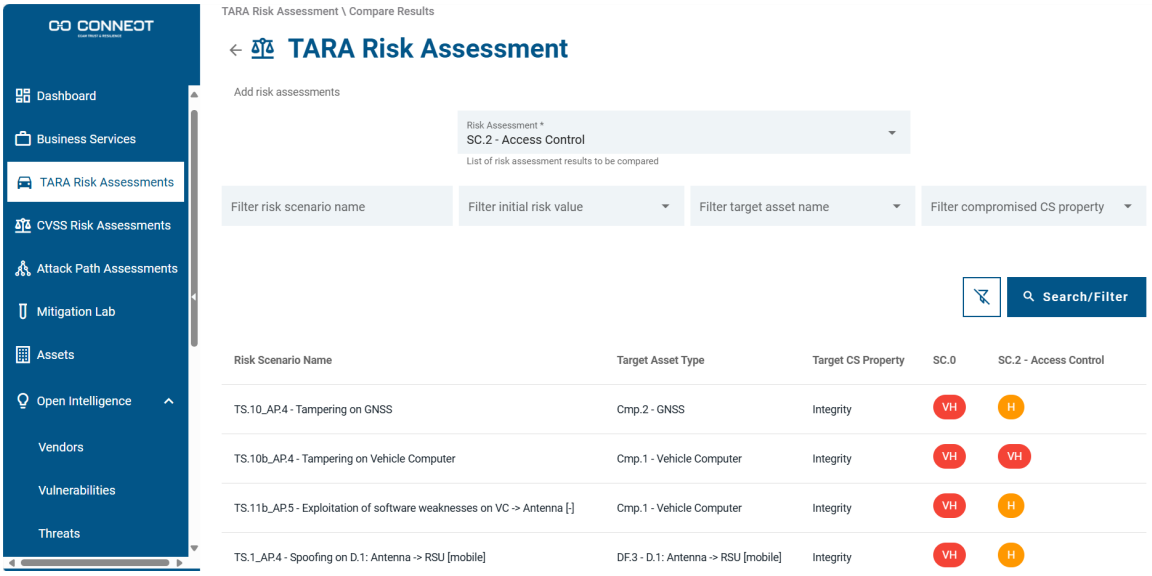


Figure 8.21: Step 8: Comparison of results from different risk assessment tasks per risk scenario.

8.6 Specification of Implemented Interfaces

This section presents the specification of the CONNECT RA interfaces that are implemented up to the second release. The interfaces presented prior to this release focus on the execution of concrete CVSS-based and TARA risk assessment processes. Specifically, As part of the first release, the CONNECT RA Engine supports all Create, Update, Read, Delete (CRUD) operations for the necessary data models (e.g., assets, threats, vulnerabilities, controls) and implements the risk quantification mechanisms specified in [5].

In this second release, additional interfaces are implemented to support the execution of the Attack Path Calculator functionalities and to support the necessary queries for the RTL Calculator to extract the corresponding risk scenarios that are relevant for a specific context. The tables presented below summarize all the implemented endpoints that are implemented by the CONNECT RA Engine. Tables 8.3 - 8.16 refer to the interfaces initially introduced in the first release, while Tables 8.17 - 8.19 document the latest interfaces developed in release v2.0.0.

CONNECT RA	Interface Technical Details	
Type of Interface	REST HTTP POST	
Purpose	Addition of an process - /api/v1/processes	
Inputs & Outputs	Inputs	Outputs
	Process name	Id of the process as persisted in the *CONNECT* RA database.

Interaction with components	1. Asset modeling and visualization component 2. Risk Quantification Engine, 3. OLISTIC Backend, 4. OLISTIC Frontend, 5. External OEM integration activities

Table 8.3: Add a process in *CONNECT* RA

CONNECT RA	Interface Technical Details	
Type of Interface	REST HTTP POST	
Purpose	Addition of an asset - /api/v1/assets	
Inputs & Outputs	Inputs	Outputs
	1. Asset name 2. Attributes (key-value pairs) 3. Relationships (i.e., interdependencies with other assets, data flows, participating functions) 4. Linked cybersecurity properties	1. Id of the asset as persisted in the *CONNECT* RA database.
Interaction with components	1. Asset modeling and visualization component 2. Risk Quantification Engine, 3. OLISTIC Backend, 4. OLISTIC Frontend, 5. External OEM integration activities	

Table 8.4: Add an asset in *CONNECT* RA

CONNECT RA	Interface Technical Details	
Type of Interface	REST HTTP POST	
Purpose	Addition of a vulnerability - /api/v1/vulnerabilities	
Inputs & Outputs	Inputs	Outputs
	1. Vulnerability name, 2. Vulnerability description, 3. Vulnerability library, 4. Published date, 5. (Optional) CVSS attributes (v2.0/v3.1)	1. Id of the vulnerability as persisted in the *CONNECT* RA database.
Interaction with components	1. Vulnerability and Threat modeling component 2. Risk Quantification Engine, 3. OLISTIC Backend, 4. OLISTIC Frontend, 5. External OEM integration activities	

Table 8.5: Add a vulnerability in *CONNECT* RA

CONNECT RA	Interface Technical Details	
Type of Interface	REST HTTP POST	
Purpose	Addition of a threat - /api/v1/threats	
Inputs & Outputs	Inputs	Outputs
	1. Threat name, 2. Threat description, 3. Threat library, 4. Published date	1. Id of the threat as persisted in the *CONNECT* RA database.
Interaction with components	1. Vulnerability and Threat modeling component 2. Risk Quantification Engine, 3. OLISTIC Backend, 4. OLISTIC Frontend, 5. External OEM integration activities	

Table 8.6: Add a threat in *CONNECT* RA

CONNECT RA	Interface Technical Details	
Type of Interface	REST HTTP POST	
Purpose	Addition of a mitigation control - /api/v1/controls	
Inputs & Outputs	Inputs	Outputs
	1. Control name, 2. Control description, 3. Control library, 4. Published date, 5. Attributes	1. Id of the control as persisted in the *CONNECT* RA database.
Interaction with components	1. Mitigation Strategies component 2. Risk Quantification Engine, 3. OLISTIC Backend, 4. OLISTIC Frontend, 5. External OEM integration activities, 6. TAM	

Table 8.7: Add a control in *CONNECT* RA

CONNECT RA	Interface Technical Details	
Type of Interface	REST HTTP POST	
Purpose	Addition of a TARA damage scenario - /api/v1/tara/damage-scenarios	
Inputs & Outputs	Inputs	Outputs

	1. Damage scenario name, 2. Safety impact, 3. Financial impact, 4. Operational impact, 5. Privacy impact, 6. Overall impact	1. Id of the TARA damage scenario as persisted in the *CONNECT* RA database.
Interaction with components	1. Risk Quantification Engine, 2. OLISTIC Backend, 3. OLISTIC Frontend, 4. External OEM integration activities, 5. TAM	

Table 8.8: Add a TARA Damage Scenario in *CONNECT* RA

CONNECT RA	Interface Technical Details	
Type of Interface	REST HTTP POST	
Purpose	Addition of a TARA attack path - /api/v1/tara/attack-paths	
Inputs & Outputs	Inputs	Outputs
	1. Attack path name, 2. Chain of intermediate asset id forming the attack path, 3. Target asset id, 4. Target Cybersecurity property, 5. Attack path Elapsed Time, 6. Attack path Expertise, 7. Attack path Knowledge, 8. Attack path Windows of opportunity, 9. Attack path Equipment, 10. Attack path Attack Feasibility Rating	1. Id of the TARA attack path as persisted in the *CONNECT* RA database.
Interaction with components	1. Risk Quantification Engine, 2. OLISTIC Backend, 3. OLISTIC Frontend, 4. External OEM integration activities, 5. TAM	

Table 8.9: Add a TARA Attack Path in *CONNECT* RA

CONNECT RA	Interface Technical Details	
Type of Interface	REST HTTP POST	
Purpose	Addition of a TARA threat scenario - /api/v1/tara/threat-scenarios	
Inputs & Outputs	Inputs	Outputs

	1. Threat scenario name, 2. Related (target) asset id, 3. Related process id, 4. Related cybersecurity property	1. Id of the TARA threat scenario as persisted in the *CONNECT* RA database.
Interaction with components	1. Risk Quantification Engine, 2. OLISTIC Backend, 3. OLISTIC Frontend, 4. External OEM integration activities, 5. TAM	

Table 8.10: Add a TARA Threat Scenario in *CONNECT* RA

CONNECT RA	Interface Technical Details	
Type of Interface	REST HTTP POST	
Purpose	Creation of a TARA risk assessment task - /api/v1/tara/risk-assessments	
Inputs & Outputs	Inputs	Outputs
	1. Risk assessment name, 2. Related process id	1. Id of the TARA risk assessment task as persisted in the *CONNECT* RA database.
Interaction with components	1. Risk Quantification Engine, 2. OLISTIC Backend, 3. OLISTIC Frontend, 4. External OEM integration activities, 5. TAM	

Table 8.11: Add a Risk Assessment task in *CONNECT* RA

CONNECT RA	Interface Technical Details	
Type of Interface	REST HTTP POST	
Purpose	Adjustment of a TARA attack path inside a risk assessment task - /api/v1/tara/attack-paths/{id}/attack-path-profiles	
Inputs & Outputs	Inputs	Outputs

	1. TARA Attack path id within a risk assessment task, 2. Chain of intermediate asset id forming the attack path, 3. Target asset id, 4. Target Cybersecurity property, 5. Attack path Elapsed Time, 6. Attack path Expertise, 7. Attack path Knowledge, 8. Attack path Windows of opportunity, 9. Attack path Equipment, 10. Attack path Attack Feasibility Rating	1. HTTP OK
Interaction with components	1. Risk Quantification Engine, 2. OLISTIC Backend, 3. OLISTIC Frontend, 4. External OEM integration activities, 5. TAM	

Table 8.12: Update TARA Attack Path in a Risk Assessment task in *CONNECT* RA

CONNECT RA	Interface Technical Details	
Type of Interface	REST HTTP POST	
Purpose	Adjustment of damage scenario within a risk assessment - /api/v1/tara/risk-assessments/{raId}/damageScenario	
Inputs & Outputs	Inputs	Outputs
	1. Risk assessment id, 2. Damage scenario id, 3. Safety impact, 4. Financial impact, 5. Operational impact, 6. Privacy impact, 7. Overall impact	1. HTTP OK
Interaction with components	1. Risk Quantification Engine, 2. OLISTIC Backend, 3. OLISTIC Frontend, 4. External OEM integration activities, 5. TAM	

Table 8.13: Update TARA Damage Scenario in a Risk Assessment task in *CONNECT* RA

CONNECT RA	Interface Technical Details
Type of Interface	REST HTTP POST

Purpose	Inclusion of a control in a risk assessment task - /api/v1/tara/risk-assessments/{raId}/damageScenario	
Inputs & Outputs	Inputs	Outputs
	1. Risk assessment id, 2. Control id	1. HTTP OK
Interaction with components	1. Risk Quantification Engine, 2. OLISTIC Backend, 3. OLISTIC Frontend, 4. External OEM integration activities, 5. TAM	

Table 8.14: Update control in a Risk Assessment task in *CONNECT* RA

CONNECT RA	Interface Technical Details	
Type of Interface	REST HTTP POST	
Purpose	Execute a risk assessment task - /api/v1/tara/risk-assessments/{raId}/execute	
Inputs & Outputs	Inputs	Outputs
	1. Risk Assessment id	1. HTTP OK
Interaction with components	1. Risk Quantification Engine, 2. OLISTIC Backend, 3. OLISTIC Frontend, 4. External OEM integration activities, 5. TAM	

Table 8.15: Execute a Risk Assessment in *CONNECT* RA

CONNECT RA	Interface Technical Details	
Type of Interface	REST HTTP GET	
Purpose	Query for TARA risk results - /api/v1/tara/risk-assessments/102/risk-scenarios	
Inputs & Outputs	Inputs	Outputs
	1. Risk assessment id 2. Query parameter 1: Attack path name 3. Query parameter 2: Damage scenario name 4. Query parameter 3: Risk level 5. Query parameter 4: Risk assessment name	Array list where each entry is a risk scenario, containing the following: 1. Risk assessment id 2. Attack path id 3. Attack path name 4. Attack path target asset id 5. Attack path target cybersecurity property 6. Attack path feasibility rating 7. Damage scenario id 8. Damage scenario name 9. Damage scenario overall impact 10. Risk scenario level

Interaction with components	1. Risk Quantification Engine, 2. OLISTIC Backend, 3. OLISTIC Frontend, 4. External OEM integration activities 5. TAM

Table 8.16: Get Risk Assessment results in *CONNECT* RA

CONNECT RA	Interface Technical Details	
Type of Interface	REST HTTP GET	
Purpose	Query for TARA comparison results per risk scenario - /api/v1/tara/risk-assessments/102/compare-results	
Inputs & Outputs	Inputs	Outputs
	1. Base Risk assessment id (i.e., with no controls in place) 2. Query parameter 1: List of risk assessment ids 3. Query parameter 2: Asset name 4. Query parameter 3: Cybersecurity property 5. Query parameter 4: Damage Scenario name 6. Query parameter 4: Initial Risk Level	Array list where each entry is a risk scenario, containing the following: 1. Risk assessment id 2. Attack path id 3. Attack path name 4. Attack path target asset id 5. Attack path target cybersecurity property 6. Attack path feasibility rating 7. Damage scenario id 8. Damage scenario name 9. Damage scenario overall impact 10. Risk level per risk assessment id
Interaction with components	1. Risk Quantification Engine, 2. OLISTIC Backend, 3. OLISTIC Frontend, 4. External OEM integration activities 5. TAM	

Table 8.17: Compare Risk Assessment results in *CONNECT* RA on a risk-scenario level

CONNECT RA	Interface Technical Details	
Type of Interface	REST HTTP POST	
Purpose	Execute an attack path assessment task - /api/v1/attack-path-assessment/{apId}/execute	
Inputs & Outputs	Inputs	Outputs
	1. Attack Path Assessment id	1. HTTP OK

Interaction with components	1. Risk Quantification Engine, 2. Attack Path Calculator, 3. OLISTIC Backend, 4. OLISTIC Frontend, 5. External OEM integration activities, 6. TAM

Table 8.18: Execute an Attack Path Calculator task in *CONNECT* RA

CONNECT RA	Interface Technical Details	
Type of Interface	REST HTTP GET	
Purpose	Query for TARA risk results - /api/v1/tara/attack-path-assessments/103/visualize	
Inputs & Outputs	Inputs	Outputs
	1. Attack path assessment id	Array list where each entry is a attack path, containing the following: 1. Attack path assessment id 2. Attack path id 3. Attack path name 4. Attacker skill level 5. Attack step id (Asset id and vulnerability id) 6. Overall CRL for the identified path
Interaction with components	1. Risk Quantification Engine, 2. Attack Path Calculator, 3. OLISTIC Backend, 4. OLISTIC Frontend, 5. External OEM integration activities 6. TAM	

Table 8.19: Get Attack Path Assessment results in *CONNECT* RA

8.7 Reinforcing RTL calculation by abstracting the underlying risk quantification engine

In this second release of the CONNECT RA Engine we focused on the enabling factors revolving around the realization of the RTL constraints. Moving towards the second release of the integrated CONNECT Architecture, the goal is to evaluate the implemented features of the CONNECT RA Engine in the context of the use cases. This will provide key insights for the evaluation of the robustness and potential fine-tuning of the RTL equations.

As shown in the RTL belief and disbelief equations in Chapter 10, a crucial aspect that determines the final constraints is the quantification of risk derived from the TARA methodology. However, it is important to emphasize that the TARA specification does not mandates a single, universal risk quantification equation applicable across all domains and requirements. Instead, the risk

equation presented in this document and in [5] is considered an illustrative example, formulated within the context of the ISO/SAE 21434 standard, from which TARA originates.

Consequently, the performance evaluation of the second release of the CONNECT RA Engine in the context of [9] is envisioned in two core dimensions: i) experimentation setup based on a more complex component diagram describing a typical CCAM function as part of the in-vehicle topology, and ii) evaluation of different risk quantification methodologies, namely the baseline TARA risk quantification equation compared against the CVSS-based equation. These two dimensions aim to evaluate the robustness of the RTL equations and to assess how the chosen risk quantification methodology influences the strictness and accuracy of the RTL constraints.

Chapter 9

Algorithms for Trust Quantification

In the use cases of the CONNECT project, several trust sources are used. These trusted sources, for which a quantification algorithm is required, are: i) the Misbehavior Detection System (MBD), ii) the Attestation and Integrity Verification (AIV) and iii) the Trustworthiness Claims Handler (TCH). In the following, the algorithms are described, which take the evidence provided by these trust sources into account and calculate the corresponding trust opinion based on it.

9.1 Trust Quantification for MBD

The MBD system is running on the trustor node and consists of a fixed set of misbehavior detectors. The output of the detectors is forwarded to the TAF, regardless of whether the detectors have detected anything or not. Each misbehavior detector implements a binary detection pattern (0 = no detection, 1 = detection). The list of detectors used in this project is given below:

- Distance Plausibility (DP)
- Speed Plausibilty (SP)
- Speed Consistency (SC)
- Position Speed Consistency (PSC)
- Kalman Position Consistency (KPC)
- Kalman Position Speed Consistency (Speed) (KPCS)
- Kalman Position Speed Consistency (Position) (KPCP)
- Local Perception Verification (LPV)

9.1.1 Trust Quantification Approach for MBD

For the trust quantification, two weights are specified for each detector at design time. These weights are specified for the case that the detector has detected something and for the case that the detector has not detected anything. The reason why two weights are specified here for each detector is that, depending on the detector and whether something has been detected or not, we have more knowledge about the trustworthiness of the data. This should be reflected in the trust opinion of the corresponding data item. A proposal for the weights of the single detectors is shown below:

Table 9.1: Proposal for weights

Metric Name	Weight - No Detection	Weight - Detection
Distance Plausibility (DP)	1	2
Speed Plausibility (SP)	1	2
Speed Consistency (SC)	1	2
Position Speed Consistency (PSC)	1	2
Kalman Position Consistency (KPC)	2	2
Kalman Position Speed Consistency (Speed) (KPCS)	2	1
Kalman Position Speed Consistency (Position) (KPCP)	1	2
Local Perception Verification (LPV)	1	2

Based on the specified weights and an exponential function, the trust opinion is calculated. A proposal for the exponential function is shown below. However, other exponential functions are also conceivable here.

$$f(sumWeights) = -1.3^{-sumWeights} + 1$$

9.1.2 Example for Trust Calculation Based on MBD

In the following, an example of the calculation of the trustworthiness of an observation is provided. Let us assume, that the output of the detectors is the one presented in the second column of Table 9.2 (0: no detection, 1: detection). Using these detection results and the weights specified in Table 9.1, the corresponding weights for each detection are listed in the third column of Table 9.2.

Table 9.2: Metrics with Detection, Weight, Belief, and Disbelief

Metric	Detection	Weight	Weight Belief	Weight Disbelief
Distance Plausibility (DP)	0	1	1	-
Speed Plausibility (SP)	1	2	-	2
Speed Consistency (SC)	1	2	-	2
Position Speed Consistency (PSC)	1	2	-	2
Kalman Position Consistency (KPC)	1	2	-	2
Kalman Position Speed Consistency (Speed) (KPCS)	0	2	2	-
Kalman Position Speed Consistency (Position) (KPCP)	0	1	1	-
Local Perception Verification (LPV)	0	1	1	-
Summary	-	13	5	8

- Step 1: Calculation of function value: f(sumWeights) = $-1.3^{-sumWeights}+1 = -1.3^{-13}+1 = 0.97$
- Step 2: Belief calculation: $b = (weightBlief/sumWeights) * 0.97 = (5/13) * 0.97 = 0.37$
- Step 3: Disbelief calculation: $d = (weightDisbelief/sumWeights) * 0.97 = (8/13) * 0.97 = 0.60$

- Step 4: Uncertainty calculation: $u = 1 - f(sumWeights) = 1 - 0.97 = 0.03$

Resulting trust opinion: $\omega = (0.37, 0.60, 0.03)$

9.2 Trust Quantification Approach for AIV

The Attestation and Integrity Verification (AIV) is a component used in in-vehicle networks, which provides trustworthiness claims to the trustor including evidence about the trustworthiness of the vehicle's components, such as ECUs or vehicle computers. In the following, the trust quantification approach is described for a vehicle computer, called VC1. The trustworthiness claims include evidence about several aspects about the ECU. The evidences taken into account in this project are shown in Table 9.3

Table 9.3: AIV Evidence

Control	Description
Access Control	Regulates which process can read or modify resources in the system (System Integrity)
Secure Over The Air (OTA) Updates	Remote software updates ensuring data integrity of the delivered update (System Integrity)
Secure Boot	Mechanism that ensures the device starts using only trusted and authenticated software (System Integrity)
Application Isolation	Mechanism that isolates applications from each other and the system, such as virtual machines (Application Integrity)
Control Flow Integrity	Mechanism that ensures that the execution sequence of a program corresponds to the intended execution sequence (Application Integrity)
Configuration and Integrity Verification	Mechanism that ensures the correctness of the configuration of any device that is participating in the service graph chain (Application Integrity)

The evidence about these security controls is received from the AIV. For each control, three values can be provided as evidence. In the following the values and the semantics are shown for secure boot:

- Value 0: Secure Boot didn't run successfully
- Value 1: Secure Boot run successfully
- Value -1: Secure Boot not implemented/active

The trust quantification approach is divided into three steps. These steps are described in the following part of this paragraph.

1. **Step: Design Time Trust Opinion + Weights Calculation** In the first step a design time trust opinion is calculated for the ECU based on design time information. This opinion is referred to as ω_{DTI} in the following. Here, all security controls are taken into account, for which during runtime no evidence can be provided about their existence and if they are

active or not. Security controls for which during runtime evidence can be provided are considered in the following steps. The design time trust opinion is calculated based on the output of a TARA and the identified risks and risk levels. For this purpose, the CONNECT Risk Assessment Engine described in Chapter 8 can be used. In addition to the design time trust opinion, also weights are calculated for each foreseen security control, which is checked during runtime.
The design time trust opinion, the weights and which security controls are to be checked during run time, is stored in the trust model. During run time, these controls are taken into account to calculate the trust opinion.

Example An example of identified risks and risk levels is shown below for a vehicle computer. We assume here, that six security controls (C1 to C6) are foreseen in VC1 which are checked during run time. Each column represents the risk level in case the corresponding security control is active in the system.

Table 9.4: Risks and risk levels for VC1

Risk	Original Risk	C1	C2	C3	C4	C5	C6
R1	3	2	1	0	2	3	3
R2	3	2	1	3	2	3	2
R3	4	1	4	4	1	4	2
R4	2	1	2	2	1	2	2
R5	1	1	1	1	0	1	1
R6	4	2	4	4	2	1	1
R7	1	1	1	1	1	1	1
Risk Summary	**18**	**10**	**14**	**15**	**9**	**15**	**12**

In the following, the formula is described, based on which the weights are calculated. This formula takes the risk levels of the identified risks into account, which are shown in the table above.

$$W_{C_x} = \frac{sumRisks_{\emptyset} - sumRisks_{C_X}}{\sum_{N=1}^{|C|}(sumRisks_{\emptyset} - sumRisks_{C_N})}$$

$$W_{C_1} = \frac{18-10}{(18-10)+(18-14)+\cdots} = \frac{8}{8+4+3+9+3} = \mathbf{0.24}$$

$$W_{C_2} = \mathbf{0.12}\,,\ W_{C_3} = \mathbf{0.09}\,,\ W_{C_4} = \mathbf{0.27}\,,\ W_{C_5} = \mathbf{0.09}\,,\ W_{C_6} = \mathbf{0.18}$$

2. **Step: Runtime Trust Opinion based on existence of controls** During runtime, the TAF receives evidence from the Attestation and Integrity Verification Component (AIV), regarding the implementation status of designated security controls. Each control is marked as either not implemented (Value -1) or implemented (Value ≥ 0). The runtime trust opinion is computed based on the specified weights in conjunction with the design-time trust opinion. The weights are used to calculate a delta value, which indicates how much the belief or disbelief should be adjusted depending on whether each control is implemented or missing. This adjustment ensures that the trust assessment accurately reflects the current security posture. The delta values are always the same for the belief, disbelief and uncertainty. These values are calculated based on specified weights and a design time trust opinion.
Example The equation for the calculation of the delta values of the single security controls is shown below:

$$\Delta b_{C_x} = \Delta d_{C_x} = \Delta u_{C_x} = W_{C_x} \times u_{DTI} \tag{9.1}$$

Lets assume that the calculated design time trust opinion is $\omega_{DTI} = (0.25, 0.18, 0.57)$. The amount of belief/disbelief added by each control is calculated with Equation 9.1. We assume that only for the the controls C_1 to C_4 evidence was provided. There for only for these controls the delta values are calculated in the following. For C_1, C_2 and C_4 positive evidence was provided. Only for C_3 negative evidence was provided. The corresponding delta values are:

$$\Delta b_{C_1} = \Delta u_{C_1} = 0.24 \times 0.57 = 0.13$$
$$\Delta b_{C_2} = \Delta u_{C_2} = 0.12 \times 0.57 = 0.06$$
$$\Delta d_{C_3} = \Delta u_{C_3} = 0.09 \times 0.57 = 0.05$$
$$\Delta b_{C_4} = \Delta u_{C_4} = 0.27 \times 0.57 = 0.15$$

A visualization of the calculation of the run time trust opinion based on the existence of controls is shown in Figure 9.1. Here controls C1, C2 and C4 are implemented. C3 is not implemented and for C5 and C6 no evidence was received.

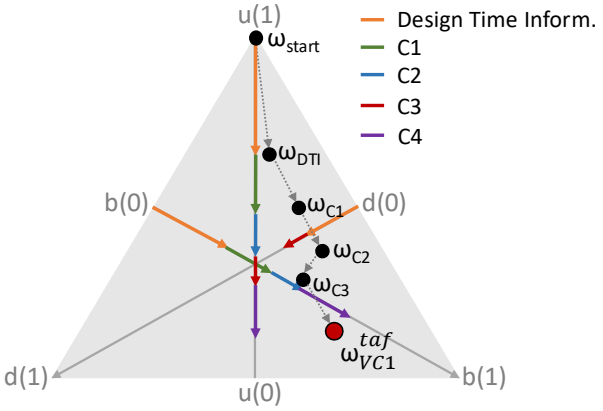


Figure 9.1: Calculation of a trust opinion based on implemented security controls.

3. **Step: Run Time Trust Opinion based on output of controls**
   In the third step the output(s) of the controls (if available) are taken into account. For instance, in the case of secure boot, the system checks whether the secure boot process was successful. The trust opinion calculated in step two is utilized, with adjustments made only if any negative output is detected from the security controls (e.g., if secure boot identifies an issue). During the design phase, it is predetermined how the trust opinion should respond to negative outputs for each control. Specifically, it can either remain unchanged, adjust according to a specified delta value, or shift to complete disbelief.
   Example The following scenarios illustrate these three cases. Lets assume that the calculated trust opinion in step 2 is $\omega_{VC1} = (0.58, 0.1, 0.1)$ and negative evidence is present only for C1.
   1st Case: If the trust opinion is to remain unchanged, then the output would be $\rightarrow \omega_{VC1} = (0.8, 0.1, 0.1)$
   2nd Case: If the trust opinion is adjusted based on the delta value for C1, let's assume $\Delta b_{C_1} = \Delta u_{C_1} = 0.10$ The new trust opinion would be $\rightarrow \omega_{VC1} = (0.7, 0.2, 0.1)$
   3rd Case: If the trust opinion shifts to complete disbelief, the output would change to $\rightarrow \omega_{VC1} = (0.0, 1.0, 0.0)$

After completing the three steps for all designated security controls, the trust opinion derived from

the third step becomes the final trust opinion. This final trust opinion is then recorded in the trust model for future reference and evaluation.

9.3 Trust Quantification Approach for TCH

The Trustworthiness Claims Handler (TCH) provides a `TCH_NOTIFY` message to the trustor. The goal of the TCH is to transmit trustworthiness claims about the in-vehicle topology in a privacy preserving manner. In the context of the CONNECT project, the TCH provides evidence about the vehicle computer of the external vehicle. The evidences of all other ECUs of the vehicle are not included in the verifiable presentations. The calculated trust opinion thus represents the trustworthiness of the vehicle computer, as this entity collects the data and send the CPM. The trustworthiness of the other ECUs will be included in the other trust opinions of the trust chain. This approach is visualized in Figure 9.2. Here the ego vehicle (V_e) wants to create a trust opinion on another vehicle (V_x). For this purpose, the V_e only takes the evidence about the vehicle computer (VC) of C_x into account. All other components within V_x are not taken into account to create the trust opinion on the vehicle V_x. These components only become relevant, when V_x creates a trust opinion on its own created data (C_x). The calculation of this trust opinion (ω_{Cx}^{VC}) is out of scope of this chapter.

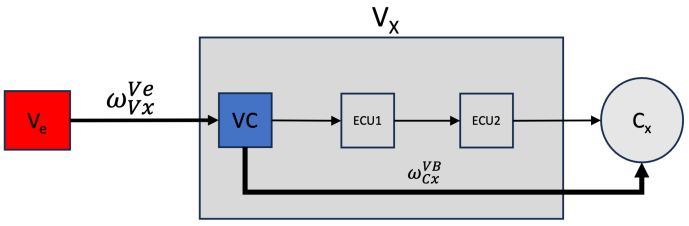


Figure 9.2: Trust model of vehicle

In the context of the CONNECT project, the several security controls are taken into account for an external vehicle. These security controls have been selected to take into account evidence about the system integrity, but also about the application integrity. The security controls are the ones shown in Table 9.3.

The trust quantification approach is very similar than the approach used for the AIV. For each of the specified controls two weights have to be specified. These weights can be specified, e.g. based on a security analysis of the corresponding application or based on the experience of a security analyst. The first set of weights are the existence weights. They specify how important it is, that a specific security control is actually in place and active in the system, e.g. secure boot is used in the system. These weights need to have a value between zero and one and should not add up to more than one. The second set of weights is used to take the output of the single security controls into account. This set of weights is only used in case there is negative output of the security controls, e.g. secure boot failed. These weights can have three different values. These three values define

- Value: 1: in case of negative output of the corresponding security control the output should not change
- Value: 2: in case of negative output of the corresponding security control the output should change by a certain amount

- Value: 3: in case of negative output of the corresponding security control the output should change to complete disbelief

A proposal of the weights is shown in Table 9.5.

Table 9.5: Proposal for weights.

Security Controls	Existence Weights	Runtime Output Weight
Access Control	0.16	2
Secure OTA	0.08	1
Secure Boot	0.24	3
Application Isolation	0.16	1
Control Flow Integrity	0.08	3
Configuration and Integrity Verification	0.24	3

Based on these weights the trust opinion can be calculated. The calculation of the trust opinion is done in two steps. These steps are described in the following part of this section.

1. **Step: Existence of controls** In the first step, it is analyzed if the controls are not implemented (Value -1) or implemented (Value ge 0) in the system. Based on the specified weights for each control, the trust opinion is calculated. The calculation starts with the trust opinion $\omega = (0, 0, 1)$. Based on the existence-weights, the delta value is specified, how much the belief or disbelief will be increased (depending on if the control is implemented or not). The specified weights corresponding here directly to the delta values.

   $\Delta b_{C_AC} = \Delta u_{C_AC} = 0.16$
   $\Delta b_{C_SOTA} = \Delta u_{C_SOTA} = 0.08$
   $\Delta b_{C_SB} = \Delta u_{C_SB} = 0.24$
   $\Delta d_{C_AI} = \Delta u_{C_AI} = 0.16$
   $\Delta b_{C_CFI} = \Delta u_{C_CFI} = 0.08$
   $\Delta d_{C_CIV} = \Delta u_{C_CIV} = 0.24$

   Example If the control is implemented, the belief is increased by the delta value and the uncertainty is reduced by the delta value. If the control is not implemented in the system, the disbelief is increased by the delta value and the uncertainty is reduced by the delta value. Based on the specified delta values, the resulting trust opinion is $\omega_{Step1} = (0.56, 0.40, 0.04)$.
2. **Step: Output of controls** In the second step, the output of controls (if available) are taken into account. For example, for secure boot, it is checked if secure boot run successfully or not. The trust opinion calculated in the first step is used and only adjusted in case of negative output of the security control (e.g. secure boot did not run successfully). If the security control did not run successfully, depending on the second weight specified for the control, the trust opinion will be adjusted:

   1. Case (Weight=1): Trust Opinion should not be adjusted
   2. Case (Weight=2): Trust Opinion should be adjusted based on the delta value for the control. Lets assume that for C_{AC} there is negative output. In this case the belief of the trust opinion will be reduced by the specified delta value and the disbelief will be increased by the delta value.
   3. Case: Trust Opinion should change to complete disbelief $\rightarrow \omega_{VC1} = (0.0, 1.0, 0.0)$

Example As only for the security control AC negative output was provided, only for this control the trust opinion calculated in the first step needs to be adjusted. The second weight for this control is two. Therefore, the trust opinion has to be adjusted by the delta value specified above ($\Delta b_{C_AC} = \Delta u_{C_AC} = 0.16$). The belief will be reduced and the disbelief will be increased by this delta value. This results in the trust opinion

$$\omega_{Step2} = (0.56 - 0.16, 0.40 + 0.16, 0.04) = (0.40, 0.56, 0.04) = \omega_{VC1} = \omega_{Vx}$$

After the two steps were done for all the foreseen security controls, the trust opinion calculated in the second step is the final trust opinion. This trust opinion is then stored in the trust model.

Chapter 10

Required Trustworthiness Level (RTL)

The Required Trustworthiness Level (RTL) defines the minimum level of trustworthiness required for a Trustee, such as a vehicle function or data, to be considered reliable. Its definition might be based on risk analysis output, technical requirements, or regulations for type approval. An RTL is established during the design phase, and the approach presented in this deliverable for calculating RTL is based on risk assessment. Using the subjective logic concept, RTL serves as a numerical trustworthiness threshold for trust decision-making.

RTL sets thresholds for minimum acceptable belief (b_t), maximum permitted disbelief (d_t), and maximum acceptable uncertainty (u_t) values of subjective trust opinions. These three thresholds can range from 0 to 1 and can be used independently of each other. Although they may share a common foundation, such as inputs from the same risk assessment, they consider different aspects. For b_t, 0 means no belief level is required, and 1 means full belief is needed. While b_t can be zero, having it at zero is not recommended, as it may indicate a breach in the zero-trust model. The range for d_t and u_t is also from 0 to 1, but they reflect their maximum accepted level. High tolerance of disbelief and uncertainty is not recommended, as it weakens decision-making and leads to vague conclusions due to a lack of trust and clarity. Figure 10.1 shows the boundaries set by this triad and indicates where a trustworthy opinion should be positioned. The green area indicates the belief threshold, the red area indicates the disbelief threshold, the yellow area represents the uncertainty threshold, and the blue area, formed by the overlap of the other three thresholds, is the RTL area.

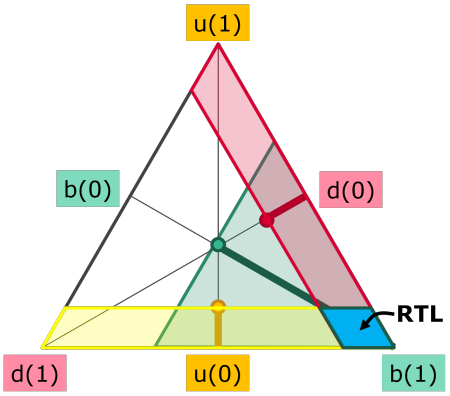


Figure 10.1: Graphical representation of RTL within subjective logic triangle

RTL is established during the design phase and serves as a threshold for trust decisions. Since the vehicle is still in development, RTL cannot rely on actual evidence. However, it is possible to

utilize risk assessment tools, along with assumptions or external requirements, such as company policies or type approval requirements, to evaluate anticipated cybersecurity scenarios.

To set RTL, we recommend taking advantage of risk assessment, impact evaluation, and the system's capabilities to detect cybersecurity incidents. The Threat Analysis and Risk Assessment (TARA), standardized by ISO/SAE 21434 [11], is employed by vehicle manufacturers during their development process to evaluate cybersecurity risks and serves as the foundation for defining RTL in our approach.

Even though RTL is predefined, it can still be adjusted at runtime based on risk assessments. New threats and vulnerabilities, the availability of a patch that addresses a critical cybersecurity issue, or the addition of new functions or components within the system may necessitate recalculating the RTL to align with these updated requirements. The CONNECT RA, as described in Chapter 8, illustrates how an RTL can be derived from a risk assessment tool from a technical perspective. Specifically, it shows how a risk assessment tool can be used in the definition of the RTL. For instance, with the CONNECT RA, an automated calculation of the RTL can be implemented, allowing it to be dynamic. This means that any changes in the scope-related risk assessment would be directly reflected in the RTL.

In Section 10.1, we define RTL based on thresholds of belief, disbelief, and uncertainty derived from design-time assessments, and introduce methods to calculate each of these thresholds. In Section 10.2, we demonstrate a practical application of these methods through a specific use case.

10.1 RTL thresholds

This section aims to present a method for estimating the RTL thresholds. To achieve this, it is essential to review some definitions from subjective logic and apply them within the RTL context, clarifying their meanings in this framework.

Uncertainty refers to a lack of information, making it difficult to determine whether a trustee is trustworthy. Since the system cannot reduce this uncertainty directly, we address it by evaluating the system's ability to detect misbehaviour. This may involve implementing misbehaviour detectors or using various sensors to verify whether the trustee is acting appropriately. Additionally, the assigned cybersecurity assurance level indicates the allowable level of uncertainty within the system, as it suggests the rigour of the testing process.

Disbelief refers to characteristics of a system or the relationship between the trustee and trustor that suggest the system is uncertain, often named as negative evidence. By using an impact-based approach, disbelief can be assessed based on the impact of damage scenarios. In other words, if certain threats are anticipated and the decision is made to accept the risk, or if any residual risk persists after applying mitigation strategies, this may be classified as negative evidence and the impact of having these residual risk is taking into consideration to calculate how much negative evidence the system can handle in a trustworthy mode.

In terms of belief, positive evidence is used to establish trust in the trustee. Secured connections and protocols, authentication methods, physical protection, and a variety of other features can all be used to create a trustworthy system. According to our risk-based approach, the riskier the system is, the greater the required belief value.

Similar to ATL values, RTL constraints are associated with a specific context, including the scope

and security properties. Therefore, the risks considered when deriving a set of RTL values are specifically tailored to that particular property. In the following sections, we will explore the definitions of all three thresholds, detailing the expected inputs and how to calculate each one.

10.1.1 Belief threshold calculation

In terms of belief, positive evidence is used to increase confidence in the trustee. Secure connections, protocols, authentication techniques, physical security, and various other security elements or strategies can be used to build a trustworthy system.

For example, consider two comparable systems that provide the same type and number of trust sources. Based on risk assessment, risk levels can vary from very low to very high. For instance, consider that systems X and Y have the same capabilities and the same identified risks, but while system X lacks security and safety features, system Y was designed with features to mitigate some critical risks, resulting in lower risk levels when compared to system X. In this example, system X would require more evidence to attest to its trustworthiness since its risk levels are higher. On the other hand, system Y would not require the same amount of evidence as X, considering its lower risk levels. The risk-based approach considers the risk levels of the item to determine the required trustworthiness level (RTL), to make the trust decision.

To calculate the belief threshold, we take advantage of the automotive Threat Analysis and Risk Assessment (TARA). TARA is a standardized framework implemented throughout vehicle design time [11], but may evolve throughout the lifecycle of the vehicle. This allows cybersecurity engineers to anticipate cybersecurity threats and give support to create strategies to minimize or mitigate the risk to the system. TARA has as output a list of threats and damage scenarios for a given technical system model, including *attack feasibility* (F), *impact rating* (I), and *risk level* (R) calculated from I and F.

In our risk-based methodology, the riskier the system, the higher the required belief value. The risk-based scheme for calculating the b_t component of the RTL is shown in Figure 10.2. This approach maps risk levels from risk assessment, considering risk likelihood and impact rating, into b_t.

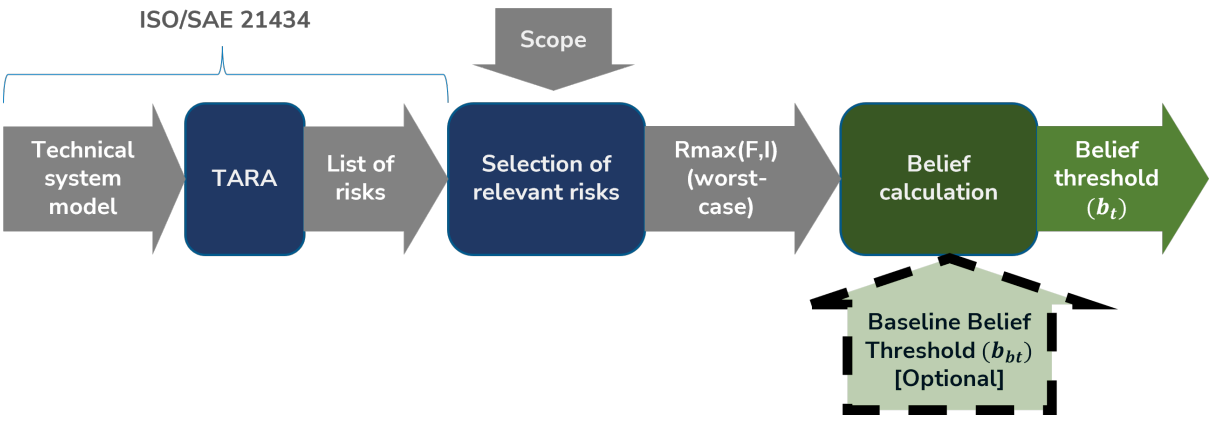


Figure 10.2: Risk-based belief scheme calculation

To calculate b_t, we leverage risk assessment, which enables engineers to anticipate risks the system will likely face and assists in making risk mitigation decisions. The risk assessment results in a list of risks attached with their impact and likelihood for a specific technical system model.

The first step, as illustrated in Figure 10.2, involves gathering the risk assessment output and identifying which risks are relevant to the specific scope. Among the wide range of potential risks, some may be irrelevant, so it is essential to focus on those that are scope-relevant. This step is crucial for ensuring that the b_t calculation addresses the most critical risks.

A recommended approach is to consider the worst-case scenario, which ensures that RTL accounts for all possible risk scenarios. However, this approach may require stronger trust evidence, making the system more stringent. In this document, we adhere to the criteria of the worst-case scenario.

From the list of relevant risks, we select the worst-case scenario based on the highest risk level. An optional input, the *Baseline Belief Threshold* (b_{bt}), is designed to establish a minimum required belief level following company-specific policies or type approval requirements that may not be addressed in the risk assessment. For instance, when the worst-case scenario has very low attack feasibility or a negligible impact rating, the b_{bt} helps enforce a stricter belief threshold. This ensures that belief requirements are not too low or absent.

To map the belief threshold from the risk level, the following mathematical equations are used:

$$\Delta = \frac{1 - b_t}{5}, 0 \leq b_t \leq 1 \tag{10.1}$$

Where Δ is the difference between levels, b_t is the set belief threshold, a number between 0 and 1, and 5 is the number of risk levels set by TARA.

In this deliverable, Δ is treated as a fixed value, resulting in evenly distributed spaces. However, in different applications, Δ can be represented as a function of the risk level, which allows for varying spaces between levels. For instance, if a positive quadratic function is assigned to Δ, the extremes will increase at a faster rate than the middle values. On the other hand, if an exponential function is used for Δ, the final belief threshold will remain similar for lower risks, but as the risk increases, the growth will become more pronounced.

b_t is defined by Equation 10.2

$$b_t = b_{bt} + ((R_{max}(F, I) - 1) \times \Delta) \tag{10.2}$$

Where b_t represents the belief threshold in the RTL, and $R_{max}(F, I)$ is the risk level for the worst case, ranging from 1 to 5. The risk function can vary between manufacturers, according to [11].

The equation referenced as 10.2 maps the risk level, which is categorized into five levels, into a belief threshold value ranging from 0 to 1. This equation involves two key variables: $R_{max}(F, I)$ and b_{bt}. The risk level function $R_{max}(F, I)$ is determined by the car manufacturer and can be customized to meet the specific needs of each company. Although the standard [11] defines risk levels from 1 to 5, we simplify this by using a scale from 0 to 4, which is why we subtract 1 from $R_{max}(F, I)$.

The variable b_{bt} is chosen by the company and is context-related, containing subjective factors and isolated considerations. It's optional and can be set to 0 if not applicable. This document will not elaborate on the determination of b_{bt}, as it is a requirement specific to each company. For our purposes, we assume b_{bt} takes into account various safety aspects, regulations, or any relevant characteristics that are not addressed by the risk assessment, and is set by the engineering team, when applicable.

10.1.2 Disbelief threshold calculation

Disbelief refers to the characteristics of a system or the interaction between the trustee and the trustor that indicate a lack of trustworthiness, often referred to as negative evidence. The calculation of the proposed disbelief threshold (d_t) employs an impact-based method that takes into account the consequences of realized damage scenarios. In essence, this means evaluating the impact of damages that arise from risks that were anticipated, as well as the decisions made to manage those risks. If any residual risk exists after implementing mitigation techniques, it may also be regarded as negative evidence. In such cases, the impact of accepting these risks influences the level of disbelief allowance. Aspects such as safety, economic factors, privacy, operational considerations, and other relevant areas play a crucial role in defining d_t. It is assumed that d_t represents the level of risk that the system can accept while maintaining a trustworthy status.

Figure 10.3 outlines the impact-based approach for calculating d_t . This method involves a deeper evaluation of the scope-relevant risks identified in the risk assessment regarding their impact. By examining the potential impact of these risks on several aspects, we can determine d_t.

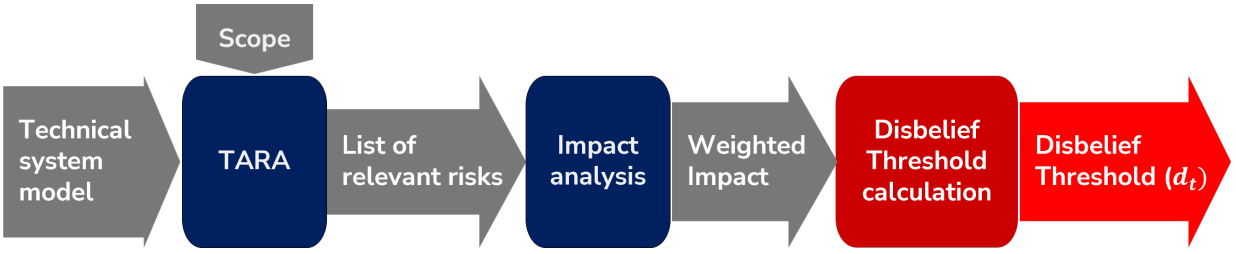


Figure 10.3: Impact-based disbelief threshold calculation method.

As shown in Figure 10.3, the first step is to gather the risk analysis output, which includes a list of expected cybersecurity and safety risks and the impact of these risks in case of occurrence. For example, we can consider these impacts, when possible:

- **Safety Impact**: The possible implications of a system failure or compromise on human safety play an important role in evaluating disbelief. Risks that directly endanger human life or well-being will significantly increase disbelief, decreasing d_t.
- **Privacy Impact**: Another key aspect is the system's ability to compromise user privacy. Risks involving unauthorized access, acquisition, or disclosure of sensitive information will raise disbelief and decrease the d_t.
- **Economic Impact**: The possible financial ramifications of a system failure or compromise, such as lost income, operational interruptions, or legal obligations, can all add to disbelief and decrease d_t.
- **Operational Impact**: Another element to examine is how a system failure or compromise affects the system's capacity to operate effectively and efficiently. Risks that threaten the system's operating capability will increase disbelief and decrease d_t.

More impact categories can be added to the list, depending on the investigated scope.

The method to calculate the disbelief threshold (d_t) uses impact rating. The impact rating can be evaluated using external frameworks such as Failure Mode and Effects Analysis (FMEA) [16] for operational impact analyses, or the gathered impact rating within the used risk assessment. For this deliverable, we use the impact assessment within TARA.

Based on the scope, each impact category (safety, financial, operational, and privacy) needs to be weighted by relevance; we suggest that the sum of the weights be 1.

Next, we calculate the weighted impact rating (I_w), considering the weights for each category, applying Equation 10.3.

$$I_w = \sum_{n=1}^{3} I_n W_n \tag{10.3}$$

In this context, the attributes are defined as follows:

- I_n represents the impact ratings (I_1 = safety, I_2 = financial, I_3 = operational and I_4 = privacy).
- W_n represents the given weights for each impact category (W_1 = safety, W_2 = financial, W_3 = operational and W_4 = privacy).

The higher the potential impact, the lower the accepted disbelief should be. In other words, d_t is inversely proportional to the impact level and it is expressed by Equation 10.4.

$$d_t = 1 - I_w \tag{10.4}$$

For multiple damage scenarios to derive impact ratings, we recommend using the worst-case scenario by selecting the damage scenario with the highest weighted impact rating as the basis for this analysis.

We can translate the impact rating to a numerical value as suggested in Table 10.1. We adapted the values suggested by the example in ISO/SAE 21434 [1], to range between 0 and 1.

Table 10.1: Translation of impact rating to a numerical value

Impact rating (I)	**Numerical value for impact rating**
Negligible	0
Moderate	0.5
Major	0.75
Severe	1

In case of a different number of impact ratings defined by the used impact assessment framework, this translation needs to be adapted accordingly. When using TARA, Equation 10.3 and 10.4 can be used directly, as suggested.

As in the belief threshold, this method allows for adding a baseline disbelief threshold (d_{bt}), defined by engineering decisions, whose range is from 0 to 1 and represents the upper limit of accepted disbelief, considering design or engineering decisions. d_{bt} is optional and if it is defined, d_t assumes its value when d_{bt} is lower than the calculated d_t.

10.1.3 Uncertainty threshold calculation

Uncertainty, in the context of RTL, refers to the lack of knowledge or information for which there is not enough evidence to establish confidence. During the design phase, factors such as the

required level of assurance of the system, which can be interpreted as system criticality, since this may be used to define testing rigour and its acceptance of uncertainty. In addition, we consider the ability of the system to detect an incident as parameters to set the maximum acceptable level of uncertainty (u_t) in a given scope, as the process illustrated in Figure 10.4.

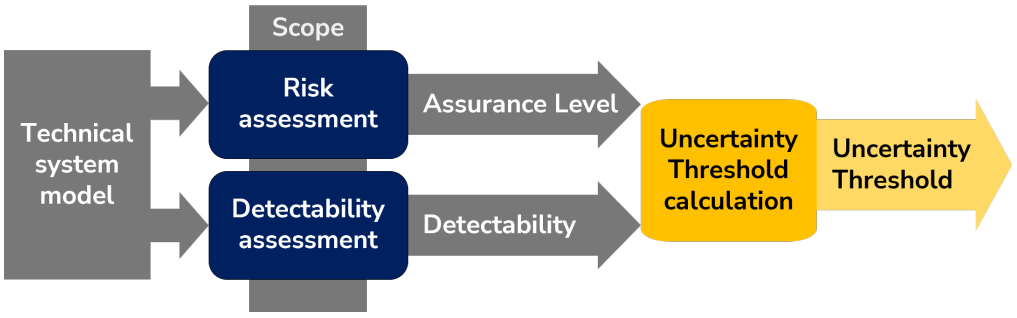


Figure 10.4: Uncertainty threshold calculation method.

Figure 10.4 summarizes the key factors that can be considered to calculate u_t:

- **Detectability of an incident**: The ability to detect a system misbehaviour that triggers a risk is an important component in assessing u_t. A system with strong detectability mechanisms may accept more uncertainty (u_t) because it can identify, and possibly mitigate, certain risks before they may exploit weaknesses. Detectability can be covered by both safety (e.g., detectability of a failure mode) and security (e.g., detectability of a cybersecurity incident) perspectives. Detectability is concerned with the detector's coverage and accuracy. This has more impact when defining allowed uncertainty.
- **Assurance level**: Uncertainty is influenced by the system's overall security and safety posture, and system maturity. A higher required assurance level requires a system to be developed with more depth and rigour (e.g., testing), allowing less uncertainty by design. The higher the required assurance, the lower u_t.

To calculate u_t, we consider the interactions of these factors. A system requiring a higher assurance level is designed to allow less uncertainty than one with a lower requirement. When it comes to the detectability of incidents, a system with few or inappropriate detectors is not expected to deal with uncertainty as well as one with satisfactory detectability features. All these discussed capabilities and expectations need to be reflected by u_t. For all parameters, the scope is applied, and only relevant aspects are considered during analyses.

The suggested factors to be considered for the u_t definition are detectability and required assurance level. Table 10.2 exemplifies how these two factors might be combined to define the uncertainty acceptance level (UAL). To determine the required assurance level indicated in the table, we refer to the Cybersecurity Assurance Level (CAL) as defined in ISO/SAE 21434 [1]. The CAL is categorized from CAL 1 to CAL 4, with CAL 4 representing the highest required level of cybersecurity assurance, suggesting a greater need for stricter testing, for example. This table can be company-specific custom.

We can observe that a system with high incident detectability and a low required assurance level is expected to accept more uncertainty; on the other hand, one with a very high required assurance level and low detectability has a very low uncertainty acceptance level. This table is suggested to be created following each system and scope; the criteria may differ based on

Table 10.2: Uncertainty acceptance level (UAL) in the system based on assurance level and level of detectability of incidents in the automotive cybersecurity perspective

UAL			Required assurance level			
			Low	Moderate	High	Very High
			CAL 1	**CAL 2**	**CAL 3**	**CAL 4**
Detectability	Detectors are present and cover the system with full or high coverage and accuracy. They can identify the root cause of the security violation.	**High**	Very high	High	High	Moderate
	Detectors are present within the core system, capable of efficiently detecting cybersecurity incidents; however, their coverage is limited, they lack the capacity for root cause identification, or exhibit moderate accuracy.	**Moderate**	High	Moderate	Moderate	Low
	Incident detectors are either absent or do not fully extend to the core system.	**Low**	Moderate	Low	Low	Very Low

the impact of the system under analysis and how flexible in terms of uncertain information. The ratings for detectability or assurance level can also include more levels to include more granularity.

To establish u_t, we map the uncertainty acceptance level, which ranges from very low to very high depending on detectability and assurance, to a numerical scale.

Uncertainty acceptance level (UAL) assumes values as follows: Very Low: 1, Low: 2, Moderate: 3, High: 4, Very High:5. To derive it to u_t, we need to map it to u_t, whose range is from 0 to 1. The simplest way is to map UAL proportionally to this interval, which we call, by using Equation 10.5.

$$u_t = UAL_{map} = \frac{UAL}{5} \tag{10.5}$$

As in the other thresholds, this mapping allows for adding a baseline uncertainty threshold (u_{bt}), defined by engineering decisions, whose range is from 0 to 1 and represents the upper limit of accepted uncertainty considering design or engineering decisions. u_{bt} is optional and if it is defined, u_t assumes its value when u_{bt} is lower than the calculated u_t.

10.2 Use case application

The use-case architecture shown in Figure 10.5 is used to demonstrate the application of the proposed methods. The use case focuses on the transmission of location data received by the

Global Navigation Satellite System (GNSS) across the vehicle network to the vehicle computer, passing through the Electronic Control Unit (ECU), responsible for processing the GNSS data, and the zonal controller. The GNSS data passes through communication channels 1, 2, and 3. In this use case, these channels are considered insecure, meaning they lack cybersecurity features. Consequently, an attacker can access or tamper with the data exchanged over these channels.

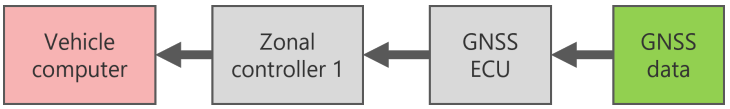


Figure 10.5: Simplified running example architecture

Based on the presented use case scenario, we derive the trust network between the vehicle computer (trustor) and the GNSS data (trustee). As a use-case scope, we are focusing on the GNSS data integrity, from the GNSS to the vehicle computer.

10.2.1 TARA and definition of the trust assessment scope

A TARA is conducted on the item shown in Figure 10.5 to support the use case application. These steps are simplified to enhance understanding, as the focus is not on TARA itself but on using its output for the thresholds calculation. The TARA results, summarized in the following tables, illustrate an example rather than a comprehensive professional analysis.

Item definition

The use-case architecture is illustrated in Figure 10.5 and shows that the item is composed of GNSS data, ECU, zonal controller, and vehicular computer, including channels 1, 2, and 3 that connect the components. Additionally, the communication between the components is unprotected. The operational environment is not considered for this example. The main function of the item in this use case is to receive location data and make it accessible on the vehicle computer.

Asset identification and damage scenario

Table 10.3 shows a list of damage scenarios that may occur if cybersecurity properties are compromised.

Threat scenario identification and attack feasibility

Threats lead to a damage scenario. Table 10.4 presents a list of threats that can be performed against the use case scenario, as well as their attack feasibility.

Impact rating

Table 10.11 presents the impact rating in case a damage scenario occurs, regarding safety (S), financial (F), operational (O), and privacy (P).

Table 10.3: Asset identification and damage scenario

ID	Asset	Damage scenario	Cybersecurity property
DS.1	GNSS data	The vehicle location can be accessed by an attacker	Confidentiality
DS.2	GNSS data	The GNSS data can be adulterated by an attacker	Integrity
DS.3	Vehicle computer	The firmware on the vehicle computer can be changed by an attacker	Integrity
DS.4	Zonal controller	The firmware on the zonal controller can be changed by an attacker	Integrity
DS.5	ECU	The data stored on the ECU can be accessed by an attacker	Confidentiality
DS.6	GNSS data	There's no GNSS signal and the vehicle cannot be located	Availability

Table 10.4: Threat scenario identification and attack feasibility

ID	Threat scenario	Attack feasibility
TS.1	Information Disclosure on ECU	Medium
TS.2	Information Disclosure on Zonal controller	Low
TS.3	Information Disclosure on Vehicle computer	Very Low
TS.4	Tampering on GNSS data	High
TS.5	Tampering on Vehicle computer	Very Low
TS.6	Tampering on Zonal controller	Low
TS.7	Physical Interference	High

Table 10.5: Impact rating

Damage scenario	S	F	O	P
DS.1	Negligible	Negligible	Negligible	Moderate
DS.2	Moderate	Negligible	Major	Moderate
DS.3	Severe	Negligible	Severe	Negligible
DS.4	Moderate	Moderate	Severe	Negligible
DS.5	Negligible	Negligible	Negligible	Moderate
DS.6	Negligible	Negligible	Major	Negligible

Risk level determination

To translate the impact rating and attack feasibility rating into risk level, with a numeric value, we are using the values in Table 10.6 and Table 10.7, also used for the example shown in the ISO/SAE 21434 [11].

Table 10.6: Translation of impact rating to a numerical value [11]

Impact Rating	Numerical value I for impact rating
Negligible	0
Moderate	1
Major	1.5
Severe	2

Table 10.7: Translation of attack feasibility to a numerical value [11]

Attack feasibility	Numerical value F for attack feasibility
Very Low	0
Low	1
Medium	1.5
High	2

For the risk level (R) calculation, we are using the following Equation 10.6 (from [11]).

$$R = 1 + I \times F \tag{10.6}$$

The translated Risk level is shown in Table 10.8.

Table 10.8: Risk level calculation considering Equation 10.6 [11]

	Very Low	Low	Medium	High
Severe	1	3	4	5
Major	1	2.5	3.25	4
Moderate	1	2	2.5	3
Negligible	1	1	1	1

Table 10.9 presents the risk level for each threat scenario and damage scenario listed in Table 10.4 and Table 10.11, for each impact category.

Cybersecurity Assurance Level and Detectability

For this example, consider that the engineering team has assigned the cybersecurity assurance level as CAL3, which means that a high cybersecurity assurance is required, and the cybersecurity activities, such as analysis and tests to search for vulnerabilities, are based on explorative methods, and the cybersecurity assessment is performed by a different team. In addition, considered that the system has no ability to identify whether an attack is occurring or a vulnerability is being exploited, resulting in a low detectability rating.

Table 10.9: Risk levels

ID	Threat scenario	Damage scenario	S	F	O	P
R.1	TS.1	DS.1	1	1	1	*2.5*
R.2	TS.1	DS.5	1	1	1	*2.5*
R.3	TS.2	DS.1	1	1	1	*2*
R.4	TS.3	DS.1	1	1	1	1
R.5	TS.4	DS.2	3	1	*4*	3
R.6	TS.5	DS.2	1	1	1	1
R.7	TS.5	DS.3	1	1	1	1
R.8	TS.6	DS.2	2	1	*2.5*	2
R.9	TS.6	DS.4	2	2	*3*	1
R.10	TS.7	DS.6	1	1	*4*	1

Trust assessment scope

The trust network is created between the vehicle computer (trustor) and the GNSS data (trustee). The vehicle computer is interested in assessing the trustworthiness of the integrity of the GNSS data.

10.2.2 Belief calculation

The GNSS data goes through an unprotected communication channel. The use case scope is focused on the integrity of the transmitted location data from the GNSS to the vehicle computer.

The cybersecurity engineering team conducted a Threat Assessment and Risk Analysis (TARA), which led to the creation of Table 10.10, detailing the risk levels for the current example. To calculate the belief threshold, we need to consider the scope to select the relevant list of risks. In this case, the required integrity of the GNSS data is associated with the damage scenarios DS.2, DS.3, and DS.4, as shown in Table 10.10. Consequently, the threats linked to these damage scenarios are TS.4, TS.5, and TS.6, referenced in Table 10.4. The relevant list of risks, along with their corresponding risk levels based on safety (S), financial (F), operational (O), and privacy (P) impacts, is summarized in Table 10.10.

Table 10.10: List of relevant risks

ID	Threat scenario	Damage scenario	S	F	O	P
R.5	TS.4	DS.2	3	1	*4*	3
R.6	TS.5	DS.2	1	1	1	1
R.7	TS.5	DS.3	1	1	1	1
R.8	TS.6	DS.2	2	1	*2.5*	2
R.9	TS.6	DS.4	2	2	*3*	1

Upon examining Table 10.10, it is clear that the highest risk level is R.5, with an operational risk score of 4 and a safety risk score of 3. In this case, we consider the highest risk—operational risk—as the primary risk level, consistent with the approach outlined in [11]. Therefore, the most significant relevant risk for the scope $R_{max}(F, I)$ is 4, corresponding to R.5.

For this use case, we are going to assume a belief baseline threshold (b_{bt}) equal to 0.5; in other words, we consider that the vehicle manufacturer has decided that the minimum acceptable belief must be equal to or higher than 0.5 for any case.

To calculate b_t, we need to calculate Δ first, by equation 10.1:

$$\Delta = \frac{1 - b_{bt}}{5} = \frac{1 - 0.5}{5} = \mathbf{0.1} \tag{10.7}$$

Replacing Δ in the equation 10.2, we have:

$$b_t = b_t + ((R_{max}(F, I) - 1) \times \Delta) = 0.5 + ((4 - 1) \times 0.1) = \mathbf{0.8} \tag{10.8}$$

For the given example, the minimum acceptable belief is 0.8. In other words, the TAF needs to collect pieces of evidence to reach at least a belief equal to 0.8 to consider the GNSS data trustworthy. The representation of the belief threshold is shown in Figure 10.6. As the risk level is very high, it shrinks the trustworthy area to 4%, when we consider only the belief threshold.

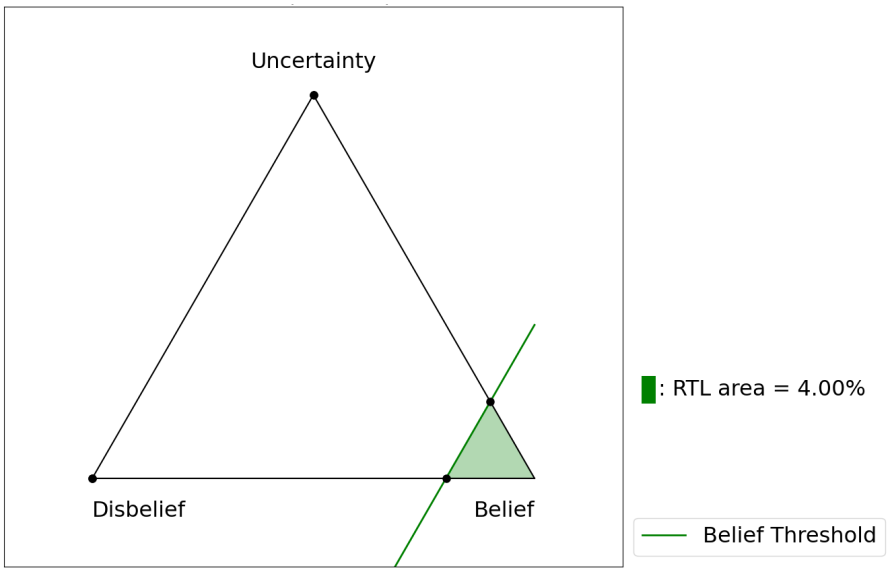


Figure 10.6: Graphical representation of belief threshold within subjective logic triangle, with b_{bt} set as 0.5.

If b_{bt} was not considered, the new b_t would be as followed:

$$\Delta = \frac{1 - b_{bt}}{5} = \frac{1 - 0.0}{5} = \mathbf{0.2} \tag{10.9}$$

Replacing Δ in the equation 10.2, we have:

$$b_t = b_t + ((R_{max}(F, I) - 1) \times \Delta) = 0.0 + ((4 - 1) \times 0.2) = \mathbf{0.6} \tag{10.10}$$

With a belief threshold at 0.6, it increases the RTL area to 16%, as shown in Figure 10.7.

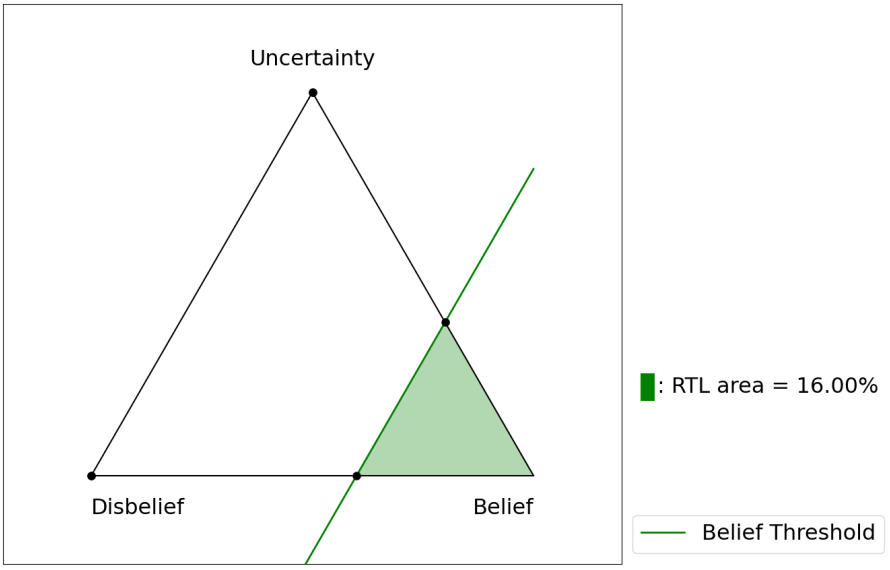


Figure 10.7: Graphical representation of belief threshold within subjective logic triangle, with b_{bt} set as 0.0.

10.2.3 Disbelief calculation

To calculate the maximum accepted disbelief expressed by (d_t), we observe the relevant damage scenarios for the scope, as defined in the performed TARA, where the impact ratings are specified. In addition, we weigh these impact ratings by relevance for each scope. After filtering and weighting the impact categories, we can calculate d_t using Equations 10.3 and 10.4. Following, we perform the d_t calculation for the scenario. As the scope is related to the integrity of the GNSS data, let's consider that the engineering team has assigned the following weights for the impacts: Safety: 0.5; Financial: 0.2; Operational: 0.3; and Privacy: 0.0.

Observing Table 10.11, from the performed TARA, we can check the impact ratings for the relevant damage scenarios (DS.2, DS.3, DS.4), also considering the translation shown in Table 10.1.

Table 10.11: List of relevant Impact ratings, with weights

Damage scenario	S (0.5)	F (0.2)	O (0.3)	P (0.0)
DS.2	Moderate (0.5)	Negligible (0)	Major (0.75)	Moderate (0.5)
DS.3	Severe (1)	Negligible (0)	Severe (1)	Negligible (0)
DS.4	Moderate (0.5)	Moderate (0.5)	Severe (1)	Negligible (0)

For each risk, let's calculate the weighted impact, using Equation 10.3:

$$
\begin{aligned}
DS.2 : I_w &= \sum_{n=1}^{3} I_n W_n = 0.5 \times 0.5 + 0.2 \times 0.0 + 0.3 \times 0.75 + 0.0 \times 0.5 = 0.475 \\
DS.3 : I_w &= \sum_{n=1}^{3} I_n W_n = 0.5 \times 1.0 + 0.2 \times 0.0 + 0.3 \times 1.0 + 0.0 \times 0.0 = \mathbf{0.8} \\
DS.4 : I_w &= \sum_{n=1}^{3} I_n W_n = 0.5 \times 0.5 + 0.2 \times 0.5 + 0.3 \times 1.0 + 0.0 \times 0.0 = 0.65
\end{aligned}
\tag{10.11}
$$

As DS.3 has the highest I_w, as 0.8, we apply Equation 10.4 for DS.3, as following:

$$d_t = 1 - I_w = 1 - 0.8 = \mathbf{0.2} \tag{10.12}$$

In this case, the disbelief threshold is defined as 0.2. The representation of the disbelief threshold is shown in Figure 10.8. As the impact level is high, it shrinks the trustworthy area to 36%, when we consider only the disbelief threshold.

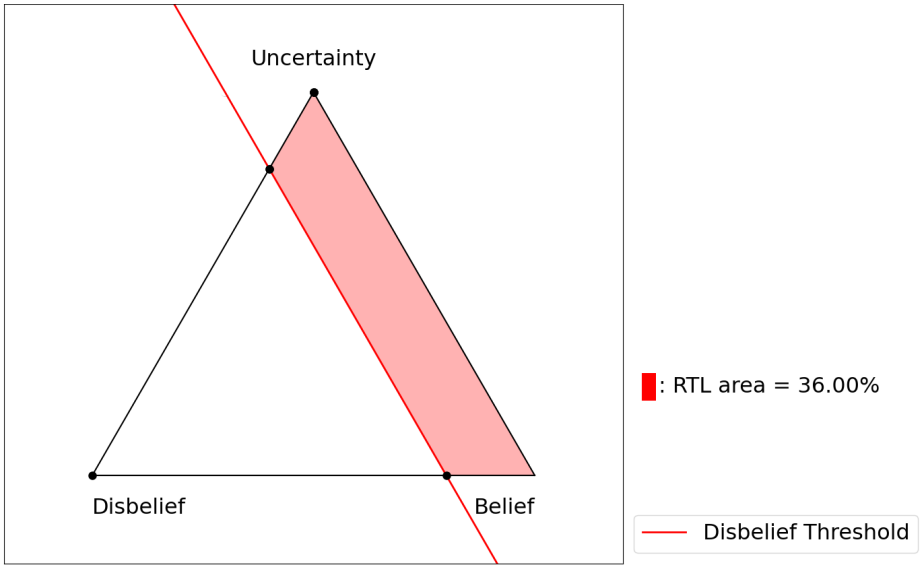


Figure 10.8: Graphical representation of disbelief threshold within subjective logic triangle

10.2.4 Uncertainty calculation

As defined in the TARA, the system requires high cybersecurity assurance (CAL3), meaning extensive and independent testing is done to find vulnerabilities. However, a significant problem is its low detectability: it can't tell if an attack is happening or if a weakness is being exploited. Therefore, despite strong preventive measures, the system has a low tolerance for uncertainty because it cannot detect ongoing threats. Considering these two facts, high assurance level and low detectability, from Table 10.2, the system has a low acceptance of uncertainty. Applying Equation 10.5, where UAL is Low (2), we have:

$$u_t = UAL_{map} = \frac{UAL}{5} = \frac{2}{5} = \mathbf{0.4} \tag{10.13}$$

In that case, the uncertainty threshold is defined as 0.4. The representation of the uncertainty threshold is shown in Figure 10.9. As the UAL is low, it shrinks the trustworthy area to 64%, when we consider only the uncertainty threshold.

10.2.5 Final RTL for the use case

For the use case, we have set the belief threshold at 0.8 (for b_{bt} = 0.5), the disbelief threshold at 0.2, and the uncertainty threshold at 0.4, based on the previous calculation. The final RTL is

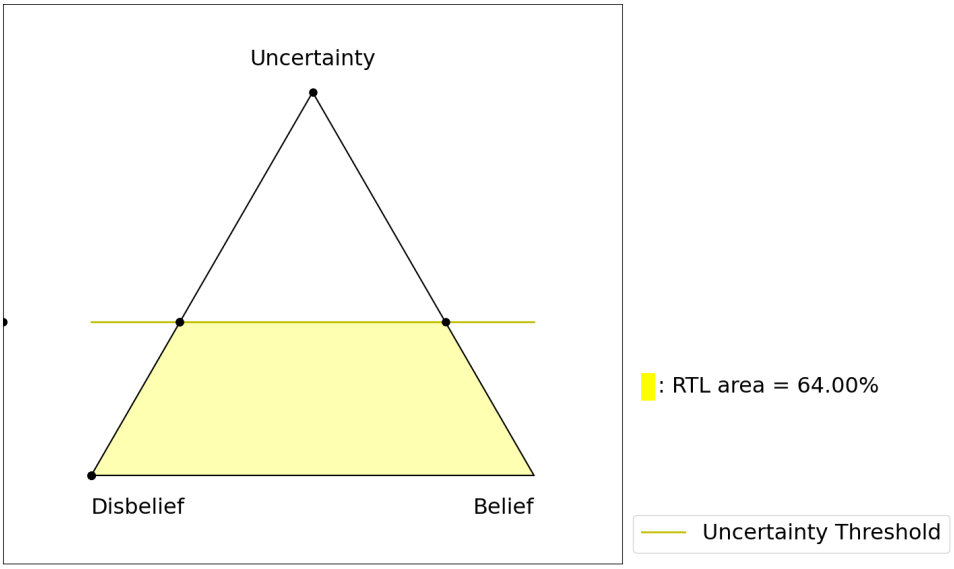


Figure 10.9: Graphical representation of uncertainty threshold within subjective logic triangle

shown in Figure 10.10. Note that the final RTL shrinks the trustworthy area to 4%, as the belief threshold is quite strict.

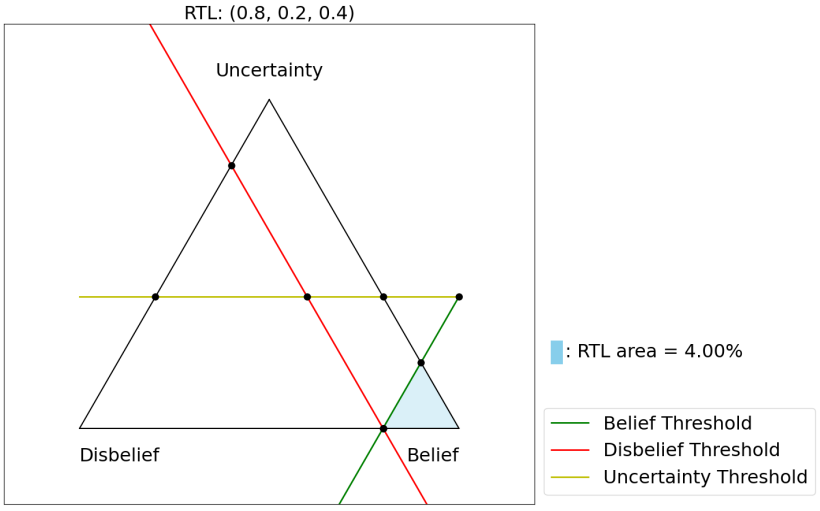


Figure 10.10: Graphical representation of final RTL within subjective logic triangle, when bt = 0.8

If the belief threshold is set at 0.6, defined when b_{bt} = 0.0, we observe a larger RTL area of 12%, shown in Figure 10.11.

These thresholds can be used independently, according to the use case. A belief threshold can be applied in systems where reliability is tied to the accumulation of positive evidence, or where only positive evidence can be checked. In systems that are more sensitive to negative evidence and can gather this type of information, it is recommended to implement a disbelief threshold. Additionally, an uncertainty threshold is recommended for systems where precision, accuracy, and certainty are crucial, especially in highly safety-critical contexts. The definition of which combination of these thresholds is more relevant for certain contexts is an open point that can be investigated in future work.

For example, if we consider only disbelief and uncertainty thresholds (belief = 0.0), the final RTL would be as shown in Figure 10.12. Note that the trustworthy area now is 16%.

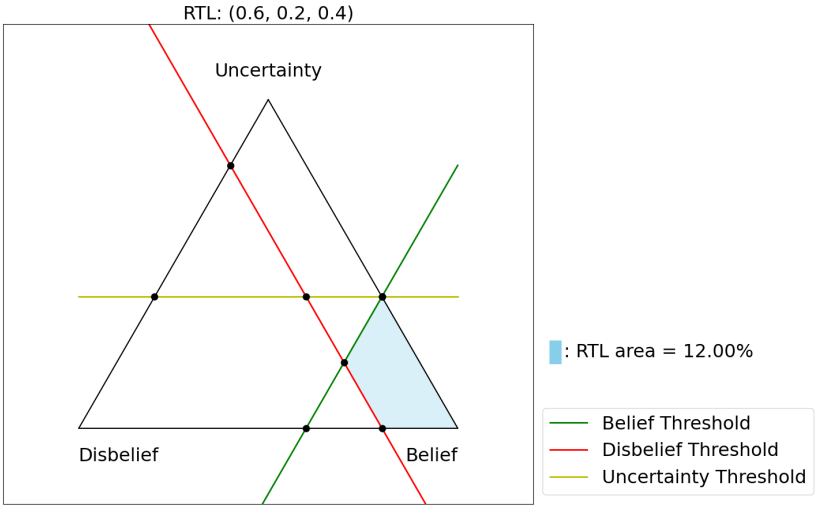


Figure 10.11: Graphical representation of final RTL within subjective logic triangle, when bt = 0.6

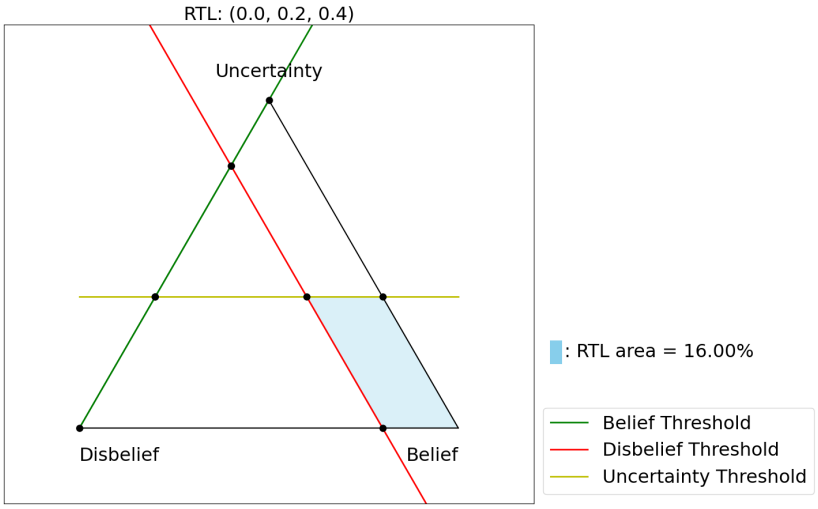


Figure 10.12: Graphical representation of RTL within subjective logic triangle, considering only disbelief and uncertainty thresholds

Chapter 11

Conclusion

Based on Deliverables D3.1 and D3.2, this document describes the final state of the Trust Assessment Framework (TAF) and all its related components, superseding D3.2.

Specifically, this deliverable details the system requirements and specifications of the TAF and presents the finalized architecture, including descriptions of both external and internal interfaces. Additionally, it provides implementation details for the prototype version of the TAF.

Beyond the core trust assessment framework, the document also describes the final state of related components, such as the Trustworthiness Level Expression Engine and the Risk Assessment Framework. It further explains the calculation of individual trust opinions based on various trust sources, as well as the determination of the required trustworthiness levels.

The deliverable also includes detailed descriptions of both the Federated TAF and the Digital Twin-TAF variants.

Based on the implementation of the TAF and its related components, comprehensive evaluations were conducted. All defined KPIs were successfully met, demonstrating that the TAF fulfills the specified requirements established at the beginning of the work package.

With this achievement, the last deliverable of work package 3 and the work package itself are concluded except for minor activities and provisioning of the open-sourced TAF.

List of Abbreviations

Abbreviation	Translation
ACC	Adaptive Cruise Control
AIV	Attestation and Integrity Verification
ATL	Actual Trustworthiness Level
BCF	Belief Constraint Fusion
CAM	Cooperative Awareness Message
CFI	Control Flow Integrity
CFI	Cumulative Risk Level
CPM	Collective Perception Message
DLT	Distributed Ledger Technology
DRL	Drools Rule Language
DSPG	Directed Series-Parallel Graph
DT	Digital Twin
ECU	Electronic Control Unit
GNSS	Global Navigation Satellite System
HARA	Hazard Analysis and Risk Assessment
IDS	Intrusion Detection Systems
IoC	Indicators of Compromise
IRL	Individual Risk Level
LDM	Local Dynamic Map
MBD	Misbehavior Detection System
MEC	Mobile Edge Computer
NDQ	Node Discovery Query
PPS	Parallel-Path Subnetwork
PR	Privileges Required
RA	Risk Assessment
RFRA	Request For Risk Assessment
RTL	Required Trustworthiness Level
STN	Subjective Trust Network
TAF	Trust Assessment Framework
TAF-DT	Trust Assessment Framework - Digital Twin
TAM	Trust Assessment Manager
TAQ	Trust Assessment Query
TAR	Trustworthiness Assessment Request
TARA	Threat Analysis and Risk Assessment
TAS	Trust Assessment Service

TC	Trustworthiness Claim
TC	Trustworthiness Claims Handler
TD	Trust Decision
TDE	Trust Decision Engine
TMI	Trust Model Instance
TMT	Trust Model Template
TSM	Trust Sources Manager
TSR	Trust Source Request
VC	Vehicle computer
WBF	Weighted Belief Fusion

Bibliography

[1] ISO/SAE 21434:2021. Road vehicles Cybersecurity engineering. *ISO/TC 22/SC 32 Technical Standard*, 2021. https://www.iso.org/standard/70918.html.

[2] The CONNECT Consortium. Architectural specification of connect trust assessment framework, operation and interaction. Deliverable D3.1, October 2023.

[3] The CONNECT Consortium. Conceptual architecture & customizable tee and attestation models specification. Deliverable D4.1, October 2023.

[4] The CONNECT Consortium. Operational landscape, requirements and reference architecture - initial version. Deliverable D2.1, October 2023.

[5] The CONNECT Consortium. Connect trust & risk assessment & cad twinning framework (initial version). Deliverable D3.2, 2024.

[6] The CONNECT Consortium. Integrated framework (first release) and use case analysis. Deliverable D6.1, November 2024.

[7] The CONNECT Consortium. Operational landscape, requirements and reference architecture - final version. Deliverable D2.2, December 2024.

[8] The CONNECT Consortium. Virtualization- and edge-based security and trust extensions (first release). Deliverable D4.2, January 2024.

[9] The CONNECT Consortium. Integrated framework (final release), use case evaluation and project impact assessment. Deliverable D6.2, August 2025.

[10] The CONNECT Consortium. Mec-enabled orchestrator, continuous authorization, trust management and dltbased secure information exchangeg. Deliverable D5.3, 2025.

[11] ISO/SAE. 21434 - road vehicles — cybersecurity engineering, 08 2021.

[12] Audun Jøsang. *Subjective logic*. Springer, 2016.

[13] Audun Jøsang, Stephen Marsh, and Simon Pope. Exploring different types of trust propagation. In *International Conference on Trust Management*, pages 179–192. Springer, 2006.

[14] Koffi Ismael Ouattara, Ana Petrovska, Artur Hermann, Nataša Trkulja, Theo Dimitrakos, and Frank Kargl. On subjective logic trust discount for referral paths. In *2024 27th International Conference on Information Fusion (FUSION)*, pages 1–8. IEEE, 2024.

[15] Red Hat. Drools: Business rule management system, 2025. Accessed: 2025-05-30.

[16] SAE International. Potential Failure Mode and Effects Analysis (FMEA) Including Design FMEA, Supplemental FMEA-MSR, and Process FMEA. Standard J1739_202101, SAE International, January 2021.