Design of Approaches for Dependability and Initial Prototypes

Antonia Bertolino, Antonello Calabro, Silvano Chiaradonna, Gabriele Costa, Felicita Di Giandomenico, Antinisca Di Marco, Mario Fusani, Fabrizio Grandoni, Valerie Issarny, Marta Kwiatkowska, et al.

▶ **To cite this version:**

Antonia Bertolino, Antonello Calabro, Silvano Chiaradonna, Gabriele Costa, Felicita Di Giandomenico, et al.. Design of Approaches for Dependability and Initial Prototypes. [Research Report] 2011. inria-00584920

HAL Id: inria-00584920

https://inria.hal.science/inria-00584920v1

Submitted on 2 May 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

CONNECT

Emergent Connectors for

Eternal Software Intensive Networked Systems

ICT FET IP Project

Deliverable D5.2

Design of Approaches for Dependability and Initial Prototypes

SEVENTH FRAMEWORK PROGRAMME

http://www.connect-forever.eu

INRIA docomo DOCOMO Euro-Labs LANCASTER UNIVERSITY THALES Universita' dell'Aquila technische universität dortmund UNIVERSITY OF OXFORD UPPSALA UNIVERSITET

Project Number	:	231167
Project Title	:	CONNECT – Emergent Connectors for Eternal Software Intensive Networked Systems
Deliverable Type	:	Report, Prototype

Deliverable Number	:	D5.2
Title of Deliverable	:	Design of Approaches for Dependability and Initial Prototypes
Nature of Deliverable	:	Report, Prototype
Dissemination Level	:	PU
Internal Version Number	:	0.8
Contractual Delivery Date	:	31 January 2011
Actual Delivery Date	:	17 February 2011
Contributing WPs	:	WP5
Editor(s)	:	Antonia Bertolino
Author(s)	:	Antonia Bertolino, Antonello Calabró, Silvano Chiaradonna, Gabriele Costa, Felicita Di Giandomenico, Antinisca Di Marco, Mario Fusani, Fabrizio Grandoni, Valerie Issarny, Marta Kwiatkowska, Eda Marchetti, Fabio Martinelli, Marco Martinucci, Paolo Masci, Ilaria Matteucci, Hongyang Qu, Rachid Saadi, Antonino Sabetta, Anna Vaccarelli
Reviewer(s)	:	Valerie Issarny, Bernhard Steffen

Abstract

The aim of CONNECT *is to achieve universal interoperability between heterogeneous Networked Systems. For this, the non-functional properties required at each side of the connection going to be established must be fulfilled. By the one inclusive term "*CONNECT*ability" we comprehend properties belonging to all four non-functional concerns of interest for* CONNECT*, namely dependability, performance, security and trust. We model such properties in conformance with a meta-model which establishes the relevant concepts and their relations. Then, building on the conceptual models proposed in the first year in Deliverable D5.1, in this document we present the approaches developed for assuring* CONNECT*ability both at synthesis time and at runtime. The contributions include:*

- *the Dependability&Performance analysis Enabler, for which we release a modular architecture supporting stochastic verification and state-based analysis;*
- *incremental verification and event-based monitoring for runtime analysis;*
- *a model-based approach to interoperable trust management;*
- *the Security-by-Contract-with-Trust framework, which guarantees and enforces the expected trust levels and security policies.*

Examples of the properties on which the above contributions are discussed are excerpted from the Terrorist Alert scenario.

Keyword List

CONNECT*ability,* CONNECT*ed System,* CONNECT*or, Dependability, Enabler, Generic Metrics, Metric Refinement, Monitoring, Networked System, Performance, Security, Security Policy, Security-by-Contract, State-Based Stochastic Methods, Stochastic Model Checking, Trust*

Document History

Version	Type of Change	Author(s)
0.1	Proposed template and TOC	A. Bertolino
0.2	First contributions to chapters by all contributors	All
0.3	Complete draft for SAC meeting (Terrorist Alert scenario)	All
0.4	Revised complete draft for internal review	All
0.5	Minor corrections	A. Bertolino
0.6	Revision after internal review by Steffen	All
0.7	Revision after internal review by Issarny	All
0.8	Final editing	A. Bertolino

Document Review

Date	Version	Reviewer	Comment
20/01/2011	V 0.5	B. Steffen - TUDo	Highlight better Y2 contributions, point at future work, plus several presentation issues
10/02/2011	V 0.6	V. Issarny - INRIA	Overall integration and consistency, formatting matters

Table of Contents

List of Figures

List of Tables

List of Acronyms

CEP Complex Event Processor

CMF CONNECT Metrics Framework

CPMM CONNECT Property Meta-Model

CSL Continuous Stochastic Logic

CTMC Continuous-Time Markov chain

DePer Dependability&Performance Analysis Enabler

DTMC Discrete-Time Markov chain

EMF Eclipse Modeling Framework

GLIMPSE Generic fLexIble Monitoring based on a Publish-Subscribe infrastructurE

JMS Java Message Service

LTL Linear Temporal Logic

LTS Labelled Transition System

MEC Maximal End Component

MDP Markov Decision Process

NS Networked System

PCTL Probabilistic Computational Tree Logic

QoS Quality of Service

RTML Role-based Trust Management Language

SAN Stochastic Activity Network

SCC Strongly Connected Component

SxCxT Security-by-Contract-with-Trust

TG Trust Graph

V&V Verification and Validation

XML eXtensible Markup Language

WP Work Package

1 Introduction

This document reports the results achieved in the second year of CONNECT within Workpackage 5 (WP5) on "Dependability Assurance". This is a broad WP encompassing dependability analysis and verification, as well as security, privacy and trust management within the CONNECTed world. In this chapter we first recall briefly the WP objectives (Section 1.1) and the conceptual models developed in Deliverable D5.1(Section 1.2). We then revisit the main comments received at the first year review and summarize how we addressed them (Section 1.3). Hence (Section 1.4), we present the pragmatic directions undertaken by the WP as a whole to make effective progress towards ensuring and assessing the dependability, security, privacy and trust of the CONNECTed systems. Finally, we provide an overview of the different complementary approaches and results produced in the second year (Section 1.5), which are then detailed in the subsequent chapters, and give a roadmap of this deliverable (Section 1.6).

1.1 The Role of Workpackage WP5

The CONNECT project aims at overcoming the interoperability barriers among heterogenous communication technologies by synthesizing suitable ad hoc CONNECTors between NSs which have expressed the intent to interact with each other. A synthesized CONNECTor should ensure that two NSs using different application protocols and middleware can each seamlessly follow their own language and procedures in the communication and yet be able to understand each other and successfully complete a collaborative transaction.

The position of CONNECT is to set the minimal possible assumptions on the NSs and to have the CONNECT Enablers automatically and autonomously derive all the required information for the synthesis and deployment of CONNECTors. There are several outstanding challenges behind such a position. Laying down the foundations of a compositional modeling and reasoning framework in which CONNECTors properties can be expressed and verified is the scope of WP2 [29]. Inferring via learning techniques the interaction behaviour of the NSs for completing their a priori functional and non-functional description, to be used in the CONNECTor synthesis, is the task of WP4 [31], whereas the synthesis of CONNECTors mediating the interactions between NSs at both application- and middleware-layers is pursued in WP3 [30]. The task of WP5 in such a context is to ensure that the interaction through the synthesized CONNECTors fulfills possible non-functional requirements, including dependability, performance, security and privacy properties, whilst also providing the involved NS at each end with the adequate trust level. Moreover, as the aim of CONNECT is to ensure *eternal interoperability*, notwithstanding component replacements or technology evolution, the assessment and assurance of such properties must continue at runtime beyond the deployment of the CONNECTors.

As described in the DOW, WP5 activity is structured into the following four tasks:

Task 5.1. Dependability metrics for open dynamic systems: the aim is to revisit classical dependability metrics to account for dynamic CONNECTions in open, evolutionary networks and elicit relevant properties to be ensured.

Task 5.2. Dependability verification & validation (V&V) in evolving, adaptive contexts: this will develop approaches for quantitative verification of dependability properties, and for lightweight adaptive monitoring that is meant to detect potential problems and provide feedback to learning and synthesis activities.

Task 5.3. Security and privacy: this will adapt and extend existing techniques for security-by-contract checking and enforcement.

Task 5.4. Distributed trust management: this will develop a unifying theory for trust and a corresponding reputation scheme.

1.2 Summary of D5.1 Achievements

As the objectives of this WP are quite broad, in the first year an intense activity has been carried out for reciprocal acquaintance between the differing backgrounds, terminologies and methods behind each task, involving the study of literature and exploration of how the various involved properties can fit together and

within the dynamic on-the-fly project vision. One challenge for the WP is in fact the need to put together several disparate approaches which address differing scientific challenges.

The Y1 achievements reported in Deliverable D5.1 [32] included:

- A conceptual framework for expressing CONNECT-relevant metrics for the properties of interest [16]: this consisted of a structured framework, in which traditional generic metrics for dependability, performance, security and trust were classified as CONNECT-specific and application-specific metrics, and could be applied to each of the four types of elements of the CONNECT architecture: the Enabler, the NS, the CONNECTor, the CONNECTed system.
- The evaluation and qualitative comparison of state-based stochastic methods and stochastic model checking for verification of CONNECT elements: the two approaches have been applied to a common case study, to better understand their respective synergies and differences. While formal verification approaches demonstrated to be very accurate in determining best and worst case behavior but for relatively small numbers of involved nodes, the state-based stochastic methods were able to provide average values for large-scale networks.
- A security model adapting the Security-by-Contract (SxC) paradigm to the CONNECT architecture: specifically, we investigated how SxC, originally developed for providing security assurances to mobile applications, could be used and adapted for guaranteeing the security of communicating heterogeneous NSs.
- A distributed trust management model for assessment of CONNECTors and CONNECT Enablers: the model allowed Enablers to estimate (also jointly with other Enablers) a measure of confidence on CONNECTors so to select among an available set the one which is the highest recommended, and to manage feedbacks to detect dysfunction and update stakeholders' trust relations.
- An analysis of the requirements for a flexible and generic monitoring framework, needed to support runtime analysis, and a preliminary discussion of how this should interact with Enablers for synthesis, learning, dependability analysis and security enforcement.

1.3 Responses to Y1 Review Comments

While acknowledging the broad range of activities carried out by WP5 in Y1, and agreeing that overall *deliverable D5.1 points into the right direction*, the reviewers also found some aspects in the WP5 approach to be confusing and consequently the Technical Review Report raised several questions to be addressed.

WP5 Scope and Focus As a first main point, it is said that *the scope and focus need to be specified more clearly*. Indeed, until the end of the first year the architecture and the overall approach of the project were still largely evolving and therefore it was not easy to grasp how dependability, security and trust aspects could fit into the larger CONNECT picture. We missed in fact important notions which should be fixed in the architecture to allow for building a dependability, security and trust infrastructure. We have since followed closely, and been also directly involved into, the advances made in WPs 1-4, especially concerning the refinement of the architecture, and hence at the end of the second year we are now able to more clearly outline WP5 scope and focus.

The key point to understand is that when in CONNECT a NS1 expresses an intent to communicate with another NS2, WP5 gets involved as far as such interaction involves some specified (be it given or learned) non-functional property. In the (remote) case that no dependability or performance or security or trust requirement is given, then evidently WP5 techniques become out of scope, as whatever the level of non-functional properties exposed by the transaction, this will always be acceptable (banally, it does not break any requirement). Such case seems quite strange, though. More realistically, considering the more or less stringent non-functional properties that an interaction has to satisfy, at the time that the communication via an on-the-fly CONNECTor has to be established, WP5 will analyse these properties for the CONNECTed system and will: *i)* provide feedback to the synthesis Enabler about whether the specified reqs are fulfilled and *ii)* possibly suggest mechanisms to be applied to the CONNECTor to improve the CONNECTed systems NF characteristics.

Specifically, focus of the WP will cover:

- Quantitative verification (e.g., mean time to repair) of properties expressed in temporal logic;
- Assessment of quantitative properties, like performance and dependability;
- Mechanisms/patterns to achieve such attributes;
- Formal assurance of security policies;
- Assessment of trust between NSs and of environment;
- Observation at runtime of complex events and of evolutions, relative to non-functional properties.

The approaches taken and the related tools which have been developed in the second year will be the subject of this deliverable.

Properties vs. Metrics Another request for clarification involved the conceptual metrics framework, in particular our use of terminology. It is noticed that not all non-functional properties which we considered are *metrics in the formal meaning of the word*. This comment correctly highlighted an improper use in D5.1 of the term metrics, and more in general hints at a need for a more rigorous modeling of the non-functional properties. We have addressed both aspects in Y2 activity via the derivation of a conceptual model for properties, both qualitative an quantitative, and for the metrics associated to the quantitative ones. In simple words, a non-functional property provides the general definition (not necessarily quantitative) of an attribute of relevance: more precisely we distinguish between prescriptive properties (must be guaranteed) and descriptive properties (provided by the NS). Metrics specify how a considered quantitative property is measured. To address such issue, and more in general to specify the conceptual framework, which was outlined during Y1, in a more rigorous and formalized way, we have defined a meta-model of CONNECTability properties (Chapter 3 of this deliverable) and derived from it as an example the model of a set of properties for a demonstrative scenario (see next chapter).

Impact on Synthesis Reviewers *would like to see how non-functional properties (desired for* CONNECT*ors and provided by components) are taken into account in the synthesis algorithms*. As noticed in the review report, monitor-based property enforcement is the approach followed for security. Concerning instead dependability and performance related properties, indeed, the approach is described in Chapter 4: we describe the architecture of the Dependability&Performance Enabler and discuss the process of interaction between synthesis and analysis (see in particular Figure 4.1).

Other more detailed comments included:

the term "dependability" is used with two different connotations. This was done on purpose: we had adopted formerly "Dependability" (capitalised) for the general concept encompassing all the relevant non-functional properties addressed in CONNECT, and "dependability" for the standard Laprie's terminology. Following the reviewers concern that this might lead to confusion, as shown in Figure 3.1 (to be considered as replacing former Figure 2.1 in D5.1) we have now introduced a specific term in substitution of capitalised "Dependability", that is "CONNECTability".

It is not clear where "security" relates to the infrastructure and where to the application itself: in order to cope with this comment, we have re-investigated the threat models of CONNECT. We have identified two main levels: the CONNECT infrastructure level, consisting of the complete architecture of the enablers, and the Networked System level, consisting of the entities that ask for the communication. Hence, we have to investigate security aspects among enablers of the CONNECT infrastructure, among the CONNECT infrastructure itself and the NSs level, and security aspects of the CONNECTed system, i.e., the system composed by the NSs and the CONNECTor (Chapter 7, Table 7.1).

It is not clear how the intended correctness-by-construction will be achieved. "Correctness-by-construction" refers to functional interoperability, which is in the scopus of WP2 and WP3. Non-functional properties are not achieved by construction, they are assessed at synthesis time and monitored at runtime.

1.4 Y2 Tactic: a Concrete Bottom-up Approach

Following the key position in the project that minimal assumptions must be made, during the first year we have tried to take the most generally possible approach. In retrospect, from the review comments we have learned that this attitude is not successful as in the end we are left with a very abstract vision which results at best confusing, at worst inconsistent.

In the second year, taking a start from the models and general concepts developed in Y1, therefore, we decided to make progress by taking a concrete bottom-up approach. By this we mean that we select a scenario and on this we apply our analysis, simulating the interaction with the other enablers. In this way, we hope to provide a clearer understanding of the approaches, and at the same time to get concrete feedback about their feasibility and suitability.

We hence proceed as follows. We first select one common scenario for demonstration of all WP5 approaches. On this, we proceed by selecting some non-functional properties, covering all the four main components of WP5, namely: dependability, performance, security and trust.

For each property we implement the corresponding approach and demonstrate how it works. As D5.2 has double nature of report and prototype, this pragmatic approach suitably fits our needs, as we can thus show how the developed prototypes work, even though the integration with the other CONNECT Enablers is still ongoing. At the time of writing, in fact, the other enablers are themselves undergoing development. It is our aim to proceed with real integration of WP5 Enablers with the other CONNECT Enablers in the third year.

1.5 Y2 Challenges and Overview of Related Achievements

Building on the conceptual models we developed in the first year, the challenges addressed in Y2 can be shortly summarized by the following questions:

C1: how can we abstract and express the non-functional properties we outlined in the Y1 CONNECT framework so that they are: general enough to match exigencies from heterogenous NSs and to be understood by CONNECT Enablers, but also precise enough for automated processing?

C2: how can we usefully employ within the CONNECT architecture the approaches for Dependability and Performance analysis that we experimented in Y1, both when a CONNECTor has to be synthesized and at runtime?

C3: how can we improve the CONNECT trust model, so to be able to handle and combine heterogenous trust models by different NSs?

C4: how can we release some strict assumptions behind the Security-by-Contract framework experimented in Y1, so to better capture the evolving CONNECT vision?

In this deliverable we report our achievements in addressing the above challenges.

Challenge C1 We have first introduced a new concept, that is CONNECTability, to better clarify and express in comprehensive way the set of non-functional properties considered in CONNECT. We have also defined a formal property meta-model that allows the specification of CONNECTability properties in a machine processable way. This meta-model aims to be the exchange language used among all CONNECT Enablers to communicate non-functional information (such as, the learned NS characteristics or the required properties for a CONNECTed system). Hence, all the CONNECT infrastructure will be able to understand and hence dynamically manage properties defined with this language. In Chapter 3, we show examples of how specific CONNECTability properties of interest can be instantiated from the defined meta-model.

Challenge C2 From the dependability and performance analysis point of view, the major contributions of Y2 consisted in: i) the definition of the architecture of a Dependability&Performance Analysis Enabler to assess, before deployment, if the dependability and performance requirements requested by the NSs can be satisfied by a CONNECTor being synthesised. It supports both stochastic model checking and state-based stochastic methods evaluation approaches; ii) preliminary prototypes for the two approaches, based on PRISM and Möbius assessment tools respectively; iii) application to the "Terrorist Alert" scenario, through which the complementarity of the two approaches has been shown. These achievements are described in detail in Chapter 4.

Concerning Dependability&Performance analysis at runtime, as we present in Chapter 5, we have two main achievements:

- an approach for quantitative incremental verification, which improves the performance of verification at run time in case probabilistic values are changed as time elapses. The key idea is to use a decomposition of the model into its strongly connected components (SCCs). Exploiting model structure in this way has already been shown to be effective for an isolated instance of probabilistic verification, but the benefits for reducing work across multiple verifications has not been considered. We proposed novel additional optimizations that can be applied when using an SCC-based analysis of a probabilistic model, incrementally or otherwise. This technique can be applied to both probabilistic formulae and reward formulae for modeling dependability properties. We implemented this technique for both explicit-state model checking and symbolic model checking in PRISM.

- the preliminary integration of the Dependability&Performance Enabler with the Monitoring Enabler, to allow for automated refinement of the analysis through on-line collected data. Indeed, to cope with potential evolutions of the NSs, the CONNECTor behaviour should be monitored routinely; during this year, the CONNECT Monitoring Enabler has been defined and implemented, using most up-to-date technologies (a prototype is now available for testing and is able to communicate with a minimum set of requisites). Some preliminary tests of the interplay between the Dependability & Performance Enabler and the monitor have been performed to evaluate the potential benefits of runtime analyses.

Challenge C3 Starting from the conceptual model for Trust introduced in the first year, in this second year we assume that CONNECTors and Enablers are trusted (which is reasonable from the CONNECT perspective) and specifically focus on trust management among heterogeneous NSs. We developed a formal trust meta-model by which we can describe any given trust model and infer composite trust models for heterogeneous NSs. There has been a proliferation of proprietary trust and reputation systems, which implement custom trust models regarding the specific stakeholders and their trust relations. As a result, the trustworthiness of a subject can be now easily determined under its associated proprietary model; however, assessing the trustworthiness of one subject under another subject's model requires overcoming trust models heterogeneity (e.g., interpretation of trust relations). This is the target of the CONNECT Trust Enabler introduced in Chapter 6.

Challenge C4 Taking advantage of the study of the threat models of the CONNECT scenario, during this second year we have proposed the new Security-by-Contract-with-Trust (SxCxT) paradigm under new assumptions with respect to those of Y1. In this way we showed that the approach can be used for guaranteeing security in wider scenarios. Furthermore, during Y2 we have integrated a trust model into the paradigm. Indeed, the Security-by-Contract was provided for guaranteeing the security of communicating systems composed of several, heterogeneous components through run-time enforcement mechanisms and active monitoring on the contract of the CONNECTor. The integration of the Trust model in the paradigm drives the application of the enforcement mechanism and/or the contract monitoring. This means that, according to the trust level of the considered Enabler, each NS decides whether to apply the contract monitoring or the enforcement mechanism or both. We have developed a prototype of the SxCxT mechanism. Another research direction we investigated during Y2 is about access control policy negotiation. We presented a trust negotiation framework for quantitative notions of trust based on soft-constraints, which will be the basis for subsequent work on automated trust negotiation among NSs.

1.6 This Deliverable

This deliverable is structured as follows: in the next chapter we give an excerpt of the Terrorist Alert scenario, fully developed in Deliverable D6.2 [35], on which we demonstrate all WP5 approaches. In Chapter 3, we describe the Property Meta-Model and instantiate a set of considered properties from it. Then, in Chapter 4 we present the architecture and the functioning of the Dependability&Performance analysis enabler, and in Chapter 5 its two ramifications at runtime, namely incremental verification and monitoring. The document proceeds in Chapter 6 with the proposal of a generic trust model, to support interoperability among NSs using different models, and then in Chapter 7 we finally illustrate advances of the Security-by-contract-with-trust paradigm, and a novel approach for access control policy negotiation. Conclusions are finally drawn in Chapter 8.

2 WP5 Scenario

In this chapter we develop the scenario that will be used throughout this deliverable for demonstration of the WP5 developed approaches. WP5 scenario extracts one situation from the CONNECT Terrorist Alert scenario (which is described in detail in Deliverable D6.2 [35]), and extends this excerpt concerning some aspects of specific interest to WP5. In Section 2.1 we briefly describe it, and then in 2.2 we define the synthesized CONNECTor. In Section 2.3 we introduce the relevant fragments of that scenario which we will refer to for dependability, performance, security and trust assessment.

2.1 The Terrorist Alert Scenario

With reference to D6.2, the Terrorist Alert scenario considers the case that during the show the stadium control center spots one suspect terrorist moving around. The alarm is immediately sent to the Police.

Figure 2.1: Terrorist Alert Scenario: Photo of a Suspect is Distributed to Policemen

Policemen are equipped with ad hoc handheld devices which are connected to the Police control center to receive command and documents. Precisely, the policemen can share documents with the Police control center and with other policemen through a *SecuredFileSharing* application, for example a picture of a suspect terrorist.

Unfortunately, the suspect is put on alert from the police movements and tries to escape, evading the Stadium.

In such an emergency situation, there may be various cases in which CONNECT can be of help. As described in D6.2, the police could for example be directly put in connection with various surveillance systems in the zone to receive videos or pictures in their devices. We focus on the case that a policeman that sees the suspect running away can dynamically seek assistance to capture him from civilians serving as private security guards in the zone of interest. To get help in following the moves of the escaping terrorist and capturing him, the policeman sends to the civilian guards an alert message in which one picture of the suspect is distributed.

On their side, to perform their service, the guards are equipped with smart radio transmitters which run an *EmergencyCall* application. This transmission follows a two steps protocol. We assume in fact that the guards that control a zone are CONNECTed in groups, and that for each group there is a Commander on duty. The protocol followed in the *EmergencyCall* application is that a request message is first sent from the guards control center to the Commander. As soon as the Commander replies with an acknowledgement of receipt, a message with details of the emergency is forwarded to all security guards. On correct receipt of the alert, each guard's device automatically sends an ack to the control center.

The two applications, *SecuredFileSharing* and *EmergencyCall*, in this scenario represent the two Networked Systems, which are not a priori compatible; hence a CONNECTor bridging between the policeman device and the guard device must be deployed.

In the following we show the LTSs modeling the two applications above mentioned.

SecuredFileSharing

- The peer that initiates the communication (hereafter denominated the *coordinator*) sends a broadcast message (`selectArea`) to selected peers (the Police control center or policemen) operating in a specified area of interest. In the SecuredFileSharing application, the coordinator can be either the Police control center or a policeman.
- The selected peers reply with an `areaSelected` message.
- The coordinator sends an `uploadData` message to transmit confidential data to the selected peers.
- Each selected peer automatically notifies the coordinator with an `uploadSuccess` message when the data have been successfully received.

The application behaviour, in the case the policeman acts as coordinator, is depicted in Figure 2.2. The LTS is shown in Figure 2.3.

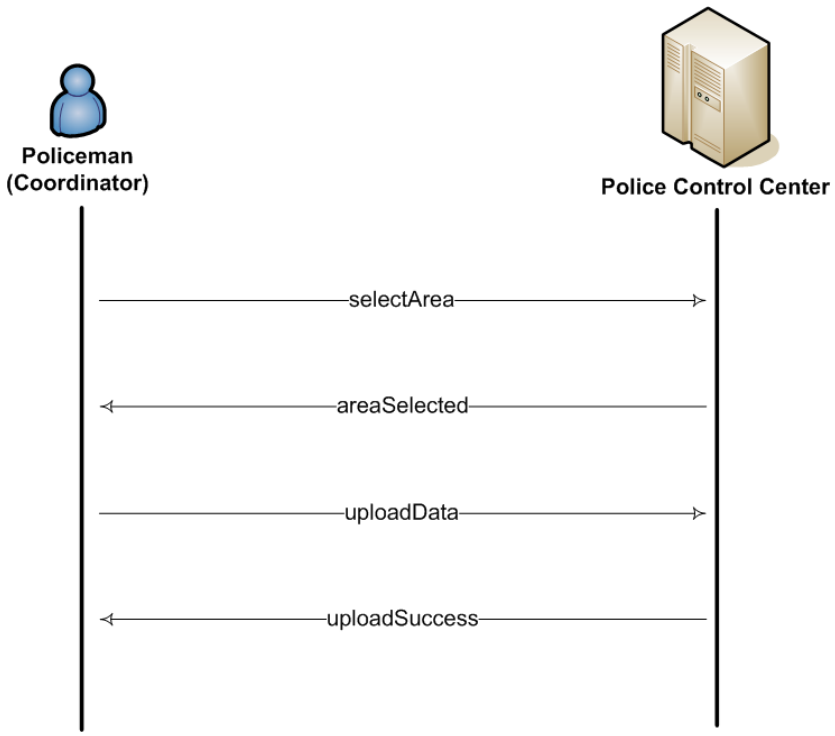


Figure 2.2: Sequence Diagram of the *SecuredFileSharing* Application

EmergencyCall

- The guards control center sends an `eReq` message to the commanders of the patrolling groups operating in a given area of interest.
- The commanders reply with an `eResp` message.
- The guards control center sends an `emergencyAlert` message to all guards of the patrolling groups; the message reports the alert details.

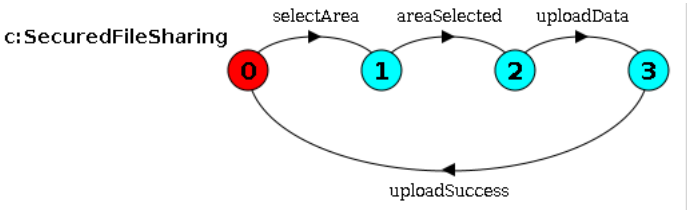


Figure 2.3: LTS of the *SecuredFileSharing* Application

- Each guard's device automatically notifies the guards control center with an `eACK` message when the data has been successfully received and a timeout is triggered after a time interval if not all guards sends back the `eAck` message. The timeout represents the maximum time that the CONNECTor can wait for the `eAck` message from the guards.

The application behaviour is depicted in Figure 2.4. The LTS of the commander is shown in Figure 2.5(a), the LTS of the other guards is shown in Figure 2.5(b).

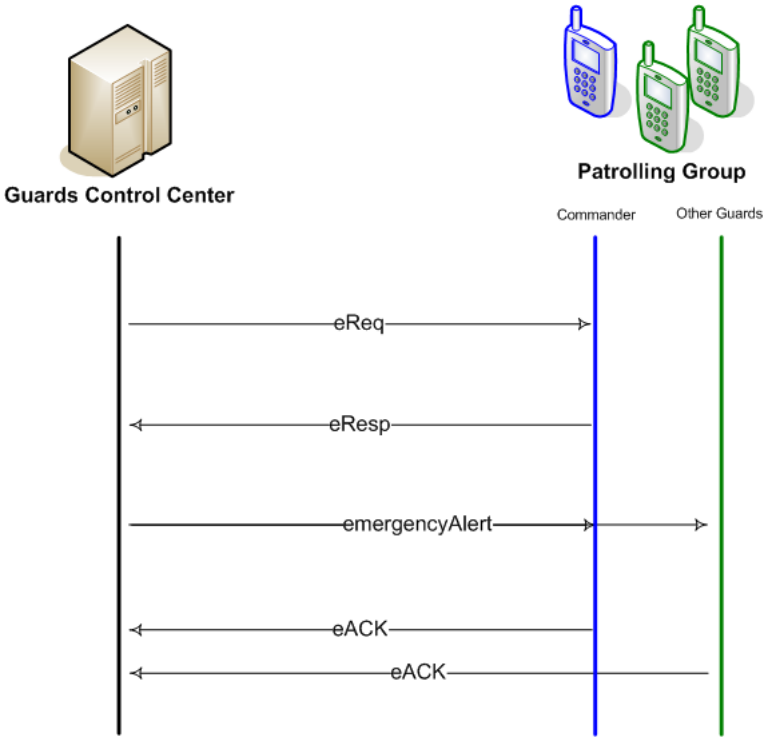
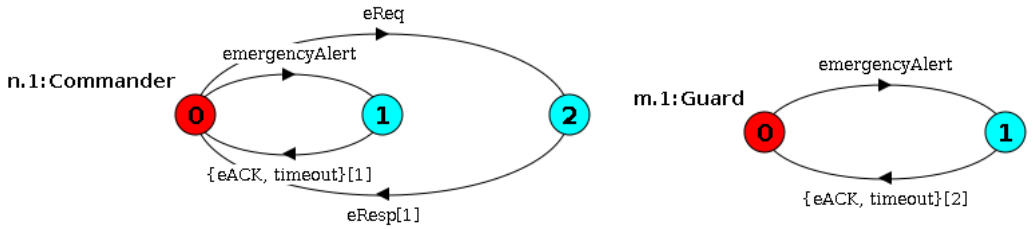


Figure 2.4: Sequence Diagram of the *EmergencyCall* Application

2.2 CONNECTing a Policeman and the Guards

As said, in the Terrorist Alert scenario, we need to synthesize on-the-fly a CONNECTor to allow a Policeman and the guards in the zone where the suspect has escaped to communicate. Precisely, we need to mediate between the two LTSs shown in Figures 2.4 and 2.5, respectively. We briefly summarise the needed mappings below.

CONNECTor



(a) Commander. (b) Other guards.

Figure 2.5: LTSs of the *EmergencyCall* Application

- The `selectArea` message of the policeman is translated into an `eReq` message directed to the commander of the patrolling group operating in the area of interest.
- The `eResp` message of the commander is translated into an `areaSelected` message for the policeman.
- The `uploadData` message of the policeman is translated into a multicast `emergencyAlert` message.
- The `eACK` messages automatically sent by the guards' devices that correctly receive the `emergencyAlert` message are collected and then translated into a single `uploadSuccess` message for the policeman.

The LTS of the CONNECTor in the case of a patrolling group consisting of one commander and two other guards is shown in Figure 2.6.

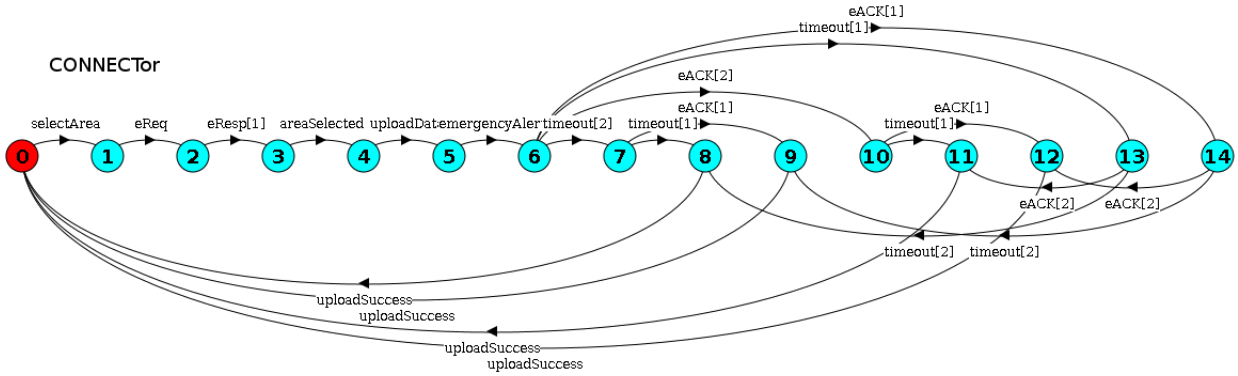


Figure 2.6: LTS of the CONNECTor

CONNECTed System

If we consider now the behaviour of the CONNECTed System in the case of one policeman and one patrolling group with one commander and two other guards, its LTS in our formalism coincides with that of the CONNECTor, as the communication that can take place between the NSs is only the one allowed by the synthesized CONNECTor. As the number of CONNECTed guards increase, the LTS also grows. Table 2.1 reports some statistics on the number of states and transitions generated for different number of guards.

WP5 approaches aim at investigating solutions able to deal with a varying number of guards, compatibly with the limitations imposed by the employed supporting tools.

#Commanders	#Guards	#States	#Transitions
1	1	15	22
1	2	33	68
1	4	249	848
1	8	10000	50418

Table 2.1: Number of States and Transitions for Different CONNECTed Systems

2.3 Properties and Metrics of Interest

In the above scenario, there are several non-functional requirements that the CONNECTion between the policeman and the guards must satisfy. We have selected a set of properties to be considered.

Dependability Belonging to Dependability attributes, we focus on the Delivery ratio, which sets a minimum coverage measure that the CONNECTed system must satisfy: it will be measured by *the percentage of guard devices that are reached by the alert message in a given time interval*.

Performance We focus on latency, and set the min/average/max *time of reaching a set percentage of guard devices* as the metric to be assessed.

Security We set two properties to be guaranteed, as follows: i) the photo sent to guards must be signed (for integrity and non-repudiation), and ii) the photo can only be received by authorized devices (for privacy). Both are qualitative properties. The first is mostly related to the possibility of the CONNECTor of performing cryptographic primitive and also depends of the synthesis strategy (as we discuss in WP3). The second one is a property of the CONNECTed system. In this deliverable we focus on this second security property.

Trust As an example, we may want to assess the reputation of security guards, and ensure that only guards above a set threshold will receive the alert. The trust enabler is still under development and will be released in the third year deliverable. Note that this trust property might interfere with the Delivery ratio metrics above defined, because it impacts the number of guard devices on which coverage has to be measured. We will consider the interesting issue of interaction between CONNECTability properties in the next year.

2.4 Conclusions and Further Research Directions

In this chapter we have briefly outlined a fragment of the Terrorist Alert scenario (fully described in Deliverable D6.2 [35]), as a common reference case study on which examples of applications of the following approaches will be developed. In the third year we will expand the demonstrations and validation of approaches to wider parts of the CONNECT scenarios.

3 The CONNECT Property Meta-Model

In this chapter we describe the structure of the meta-model we devised to specify properties in CONNECT, which we refer to as the CONNECT Property Meta-Model, or CPMM. In particular, the meta-model defines elements and types to specify prescriptive (required) and descriptive (owned) quantitative and qualitative properties that CONNECT actors may expose or must provide.

The reason of the definition of such a meta-model has to be retrieved in the dynamic nature of the CONNECT project. CONNECT aims at synthesizing automatically the CONNECTors through which Networked Systems communicate to reach a (common) goal. In order to achieve non-functional properties in the synthesis process and to feed the analysis process, prescriptive and descriptive properties must be expressed in a machine-processable language. The properties specified from this language might refer to:

- the CONNECTed system: the properties are part of a connection request and represent requirements for the CONNECTed system;
- the Networked Systems: NS properties could be exposed by the NSs themselves, or inferred by the Learning Enabler as a result of the learning phase.
- the CONNECTor: the specified properties can model requirements for the CONNECTor synthesis or can be observed by the monitoring Enabler during CONNECTor execution.
- the Enablers: the specified properties can also describe the characteristics owned by the Enablers. Mainly, the descriptive properties for the Enablers are related to trust and security. The former contains the trust models the Enabler is aware and is able to manage. The second ones instead refer to security mechanisms/policy the Enabler implements or can enforce. Such models can be incremental, since the Enabler can embed new models/mechanisms as required by the NSs.

The models conforming to such meta-model can be used: *i)* as input for the synthesis Enabler to guide the synthesis of a new CONNECTor or the selection process in case suitable CONNECTors are already available; *ii)* as input for the dependability Enabler to verify specified CONNECTability properties, and *iii)* as instrumentation for the monitoring Enabler that generates suitable probes to monitor useful properties on the CONNECTors.

As an example of using the CPMM, then we model the properties of interest for the Terrorist Alert Scenario outlined in the previous chapter. We recall that among several non-functional requirements that the CONNECTion between the policeman and the guards must satisfy, in Section 2.3 we focused on four properties (and, for quantitative ones, their related metrics) relative to Dependability, Performance, Security and Trust.

However, at the time of writing, the meta-model only partially deals with Trust since the Trust model definition is still under refinement (its current status is reported in Chapter 6). For this reason in the following we provide only the models for the coverage, latency and privacy, while concerning trust-related properties the meta-model will be completed in the next year.

The currently released version of the CPMM eCore model can be retrieved from the URL provided in the Appendix-Prototype associated with this report. We also make available an editor as an Eclipse Plugin, for deriving from CPMM instances of property models.

This chapter is organized as follows: in Section 3.1 we introduce the new term of CONNECTability to clarify and disambiguate our meaning of "Dependability" in CONNECT; in Section 3.2 we present the description of the devised meta-model and some related implementation details, and in Section 3.3 we overview related work; in Section 3.4 we outline the relationship between the presented meta-model and the CONNECT Metrics Framework introduced in Deliverable D5.1 [32]. From Section 3.5 to Section 3.7 we show the models of the performance, dependability and security properties we deal with for demonstration purposes. Finally, Section 3.8 draws conclusions and plans future works.

3.1 CONNECTability

One challenge WP5 is facing in CONNECT is that of addressing in unified way non-functional properties of CONNECTed Systems which belong to different attributes, namely Dependability, Performance, Security

and Trust. In making an attempt to deal with all such concerns altogether, a terminology issue arises, which has been also noticed at the Y1 review. WP5 is titled “Dependability assurance”, whereas in this title the term *dependability* was meant to include a broader consideration of relevant attributes for CONNECT than dependability in the stricter original Laprie’s definition [67], recently revised in [8].

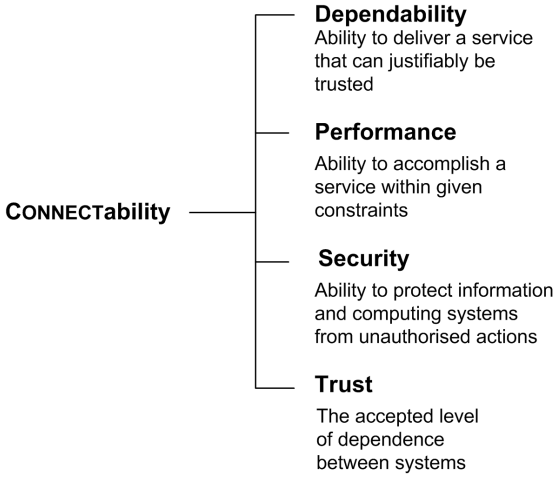


Figure 3.1: Introducing the Term CONNECTability

In Deliverable D5.1, we tried to use the term “dependability” with both senses, using capital D for the broader meaning. Following the reviewers concerns about confusion descending from this usage of a same term for two meanings, we resolved to introduce a new, ad hoc term, that is CONNECTability. As shown in Figure 3.1, CONNECTability includes all fours attributes of concern for CONNECT. Paraphrasing dependability, which is defined as the ability of a system to provide its intended services in a justifiable way, this term is meant to refer to *the ability of* CONNECT *to enable the intended (i.e., dependable, efficient, secure and trustful)* CONNECT*ion between Networked Systems in a justifiable way*. This is our ultimate goal, although we still have much work ahead of us to provide this!

3.2 CPMM Overview

The CPMM meta-model has been conceived as partitioned into six parts. In particular, we distinguished the *Property* meta-model, the *MetricsTemplate* meta-model, the *Metrics* meta-model, the *EventSet* and *EventType* meta-model and the *ApplicationDomain* meta-model.

Figure 3.2 reports the key concepts of the whole meta-model (framing within dashed lines the sub meta-model they belong to) and how they relate with each other. Note that the figure has been generated as an excerpt of the complete meta-model by visualizing the main concepts of it and removing several technical details.

As the figure shows, such main concepts are the *Property*, which can be *Quantitative* or *Qualitative*, the *MetricsTemplate* modeling a generic metric (e.g., latency as the duration of an event/operation e_1), the concrete *Metrics* that refers to a MetricsTemplate and specifies the (actual) metricParameter for the (formal) templateParameter in the particular application domain the metric has been defined. The specification of the concrete event in the metric is done via the concept the *EventType* the metric refers to. Such EventType refers to observable operations or events belonging to the ontology of the *Application Domain* for which the Property is relevant or it needs to be verified/guaranteed (by means of the *EventTypeSpecififcation*).

We want to highlight that CONNECT metrics could be very complex since they can be defined by combining several (classical) metrics. An additional issue to be addressed is that the terminology on which the metrics have to be defined comes from the application domain of the NSs that, in general, may change over time. Indeed, a connection request can possibly refer to different application domains. Taking inspiration from Monperrus et al. work [77], we separate the property definition from the application

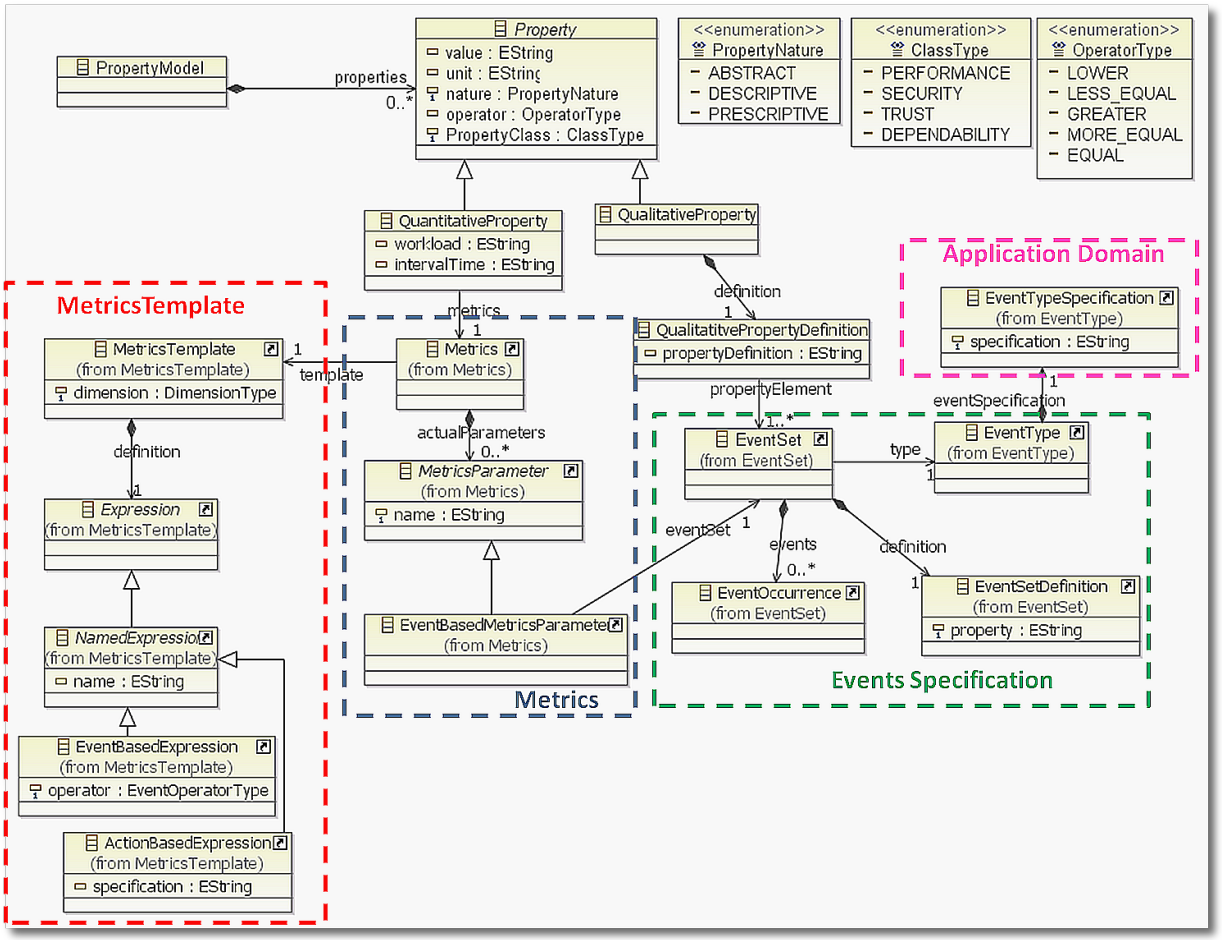


Figure 3.2: Key Concepts of the CONNECT Property Meta-Model

domain, and this is formalized in the meta-model by including the application domain meta-model derived from the ontology defined within the WP1 workpackage [33]. In the next year, the ontology will be linked in the Property meta-model via the EventTypeSpecification meta-class that will be refined to model a generic term in the ontology.

The concept of the MetricsTemplate instead models a general metric (simple or complex). Independently from the application domain, the definition of a metric is always the same, and precisely defines a template. For example the response time of a system is defined by the time needed by the system to execute a given task. This definition does not change over time. The variability stays in the object of the metric, that can be a (simple or complex) observable behavior of the application the property refers to. Such observable behavior is modelled by means of EventType. A metric fixes this variability pointing to the suitable MetricsTemplate model and specifying the application operation by associating the parametric operation defined in the template to the concrete operation in the application domain model.

In the following sections we complete the description of the proposed meta-model by giving details on the property meta-model (in Section 3.2.1); on the metrics and metric template meta-model (in Section 3.2.2); and finally on the events set and event type definition meta-model (in Section 3.2.3).

3.2.1 Property Definition

Figure 3.3 reports the portion of the meta-model describing a *Property*. A *PropertyModel* is composed by zero (modeling the empty model) or more *properties*.

A `Property` is a *NamedElement* having two required/mandatory attributes and three optional ones.

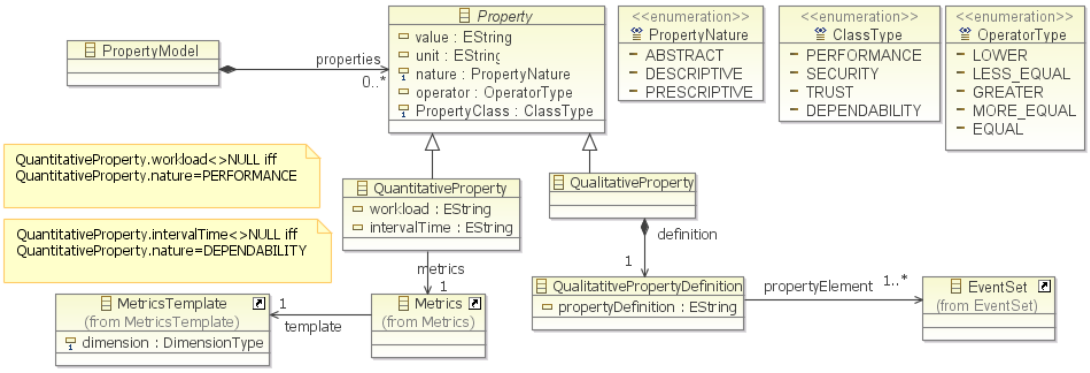


Figure 3.3: Property

The required attributes are: *nature* and *PropertyClass*. The former refers to the nature of the property that can be *ABSTRACT*, *DESCRIPTIVE*, or *PRESCRIPTIVE*. The latter can have the following values: `PERFORMANCE`, `SECURITY`, `TRUST`, and `DEPENDABILITY`.

An *ABSTRACT* property indicates a generic property (generic in that it does not specify a required or guaranteed value for an observable or measurable feature of a system).

A *DESCRIPTIVE* property represents a guaranteed/owned property of the system while a *PRESCRIPTIVE* one indicates a system requirement, then a required property in the system. In both cases, the property is defined taking into account a relational operator with a specified value.

The optional attributes of `Property` are *value*, *unit* and *operator*. These attributes are not specified in case of an *ABSTRACT* property because as described above an *ABSTRACT* property does not specify a relation with a specific value. They are specified only for the *DESCRIPTIVE* and *PRESCRIPTIVE* properties. The *value* attribute indicates a value associated to the property, the *unit* attribute indicates its unit measure, and the *operator* one models a relational *operator* (one of the operators listed in the *operatorType* enumeration).

A property can be qualitative (*QualitativeProperty*) or quantitative (*QuantitativeProperty*). The former models properties about the event occurrences of an *EventSet* that are observed and can generally not be measured. They in general refer to the behavioral description of the system (e.g., deadlock freeness or liveness). The quantitative properties are quantifiable/measurable and they have an associated *Metrics*.

The *QuantitativeProperty* has two optional attributes that are *workload* and *intervalTime*. The former must be specified for a property (*DESCRIPTIVE* or *PRESCRIPTIVE*) having a `PERFORMANCE` class while the latter must be specified for a property (*DESCRIPTIVE* or *PRESCRIPTIVE*) having a `DEPENDABILITY` class. Both are not specified for an *ABSTRACT* property.

To clarify the above concepts we report below some `Property` examples:

Property1: `Ability to provide a service according to given time requirements` (this property has been presented in D5.1 Figure 2.3 [32]). This is an *ABSTRACT* property since it does not deal with a claimed or guaranteed specified time feature. This property is also a *QuantitativeProperty* having a `PERFORMANCE` class because it is about a measurable performance dimension (time). In this case, the *value*, *unit* and *operator* attributes of *Property* and the *workload* and *intervalTime* ones of *QuantitativeProperty* are not specified.

Property2: `The system` S `in average responds in 3 ms in executing the` e_1 `operation with a workload of 10 concurrent` e_2 `operations.` This is a *DESCRIPTIVE* property asserting that the system S guarantees in average a time response having a value of 3 ms in executing the e_1 operation, with a workload of 10 e_2 concurrent operations. As *Property1*, this one is also a *QuantitativeProperty* having a `PERFORMANCE` class because it is about a measurable performance dimension (time). In this case, the *value* attribute is equal to 3, the *unit* measure is ms and the specified *operator* is *EQUAL*. The *workload* attribute is equal to 10 while *intervalTime* is not specified.

Property3: `The system S in average must respond in 3 ms in executing the` e_1 `operation with`

`a workload of 10` e_2 `operations.` This is a *PRESCRIPTIVE* property because it specifies a required time response with a value of 3 ms in executing the e_1 operation, as above with a workload of 10 e_2 concurrent operations. As *Property1* and *Property2*, this is also a *QuantitativeProperty* having a `PERFORMANCE` class. The *value* attribute, the *unit* measure, the specified *operator*, the *workload* and *intervalTime* attributes are equal to those of *Property2*.

In the above examples, operations S, e_1 and e_2 are application-dependent and their semantics can be given once the application domain has been fixed.

3.2.2 Metrics Meta-Model

The Metrics meta-model describes among the others two main concepts: the *MetricsTemplate* and the *Metrics*.

To understand the differences between them, let us make an example considering the response time metric. The response time of a system, for a given operation, is the time interval between submitting an operation request and receiving a response from the system. It can be defined as the duration of a complex operation or the difference between two time stamps of different simple atomic actions without duration. The previous definition provides two different ways to specify the response time. We call such specification *MetricsTemplate*. Hence a metric (response time) can refer to one or more (hopefully equivalent) specifications (MetricsTemplates). Moreover, the MetricsTemplate is defined upon a generic set of actions/events/operations, that is not coupled with a particular application domain. This general part of the definition is specialized by the metric that instantiates those general concepts (templateParameters) with application-based actions/events/operations (metricsParameters).

For example, with reference to the above mentioned two ways for measuring a response time, we have the two following MetricsTemplates:

- response time of the E operation = DURATION(E) where E is the operation of which we want to measure/determine the response time, and it is the templateParameter.
- response time of the E operation = timestamp(E2)-timestamp(E1) where E1 and E2 represents the starting and the ending action of E, respectively. E, E1 and E2 are templateParameters.

E, E1 and E2 are instantiated by the Metrics upon with the Property is specified for a specific (software) system.

Let us discuss, as an example, the Metrics in the Terrorist Alert Scenario presented in Chapter 3. Let us suppose that we want the response time of the system for the area selection task. This task starts with the selectArea action and ends with the selectedArea action, both executed by the ControlCenter. In this case, we define a new metric that would use the second MetricsTemplate definition and referring to selectArea and selectedArea to actualize the E1 and E2, respectively.

Figure 3.4 reports the meta-model portion describing the *MetricsTemplate* concept. A *MetricsTemplateModel* defines one or more *MetricsTemplate*s, each having a *dimension* indicating the type of the value defined by the metrics template (e.g., a TIMEd value, a PERCENTAGE, and so on). A *MetricsTemplate* is a *NamedElement* containing the *Expression* describing the mathematical definition of the template.

The *Expression* could be:

- a *MathExpression*, this meta-class allows the definition of a complex mathematical expression by nesting one or more *operands* that are in turn others Expression. The MathExpression has an attribute, *operator*, that specifies the *MathOperatorType*. At the moment we define as possible operator the ones in the *MathOperatorType* enumeration. This enumeration can be extended if necessary.
- a *QuantitativeProperty*, in this case the *Expression* can be a QuantitativeProperty already described. In this case to be used as operand of an expression, it must have a *DESCRIPTIVE nature* as set by the OCL constraint indicated in the note.
- a *Constant* that has a *value* attribute of EString type indicating the specific value the constant refers to.

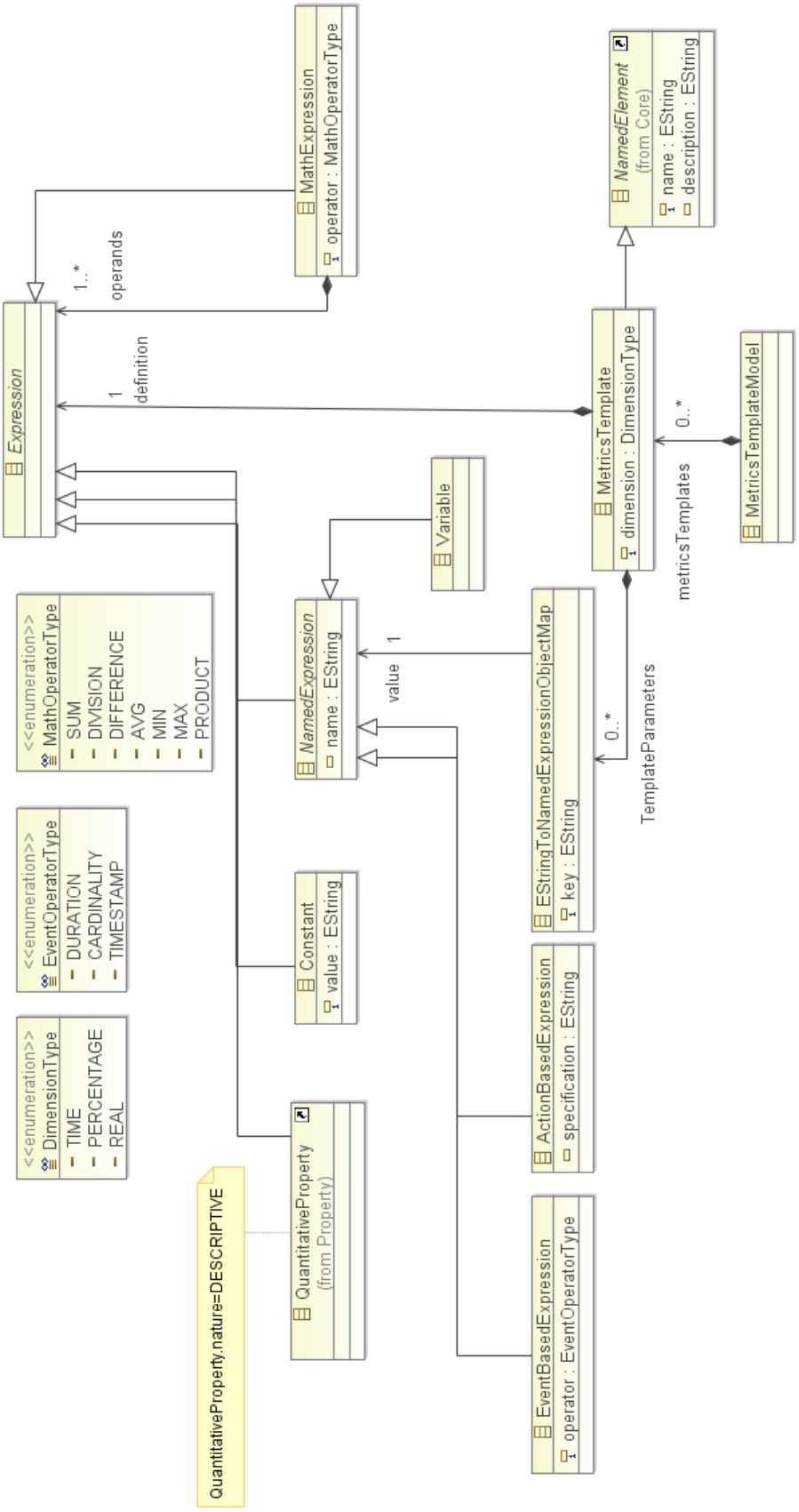


Figure 3.4: MetricsTemplate

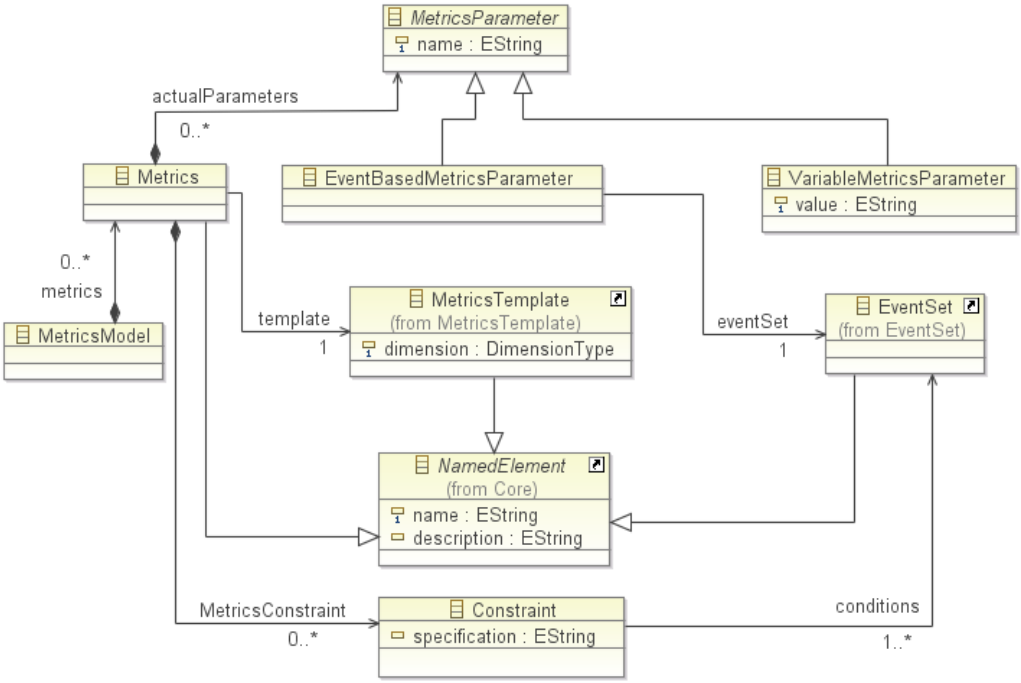


Figure 3.5: Metrics

- a *NamedExpression*. This element can represent:
  - a canonical *Variable*.
  - an *ActionBasedExpression*, that represents a simple action or a sequence of actions that, whenever has been executed, reports a value. An example could be a sort of get methods that report some information managed by some piece of software in the CONNECTed system or in the CONNECT infrastructure. The specification of an action based expression is contained in the *specification* attribute. At the moment this attribute is a simple string, but in future we aim at refine it by providing a suitable specification language that can be used in specifying action based expression.[1]
  - an *EventBasedExpression*, that represents expressions based on events or observational behavior. In this case the *name* represents the observable, simple or complex, event/behavior the *operator* applies. As first set of *EventOperatorType* we define *DURATION* for a complex event indicated in the name attribute, *TIMESTAMP* for a simple one, and *CARDINALITY* for the type of event indicated in name attribute. Whereas for the first and second operators they are applied to single occurrences of event type indicated, for the third one, instead, it is applied to the whole set of event occurrences observed in a given instant of time.

The MetricsTemplate finally contains zero or more *TemplateParameters*. These template parameters are EString *key*s exposed by the template and linked to NamedExpression by mean of the *EStringToNamedEspressionObjectMAp* concept.

Figure 3.5 reports the meta-model portion describing the *Metrics* concept. A *MetricsModel* defines zero or more *Metrics*. A metric is a *NamedElement* that refers to a *MetricsTemplate*, actualizes the *templateParameters* by means of the *MetricsParameters* (if the MetricsTemplate has no formal parameters, metric exposes zero MetricsParameters), and contains zero or more *Constraint*.

EventBasedMetricsParameter is a *MetricsParameter* actualizing the *EventBasedExpression* based *templateParameters*. To this goal, it refers to the *eventSet* describing the application based event definition

[1] This element has been required to express some operand of the Trust model. This part will be refined in the future, as soon as the Trust model is stable and validated.

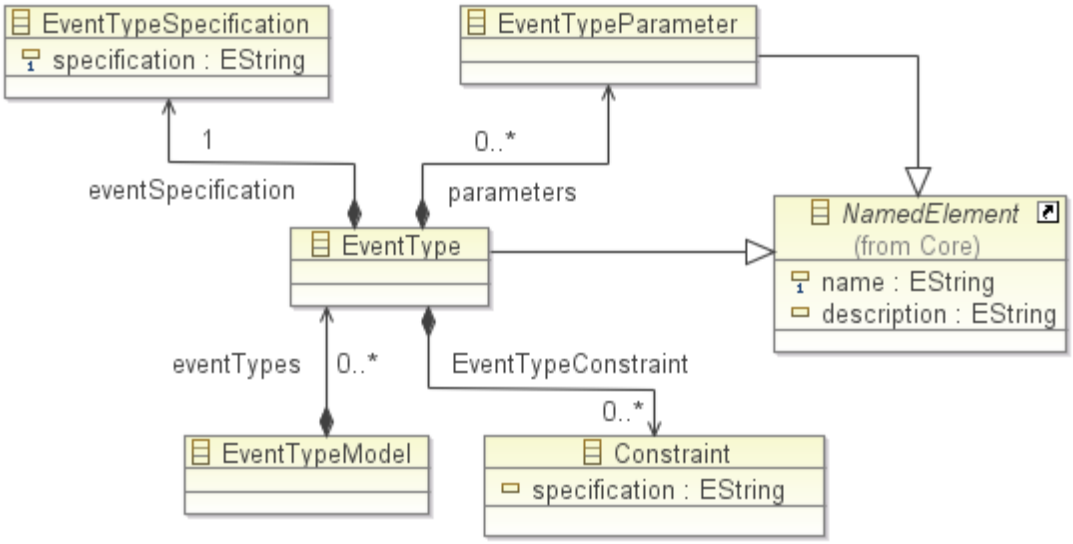


Figure 3.6: EventType

and the relative occurrences. *VariableMetricsParameters* instead is a *MetricsParameter* actualizing the *Variable templateParameters* indicating the corresponding *value*.

Finally, *Constraint* defines some condition (stored in the *specification* attribute) on the *EventSet* involved in the *Metrics* that must be satisfied. Such constraints in general allow to make a correspondence among event occurrences belonging to different types of EventSet in case the EventTypes in the EventSets have some EventTypeParameters. This type of constraint is needed to support the monitoring of the property. For example, a constraint could specify that in the derivation of a specified metric the identifier in an event must be the same as the identifier in another event.

3.2.3 Events Meta-Model

In the *Event Specification* meta-model we distinguished the *EventType* and the *EventSet* concepts. Figure 3.6 shows the portion of the meta-model describing the *EventTypeModel*. The *EventTypeModel* is composed by zero or more *EventType* elements, each one modeling the type of an event. The *EventType* models an observable system behavior that can be a primitive/simple event or operation representing the lowest observable system activity or a composite/complex event that is a combination of primitive and other composite events. An *EventType* has a specification identifying the type or class of the observable events. Such *EventTypeSpecification* belongs to the ontology of a specific application domain. In the current version of the meta-model this specification is simply defined by means of a label or string. However, in the future we plan to provide a more formal specification for complex event definition, according to an existing or a new defined event specification language. In [42, 73] some examples of complex event specification languages are presented. An *EventType* has one or more parameters and one or more constraints.

We present below some *EventType* examples including two generic primitive *EventType* and a composite *EventType* that is relating to their sequence:

- The primitive *EventType* named *SendRequest* indicating a generic request sending event with one parameter named *RequestID* that is the event identifier.
- The primitive *EventType* named *SendReplay* indicating a generic replay sending event with *ReplayID* event identifier parameter.
- The composite *EventType* named *SendRequestReplay* that represents the ordered sequence of the *SendRequest* event and the *SendReplay* one, both having the same identifier. This *EventType*

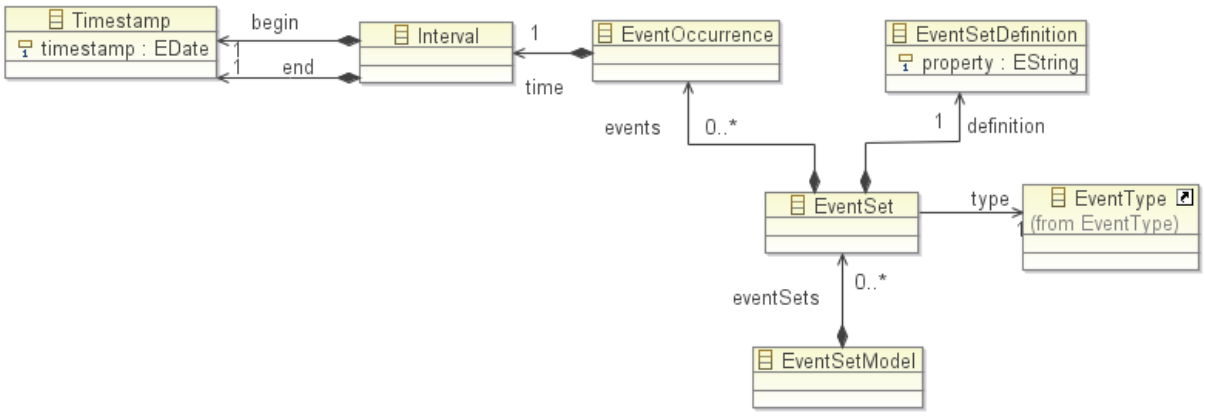


Figure 3.7: EventSet

represents a request event followed by a replay event to that request. This can be modeled by setting a specification indicating that *RequestID* and *ReplayID* parameters must have the same value. This *EventType* has a *RequestReplayID* parameter identifying the event type and a time constraint that can be for example of 5 s, meaning that the composite *SendRequestReplay* event is observable if it happens within a window of 5 seconds.

The specification of all above *EventType* examples is application-dependent and has to be consistent with the specific application domain.

Figure 3.7 reports the portion of the meta-model describing the *EventSetModel*. The *EventSetModel* has zero or more *EventSet*. An *EventSet* represents a set of event instances that refer to an *EventType*. The property of an *EventSet* identifies all observable events that have to be included in the *EventSet*. An *EventSet* has zero or more *EventOccurrence* representing the observable events that the *EventSet* contains. In the CONNECTed system this *EventOccurrence* can be generated at runtime by the CONNECT monitoring infrastructure when the probes observe the event of the *EventType* the *EventSet* refers to. An *EventOccurrence* refers to an *Interval* that represents the time range in which the observable event occurs. Each *Interval* has two associated *Timestamps* indicating its starting and ending date respectively. These *Timestamps* are equal in case of atomic/instantaneous event occurrences.

An example of *EventSet* is that containing all occurrences of the composite event with *SendRequestReplay EventType* defined above, detected in the last hour. The property of this *EventSet* specifies that it collects only the event occurrences with *SendRequestReplay EventType* generated at runtime in the last hour, then the starting and ending *Timestamps* of the time *Interval* that the *EventOccurrence* refer to are lower than the current time and this time difference is no longer than one hour.

The event occurrences collected into an *EventSet* define observable system behaviors modeled by qualitative properties or represent the parameters upon which the metrics are defined for measuring quantitative properties.

3.2.4 Meta-model Implementation Details

We used Eclipse Modeling Framework (EMF) [46, 68] for defining the meta-model sketched in Figure 3.2. With EMF it is possible to distinguish the meta-model from the actual model. The former describes the structure of the model, namely elements and types that can be defined, the latter is an instance of the meta-model, that contains specific attributes of the application scenario.

EMF is based on two meta-models: eCore and Genmodel. The eCore meta-model allows to mainly define: a class with zero or more attributes and zero or more operations, an attribute with a name and a type, a reference representing an association between two classes, and the type of an attribute, e.g., int, float, boolean. After specifying a meta-model, it is possible to generate the corresponding Java code that

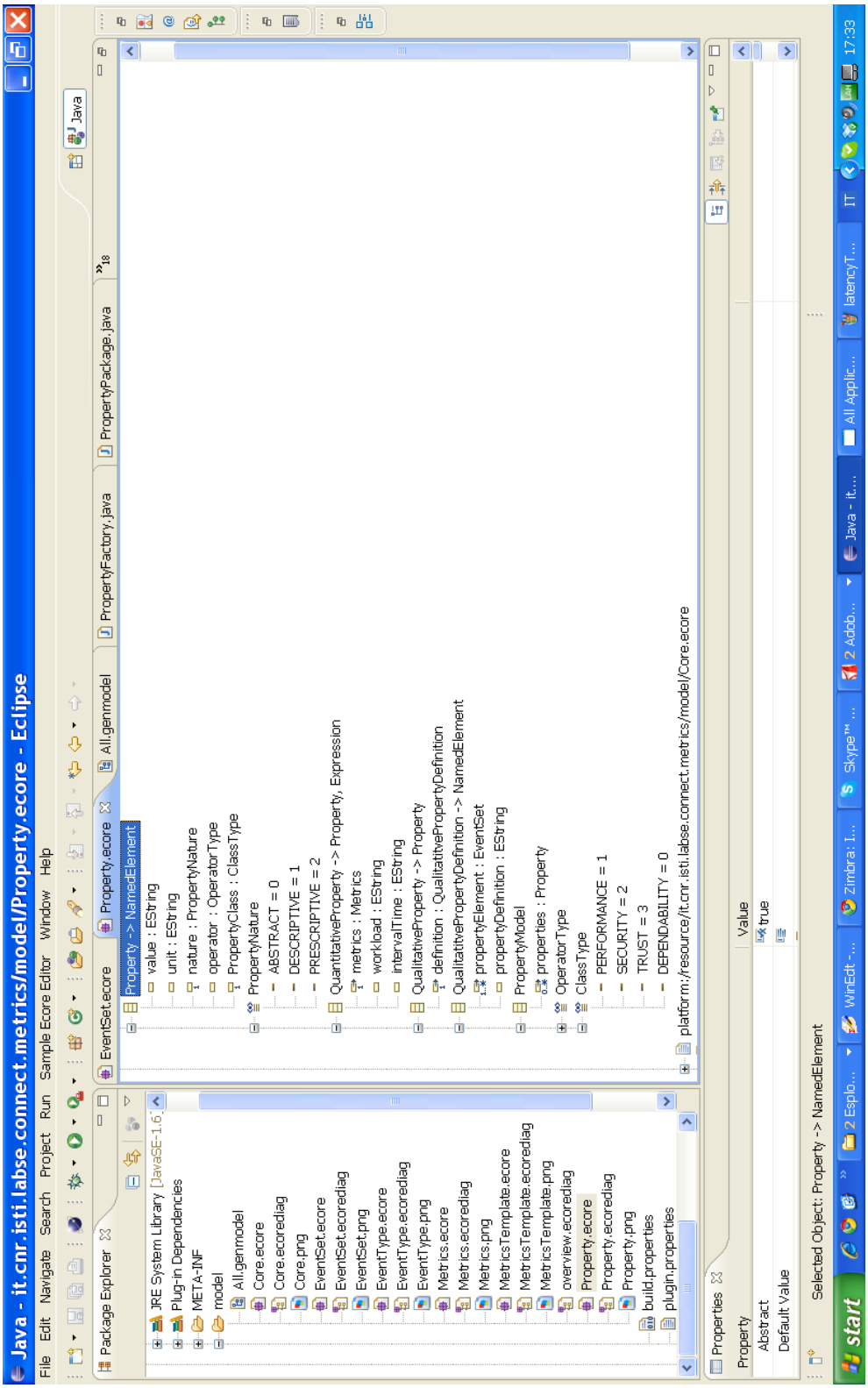


Figure 3.8: Property eCore Definition

can be safely extended by hand. The Genmodel contains additional information such as the path and file information and the control parameters for the code generation.

The domain of CPMM deals with the prescriptive and descriptive properties that the CONNECT actors expose or must provide. This meta-model has been generated as an eCore model into the Eclipse EMF framework, using facilities provided by the EMF visual editor. In particular, this meta-model has been partitioned in the following eCore models: *Core.ecore* representing a generic named element, *EventType.ecore* and *EventSet.ecore* modeling the event and the eventSet respectively, *Metrics.ecore* and *MetricsTemplate.ecore* for specifying the metrics and metricsTemplate concepts and finally the *Property.ecore* representing the Property meta-model.

All information about the classes defined into each eCore model are included into the corresponding modelname.ecore document. Figures 3.8, 3.9, and 3.10 show the eCore models of the Property, Metrics, and MetricsTemplate respectively, while Figure 3.11 reports the EventType and EventSet eCore definitions. As shown in these figures, the root object of each eCore model represents the whole model containing the defined packages whose children represent the classes, while the children of the classes represent their attributes.

From the above defined eCore models, by means of the EMF Generator model, a Genmodel model has been created and starting from it the Java code for that model has been generated. Starting from the Genmodel model, a tree-view based editor has been obtained as an Eclipse Plugin. This editor contains the information of the defined eCore models and allows to create new model instances of the Property, Metrics, MetricsTemplate, EventType and EventSet meta-models. This editor has been used for deriving, from the meta-model of Figure 3.2, the models of latency, coverage and privacy properties for the Terrorist Alert scenario.

3.3 Related Work

Meta-modeling is used in software engineering for specifying the abstract syntax of a language. The works more related to our proposal are those addressing the specification of meta-models and ontologies for defining software metrics and non functional properties of a system.

The main concept behind our proposal of specifying a metric as an instance of a metric specification meta-model is common to Monperrus et al.'s work [77], in which a generative model driven definition of software metrics is proposed. This work concerns the definition of a *domain-independent metrics meta-model*, that allows modelers to automatically add measurement capabilities to a domain specific modeling language. This applies to kind of models devised during a model-driven development process. Hence, the domain here is related to development phases (such as software architecture, requirements or implementation). Taking inspiration from Monperrus et al.'s work, CPMM separates the property definition from the application domain, and this is formalized in the meta-model by including the application domain meta-model derived from the ontology defined within the WP1 work [33]. Instead, differently from [77], the CPMM meta-model addresses specifically *non-functional property and metrics*. Indeed, it introduces additional concepts concerning the qualitative and quantitative properties definition, the events modeling and the distinction between a generic metric (represented by a MetricsTemplate) and the concrete metric (i.e., the Metrics concept) instantiated to a specific application domain, represented by the domain of the software system the CPMM is used for.

In literature many works can be found concerning meta-modeling that focus on domain-specific metrics or specifically on dependability properties. Hence, for different reasons, these approaches propose partially what CPMM proposes as a whole. Examples of the former set, [91] and [12] report two attempts of using more specific meta-models for defining object-oriented design metrics and database design metrics have been respectively presented. Other works, such as the ones described in [85, 44, 64, 57], focus on modeling only dependability properties. The authors of [85] propose a functional and non functional modeling and analysis framework that extends UML meta-models, conceived for specifying functional properties, checked by existing modeling tools, to cover non functional properties including dependability attributes.The main idea is to complete the UML meta-model with additional meta-model elements for the description of fault effects and the mapping of the UML system models to a mathematical analysis domain, allowing an automated derivation of the transformation rules. Another attempt towards the definition of a formal meta-model, structuring the notation for dependability cases (assuring that a system is acceptably

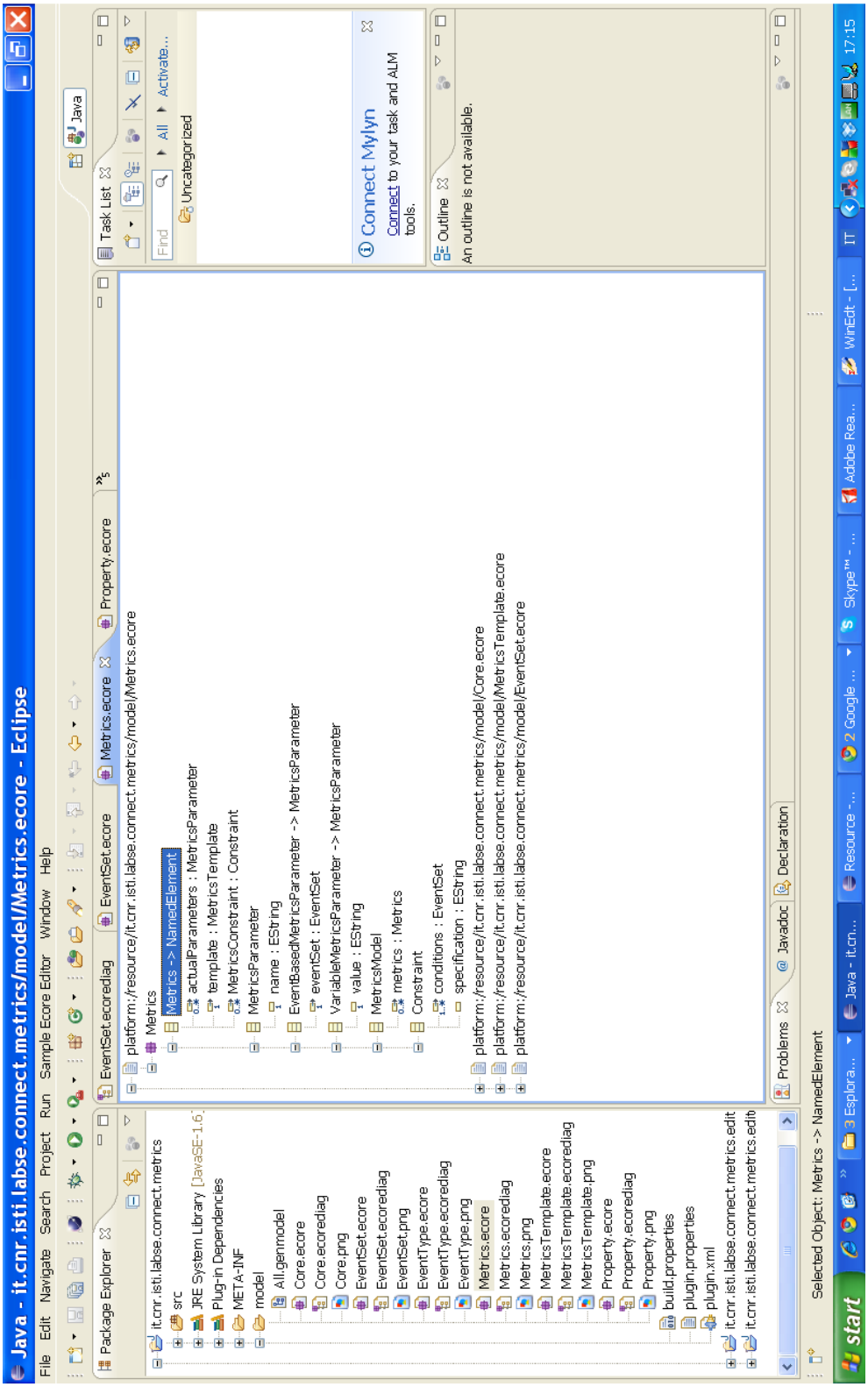


Figure 3.9: Metrics eCore Definition

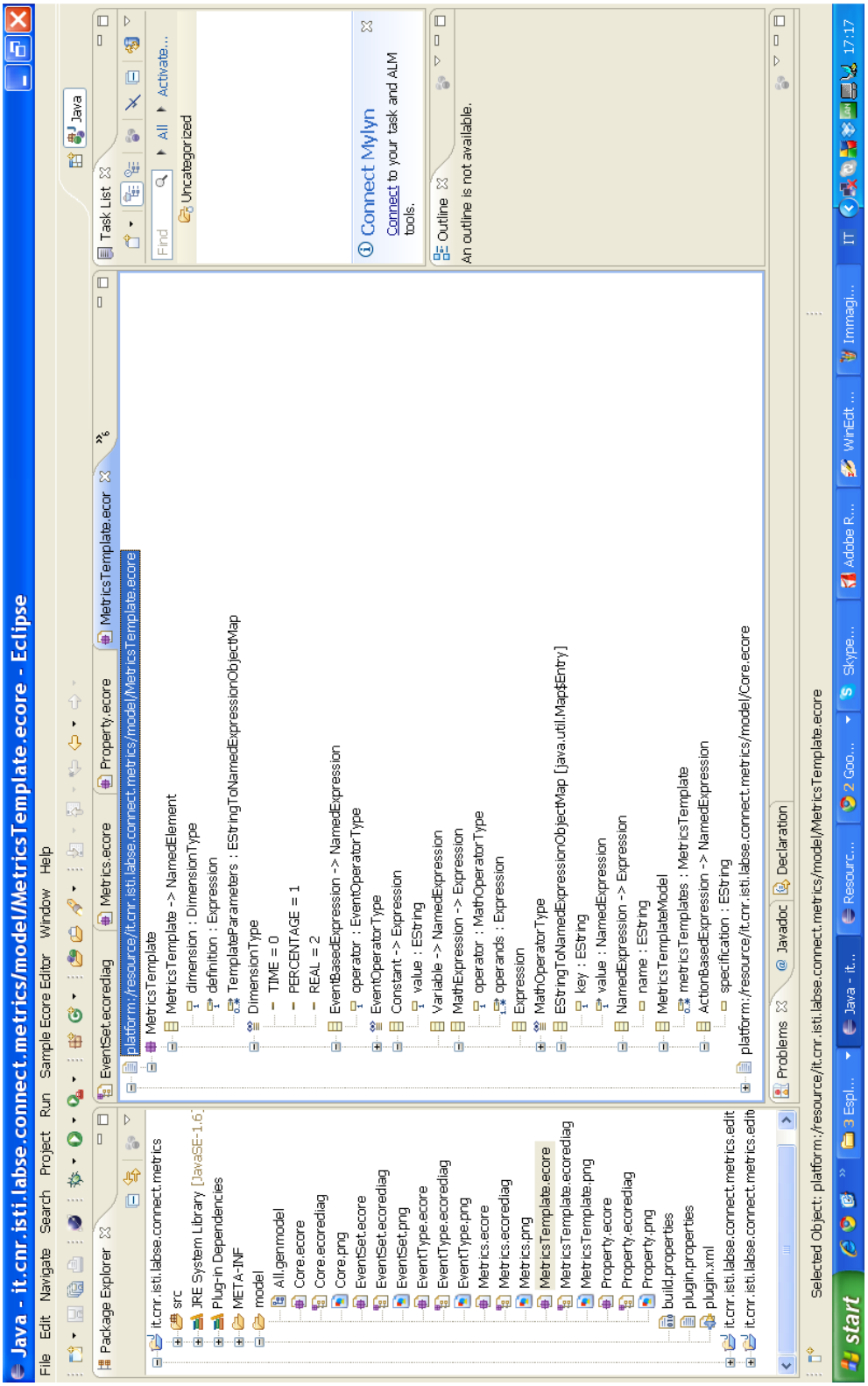


Figure 3.10: MetricsTemplate eCore Definition

Figure 3.11: EventSet and EventType eCore Definition

dependable is a given context), is that of [44]. It proposes the definition of a domain specific meta-model-based language, along the lines of OMG's SMM (Software Metrics Metamodel), and the formalization of a framework implemented through state-of-the-art model-driven tools (KM3, EMF). This proposal outlines, as main advantages of defining such meta-model: the clear view of the domain, the model management, transformation and validation and, the interpretability between different tools by means of a uniform XML-based format. The same advantages are proper to the CPMM meta-model. The notation for dependability cases, presented in [44] and better defined in [64] can be extended by an UML profile and OCL constraints presented in [57], aiming to assess syntactic completeness and consistency of arguments structures in dependability cases. Differently from the above approaches, the CPMM meta-model addresses not only dependability properties but more general descriptive and prescriptive properties that allow the specification, among the others, of performance, security and trust property and of relative metrics.

Another related research area focuses on the definition of generic models and languages, for specifying non functional (QoS) properties. There are specific QoS modeling approaches for QoS-aware service compositions in ubiquitous environments. Some of them use syntactic QoS models (QuAMobile [7] can be an example) whereas other propose semantic ones (COCOA [13] for example is based on the OWL-S semantic language [99]). The OASIS standard Web Service Quality Model (WSQM) [81] represents a standardization effort for the specification of QoS in Web Services. It provides a model for web services quality management, defining common criteria to evaluate quality levels for interoperability and security of services. A more general Quality of Service Modeling Language (QML) is proposed in [50] for describing QoS specifications for software components in distributed object systems. It is an extension of UML, allowing a fine grained specification level of attributes and operations and a dynamic and runtime check of QoS components requirements and dependencies. Similarly, the UML profile for MARTE (Modeling and Analysis of Real-Time and Embedded Systems) [82] provides a common way for modeling hardware and software aspects of a real time embedded system in order to improve interoperability between development tools. It provides facilities to annotate models with information required to perform quantitative predictions and performance analysis. Differently from the approaches presented above, our proposal focuses on a more general and complete framework, with the goal of enable the specification of CONNECT qualitative and quantitative properties into a machine-processable language.

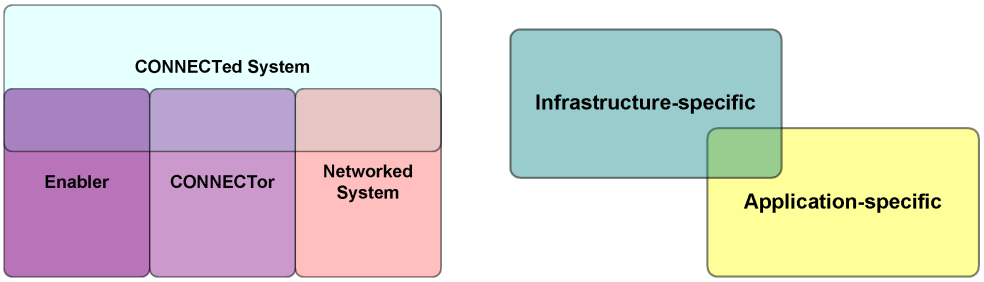
A promising research direction, addressing QoS modeling, focuses on ontologies that allow for the definition of QoS with rich semantic information [71]. In particular, a semantic QoS model is presented addressing the main elements of dynamic service environments (networks, devices, application services and end-users). It makes use of Web Service Quality Model (WSQM) [81] standard to define QoS at the service level, and comprehends four ontologies specifying respectively: the core QoS concepts, the environment and underlying network and hardware infrastructure QoS properties, the application server and user-level QoS properties. As the authors claim in [71], "Our model concentrates on QoS knowledge representation rather than a language to specify QoS. To this extent, our approach has been to decouple QoS knowledge from QoS specification by providing separate and reusable ontologies. Thus any appropriate QoS specification language can be used on top of our QoS model." Differently from this approach, CPMM allows the specification of non functional properties. In the future the integration of the two approaches can be investigated and provided.

Finally,a Quality of Service Modeling Ontology (QoS-MO) is presented in [97] allowing for the semantic specification of QoS constraints of Semantic Web Services. The goal is to extend the existing OWL-S [99] description of the functional characteristic of the Web Services with well defined QoS constraints. As our proposed CPMM meta-model, it allows to define general or domain specific quantitative properties of a service. Moreover, the QoS-MO ontology presents a tight correspondence with the OMG UML Profile [83] that extends UML 2.0 with QoS specification capabilities.

3.4 Relationship between CPMM and Y1 Metrics Framework

In this section we discuss the relationship among the presented meta-model and the CONNECT Metrics Framework presented in [32].

The framework we proposed in the first year revisits the classical dependability and QoS-related metrics to account for evolution and dynamic interactions in open, evolutionary networks. Indeed, it refines classical dependability metrics along a CONNECT-*dependent* dimension and a *context-dependent* dimen-



(a) CONNECT-dependent refinement (b) Context-dependent refinement

Figure 3.12: Dimensions to Derive CONNECT Metrics.

sion:

- **CONNECT-dependent dimension -** This dimension refines generic metrics according to the structural roles of the CONNECT architecture. The rationale behind this dimension is that different definitions of classical dependability metrics may be given for the different actors of the CONNECT architecture, namely the Networked Systems, the Enablers, the CONNECTors and the CONNECTed System. With reference to the above actors, the CONNECT-dependent dimension includes three disjoint classes: *NetworkedSystem-specific*, *Enabler-specific*, CONNECT*or-specific*, plus a fourth partially overlapping class, CONNECT*edSystem-specific* (see Figure 3.12(a)). The *Enabler-specific* and CONNECT*or-specific* classes can be used to obtain "internal" CONNECT metrics, i.e., metrics suitable to assess the dependability level of the CONNECT service.
- **Context-dependent dimension -** This dimension refines and classifies generic metrics according to the application context. The rationale behind the choice of this dimension is that CONNECT metrics can be linked to a particular application scenario and / or to heterogeneous and evolutionary aspects of the different actors of the CONNECT architecture. According to the CONNECT vision, the context-dependent dimension includes two partially overlapping classes (see Figure 3.12(b)): *application-specific*, which refines generic metrics on the basis of the application domain, e.g., safety-critical, delay-tolerant, real-time; *infrastructure-specific*, which refines generic metrics according to heterogeneity and evolution capabilities of the different actors of the CONNECT architecture, e.g., timeouts adopted by communication protocols, number of operational phases.

CPMM complies with the CONNECT Metrics Framework (CMF) since it allows the definition of the metrics according the two dimensions defined above.

For what concern the CONNECT-dependent dimension, this is implemented in the meta-model by referring to several Application Domains. When the EventTypeSpecification element in the meta-model uses terms in the ontology the Learning Enabler constructs for the Networked Systems, the metric (and more in general the property) is NetworkedSystem-specific. If the EventTypeSpecification element refers to CONNECTed Systems application domain, the metric (and more in general the property) is CONNECTedSystem-specific. Again when the EventTypeSpecification element links to terms relative to Enablers (CONNECTors) domain, the metric (and more in general the property) is Enabler-specific (CONNECTors-specific).

For what concerns the context-dependent dimension, this is not explicitly modeled in the CPMM. To make more explicit this definition, it is required that the subject of the metrics (i.e., the concepts in the application-domain model) is semantically classified as application-specific and/or infrastructure-specific referring to specific capabilities of the application or to heterogeneity and evolution capabilities of the different actors of the CONNECT architecture. We plan to refine this aspect in future, if necessary.

3.5 Latency Property for the Terrorist Alert Scenario

In this section we show how to model the following required latency property: *average time needed by the* CONNECT*ed system to reach k% guard devices must be at maximum equal to 10 seconds when in the*

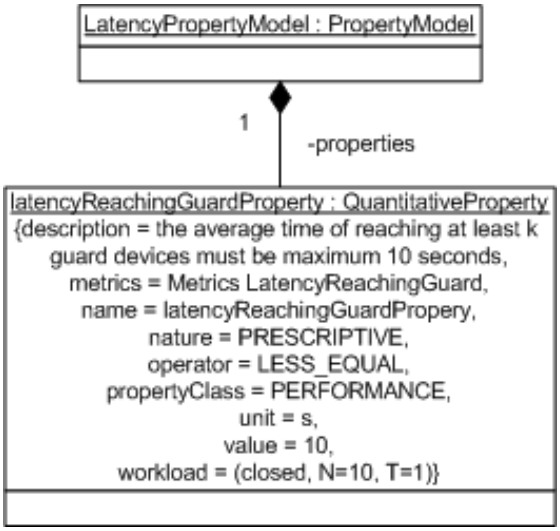


Figure 3.13: Latency Property for the Terrorist Alert Scenario

system there are 10 alerts.

For "time needed by the CONNECTed system to reach a set percentage of guard devices" we mean the average latency experienced in the system from the incoming EmergencyAlert message to the reception of a percentage of eAck coming back from the reached guards' devices. The models for the latency metrics template and its actualization metrics (called LatencyReachingGuard) are showed in Section 3.5.1, whereas the models of the events delimiting the measure (EmergencyAlert and eAck) are described in Section 3.5.2

The model for this PRESCRIPTIVE property is shown in Figure 3.13. The property is a *PERFORMANCE* requirement requiring that the metrics *LatencyReachingGuard* is *LESS_EQUAL* to *10 s* when the system has a workload of *10* alerts.

3.5.1 Latency MetricsTemplate and Metrics

In this section we present the models for the latency_AVG MetricsTemplate (in Figure 3.14) and the LatencyReachingGuard metrics (in Figure 3.15). We recall that the template is generic and the same model can be used in other scenarios. The average latency represents a TIME measure defined as average of the differences of the time stamps of two related generic event instances, in the model x and y, respectively as the latest event occurrence and the former one. Finally, the template exposes two templateParameters, e_1 linked to y, and e_2 linked to x.

The Metrics instead actualizes the template for a specific scenario. Hence it is specific to the application domain it is defined for. This characteristics is modelled by the metrics actual parameters that substitute the template parameters by linking the general description in the template to the specific ontology and hence to the application the metrics refers to. In Figure 3.15, *LatencyReachingGuard* metrics actualizes the *latency_AVG* metrics template by linking the EventSet e_1 to the templateParameter e_1 and the metricsParameter EventSet e_2 to templateParameter e_2. Finally, to fully define the metrics, the two event sets modeling the metricsParameters must satisfy the MetricsConstraint *c1* that establishes that the two event sets must be related to each other, that is refer to the same EmergencyAlert.

Finally, Figure 3.16 reports the event sets needed for the LatencyReachingGuard specification. The e_1 Event Set only refers to emergencyAlert Event Type. The e_2 EventSet instead refers to SeqOfAck EventType by introducing an additional condition. As we will see below, the SeqOfAck EventType is a sequence of eAck observed in the CONNECTed system. To be of interest of the LatencyReachingGuard metrics, the occurrences of SeqOfAck must contains at least k eAck occurrences related to each other, that means they refer to the same emergencyAlert message (in formulas in the e2Defintion.property, different occurrences of eAck in the sequence must have the same emergencyAlert ID, i.e. the same IDe).

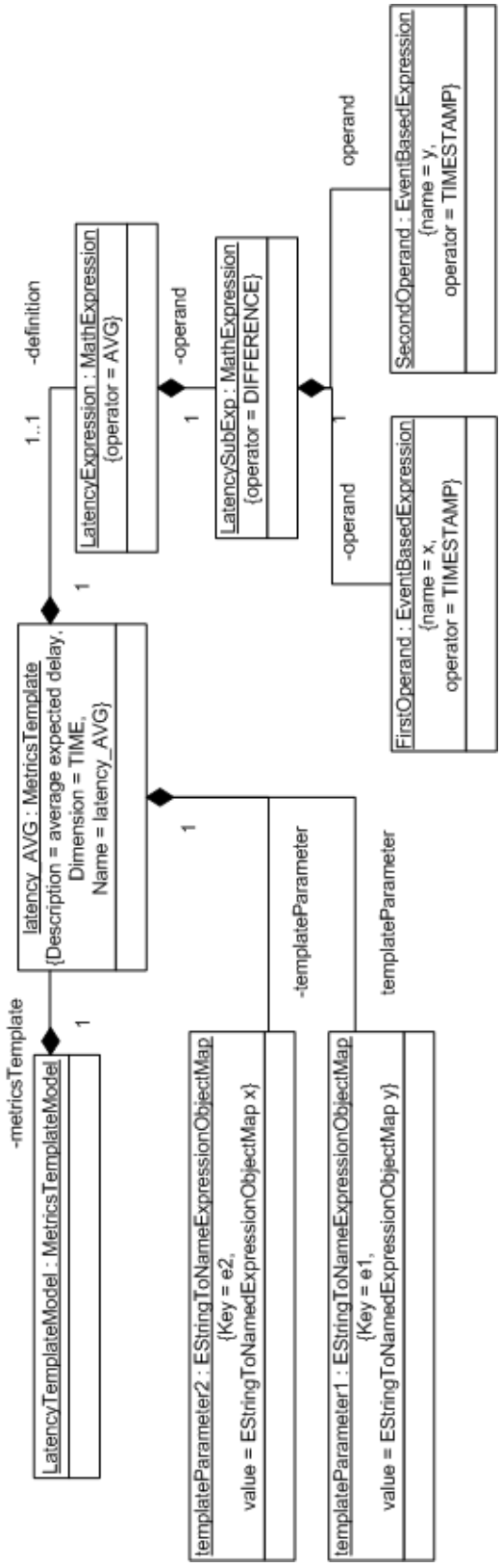


Figure 3.14: Average Latency Metrics Template

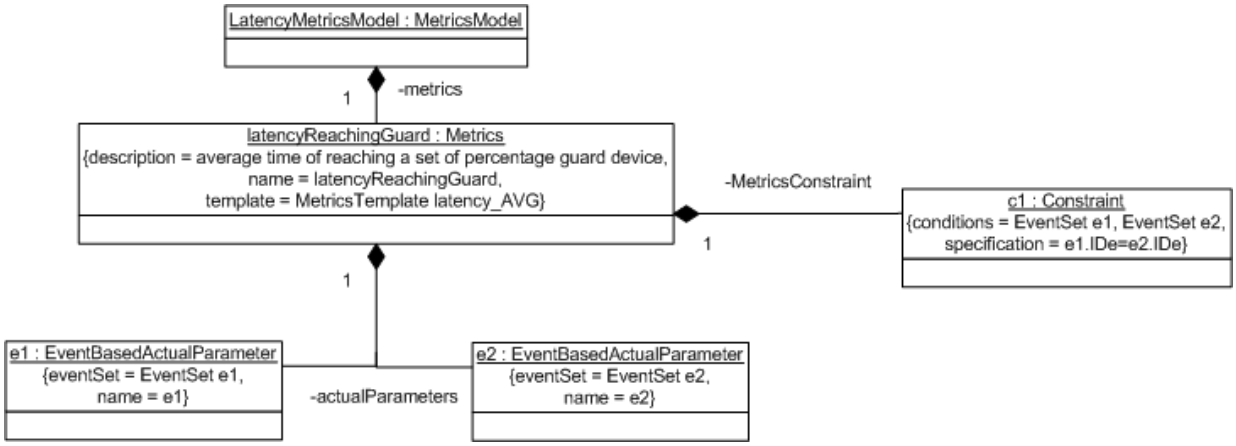


Figure 3.15: Average Latency Metrics for the Terrorist Alert Scenario

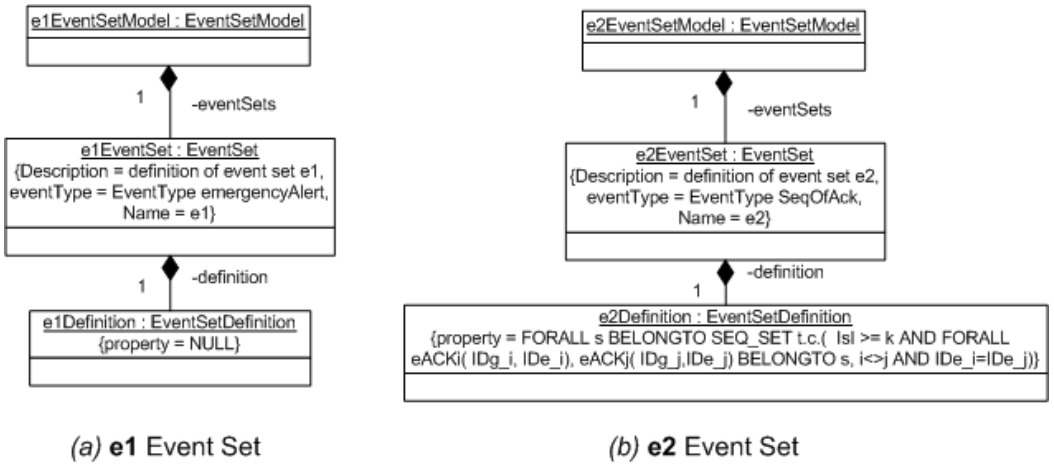


Figure 3.16: Actual Parameters in the Metrics Model

In the EventSet models presented, the EventOccurrences are missing. These model the observed events the EventSet contains, that is the observed events of the specified EventType satisfying the *EventSetDefinition.property*. Since the presented models represent part of the connection request incoming to the CONNECT infrastructure, at that time the mediator is not already synthesized, and the CONNECTed system is not running. The EventOccurrences will be generated by the monitors at runtime when the CONNECTed system is running, once the probes observe the event of the EventType the EventSet refers to.

3.5.2 Application Domain Concepts: EmergencyAlert and eAck Event Types

In this section we describe how the EventType required by the LatencyReachingGuard metrics are defined. In particular, we have two types of event to define: the emergencyAlert and the SeqOfAck event types. The former has a simple event definition since it corresponds to a message directly observable from the CONNECTed system, namely interface operation. The signature of the interface operation is *emergencyAlert(IDe)* as specified in the EventTypeSpecification element. This comes from the ontology created for the scenario. The EventType has a *parameter* that id the formal parameter of the interface operation.

SeqOfAck is a complex EventType since it is defined as a sequence of eAck simple EventType. It has two parameters, that is the emergencyAlert ID (namely IDe) the sequence refers to, and the list of guards messages acknowledging the alert reception (namely, IDgList). At the moment, the meta-model does not

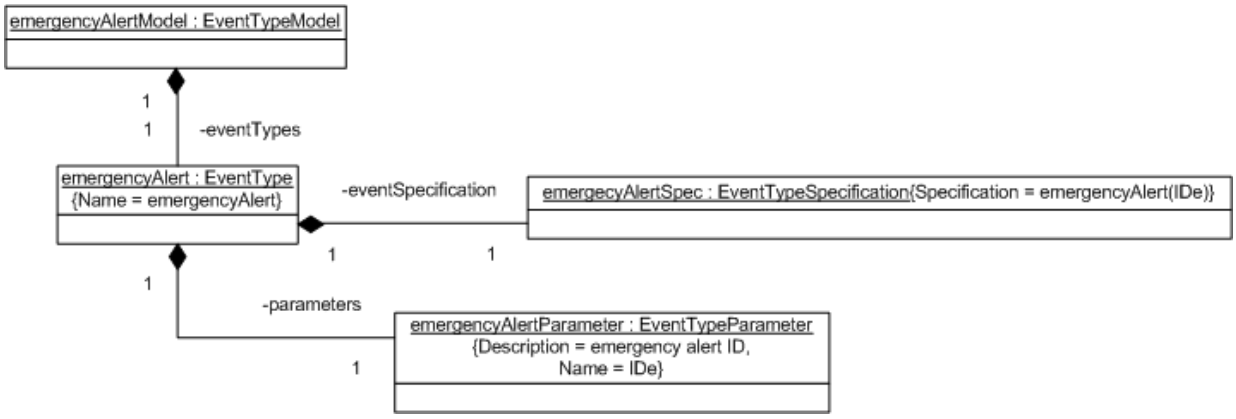


Figure 3.17: Emergency Alert

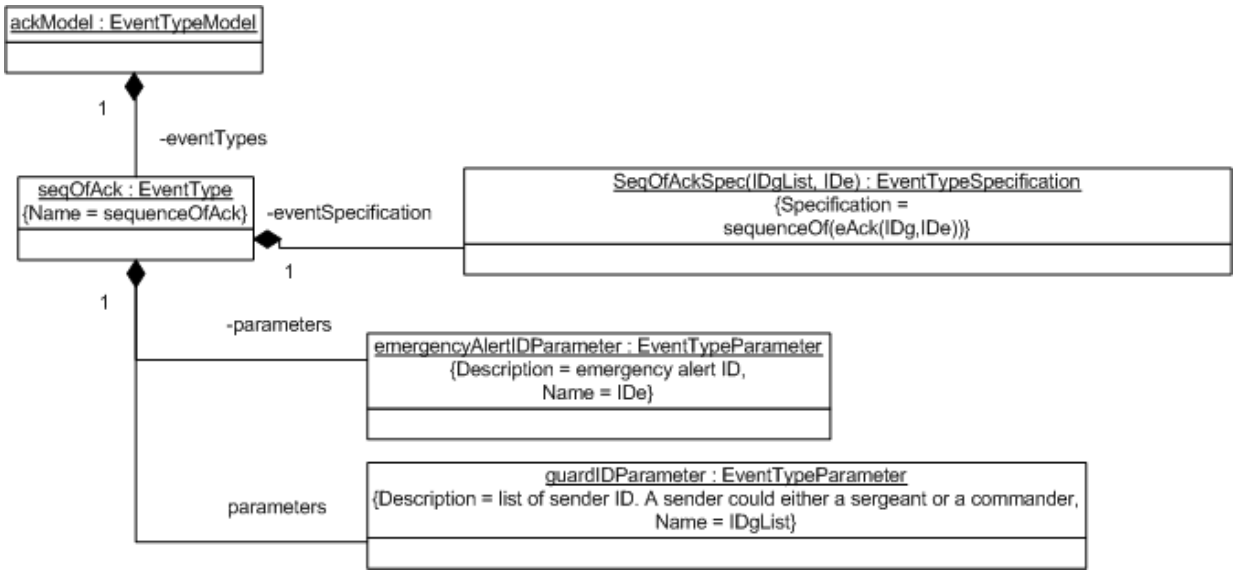


Figure 3.18: Sequence of Ack for an Alert

allow a more formal specification of complex events. We plan to refine it for this purpose in the next year.

3.6 Coverage Property in the Terrorist Alert Scenario

In this section we show how to model the following required coverage property: *the average percentage of guard devices that are reached in 10 seconds by the alert message must be greater than* 70%*.*

This means that, after 10 seconds from the emergencyAlert, at least 70% of guard devices reply with an eAck. The models for the coverage metrics template and its actualization metrics are showed in Section 3.6.1. The models of the events delimiting the measure (eAck) are the same ones of the latency and they have been described in Section 3.5.2. Hence, in Section 3.6.2 we model only the deviceRegistration event.

The model for this PRESCRIPTIVE property is shown in Figure 3.19. The property is a *DEPENDABILITY* requirement requiring that the metrics *CoverageReachingGuard* is *GREATER* than *70%* after an intervalTime of *10 seconds*.

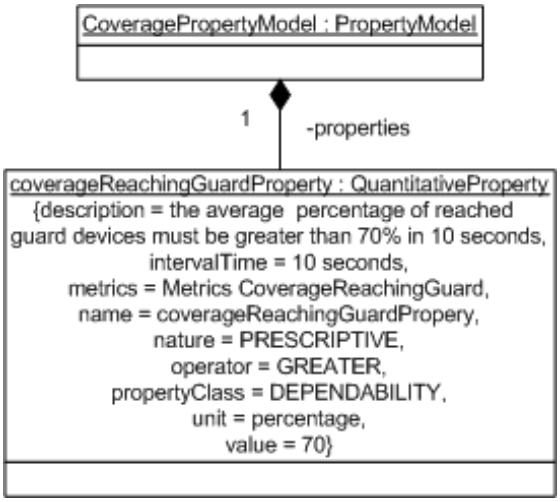


Figure 3.19: Coverage Property for the Terrorist Alert Scenario

3.6.1 Coverage MetricsTemplate and Metrics

In this section we present the models for the coverage_AVG MetricsTemplate (in Figure 3.20) and the CoverageReachingGuard metrics (in Figure 3.21). The average coverage represents a *PERCENTAGE* measure defined as average of the division among the *CARDINALITY* of two set of instances of two types of events, in the model x and y. Finally, the template exposes two templateParameters, e_2 linked to x, and e_3 linked to y.

The CoverageReachingGuard metrics, in Figure 3.21, actualizes the presented template for the Terrorist Alert Scenario by linking the EventSet e_2 to the templateParameter e_2 and the metricsParameter EventSet e_3 to templateParameter e_3. Differently from the latency, the CoverageReachingGuard metrics does not define any constraint.

Finally, Figure 3.22 reports the model for the e_3 event set, while for the definition of the e_2 event set, please refer to Section 3.5.1, since the coverage metrics makes use of the same event set used for the latency. The e_3 EventSet instead refers to deviceRegistration EventType by introducing the following condition: there are no duplication in the occurrences of the deviceRegistration event, that is there are not two different occurrences of the deviceRegistration coming from the same device (i.e., event having the same IDg value).

As for latency, for the same reasons, the EventSet models do not present the EventOccurrence elements.

3.6.2 Application Domain Concepts: EmergencyAlert, eAck and deviceRegistration Event Types

In this section we describe the deviceRegistration EventType presented in Figure 3.23. It has a simple event definition since it corresponds to a message directly observable from the CONNECTed system, namely interface operation. The signature of the interface operation is *deviceRegistration(IDg)* as specified in the EventTypeSpecification element. This comes from the ontology created for the scenario. The EventType has a *parameter* that id the formal parameter of the interface operation.

3.7 Privacy Property in the Terrorist Alert Scenario

The privacy policy required in the Terrorist Alert Scenario is *the photo can only be received by authorized devices*. This means that: *i)* they have a certain level of trust; *ii)* they have to send an ACK; *iii)* they expose a certificate for proving who and where they are.

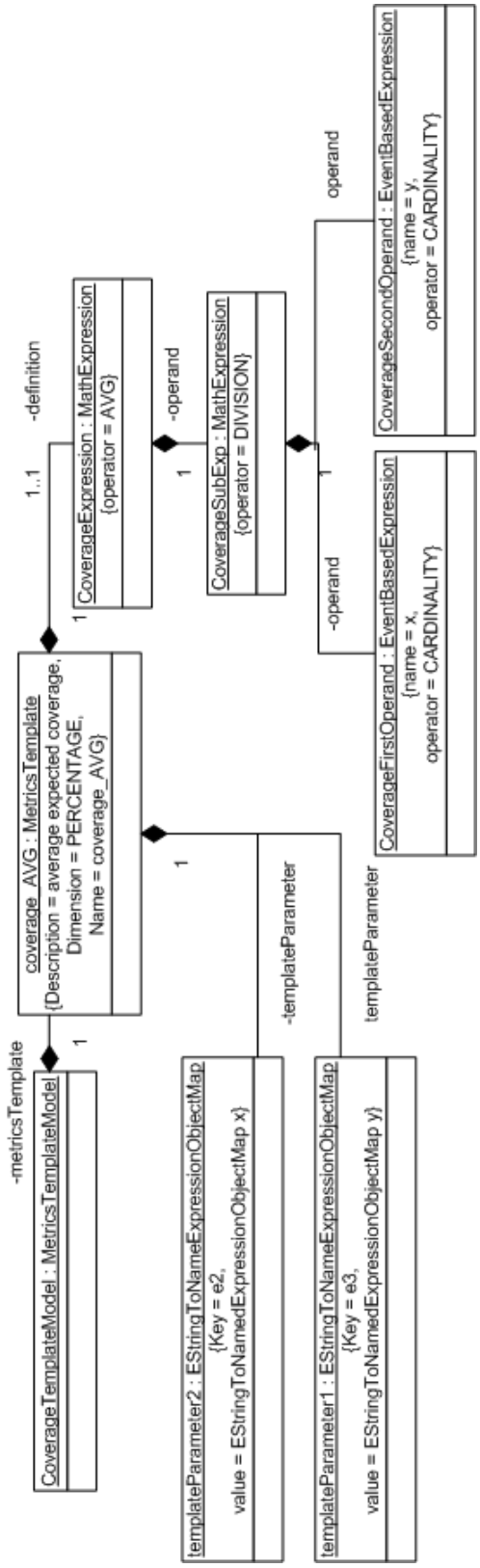


Figure 3.20: Average Coverage Metrics Template

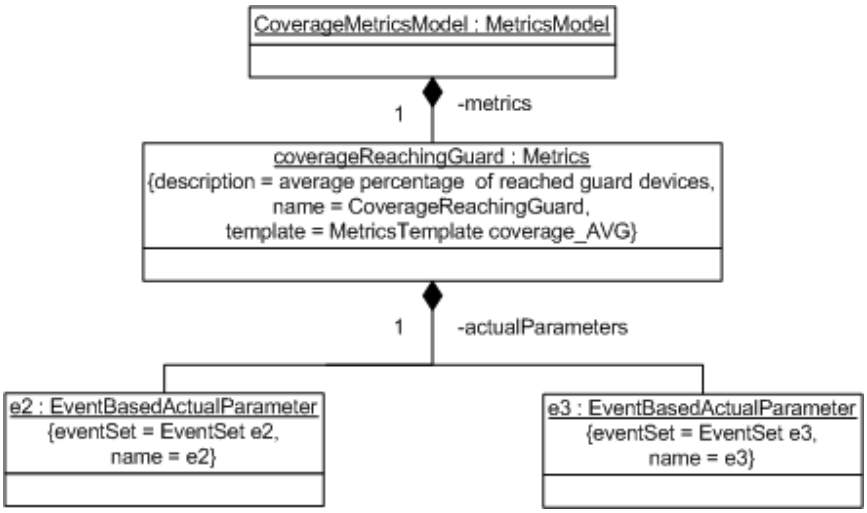


Figure 3.21: Average Coverage Metrics for the Terrorist Alert Scenario

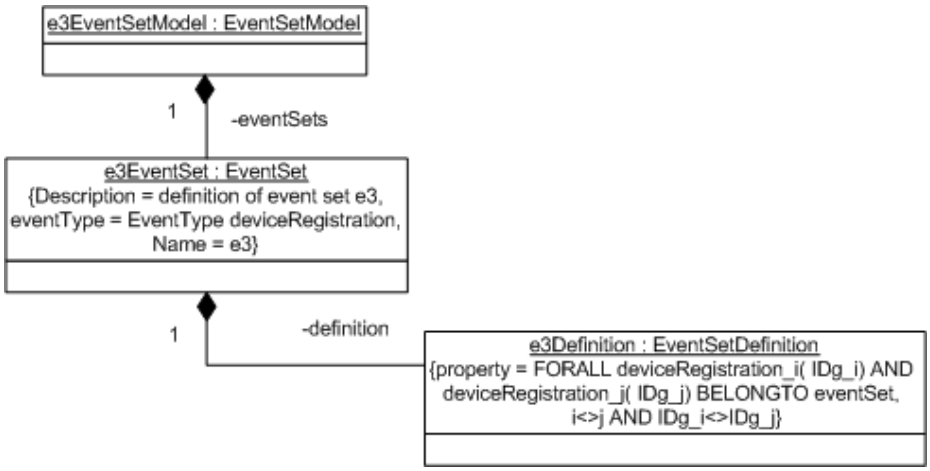


Figure 3.22: e_3 Actual Parameter in the CoverageReachingGuards Metrics Model

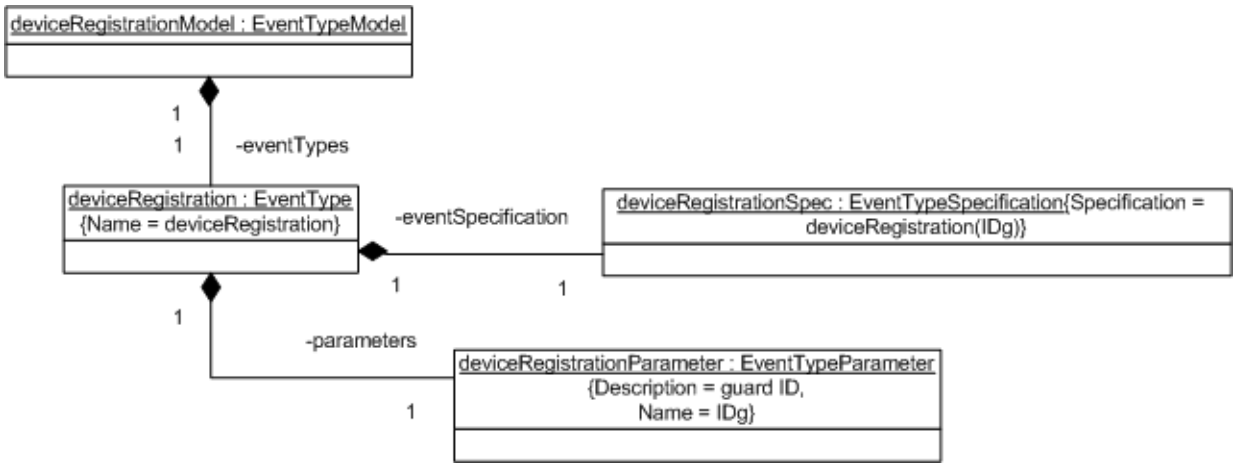


Figure 3.23: Guard Device Registration

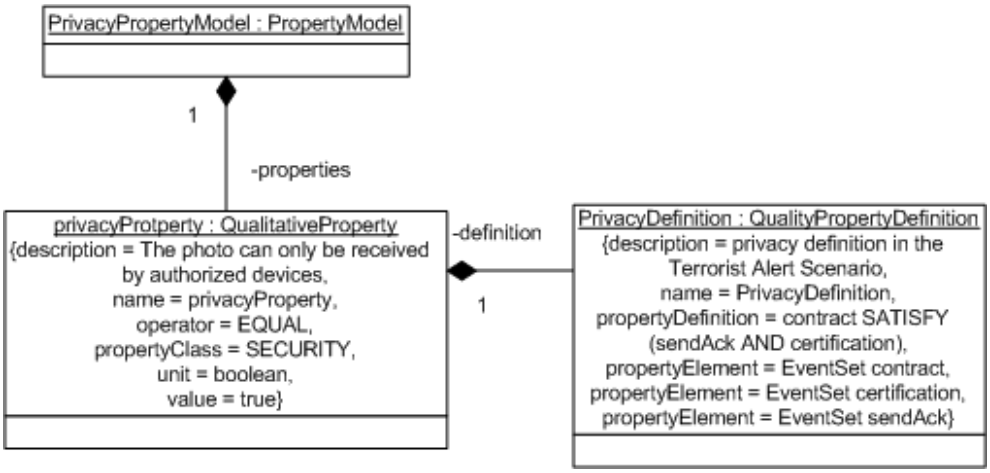


Figure 3.24: Privacy Property for the Terrorist Alert Scenario

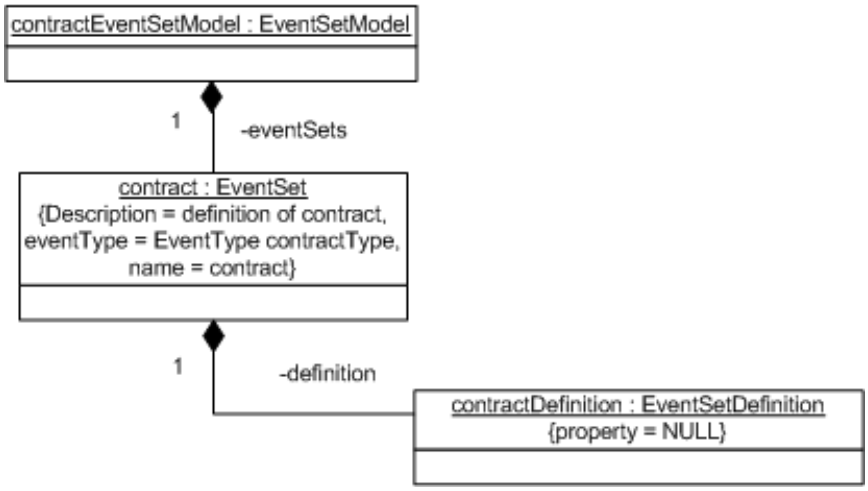


Figure 3.25: The Contract EventSet

The model for this PRESCRIPTIVE property is shown in Figure 3.24. The property is a *SECURITY* requirement that must be *true*. Its definition (*PrivacyDefinition*) sets that the *contract* must satisfy *sendAck AND certification* that are specification of the points *i)* to *iii)* above.

Figure 3.25 shows the definition of the contract EventSet that is very simple since it specifies the corresponding EventType and NULL property. We here do not report sendAck and certification EventSets since they have a specification similar to contract.

3.7.1 Application Domain Concepts: Contract, sendAck and Certification Event Types

In this section we describe the EventTypes necessary to the PrivacyProperty. They are: *i)* the *contractEventType*, in Figure 3.26, describing the contract (that is in the contractSpec element) and the parameters needed to distinguish the sender of messages (IDs and IDc EventTypePArameter); *ii)* the *sendAckEventType*, in Figure 3.27, describing in the *sendAckSpec* element the required notification of the data uploading. This EventType does not have any parameters; *iii)* the *certificationEventType*, in Figure 3.28, describing in the *certificationSpec* element the required certificate the uploading device must have, and the parameter needed to pass the certificate(i.e., the *C* EventTypeParameter);

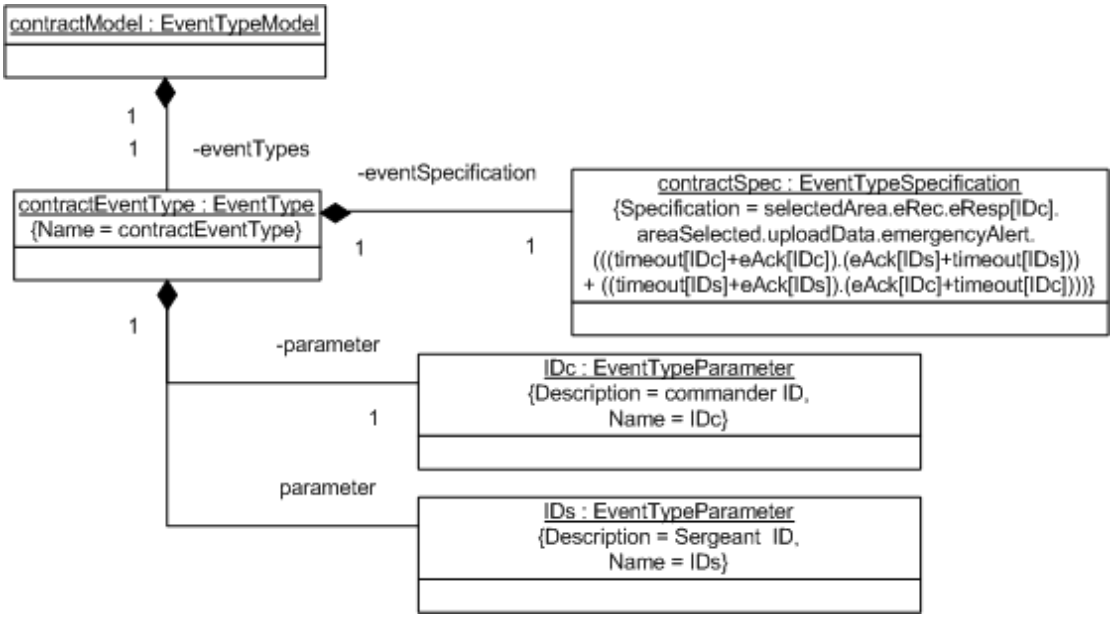


Figure 3.26: Contract Definition

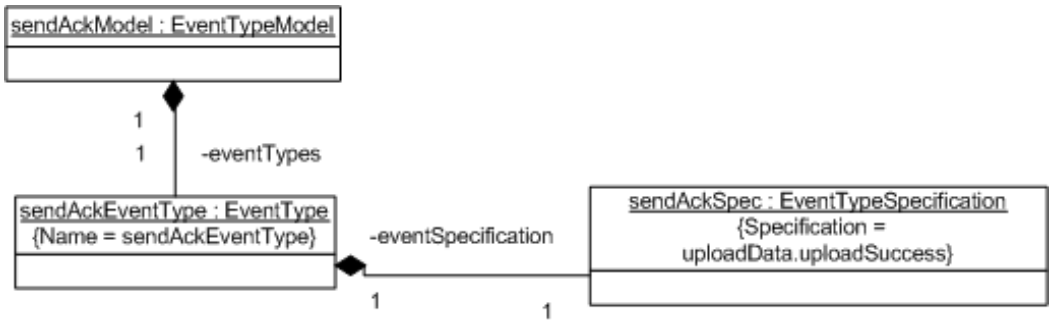


Figure 3.27: sendAck Definition

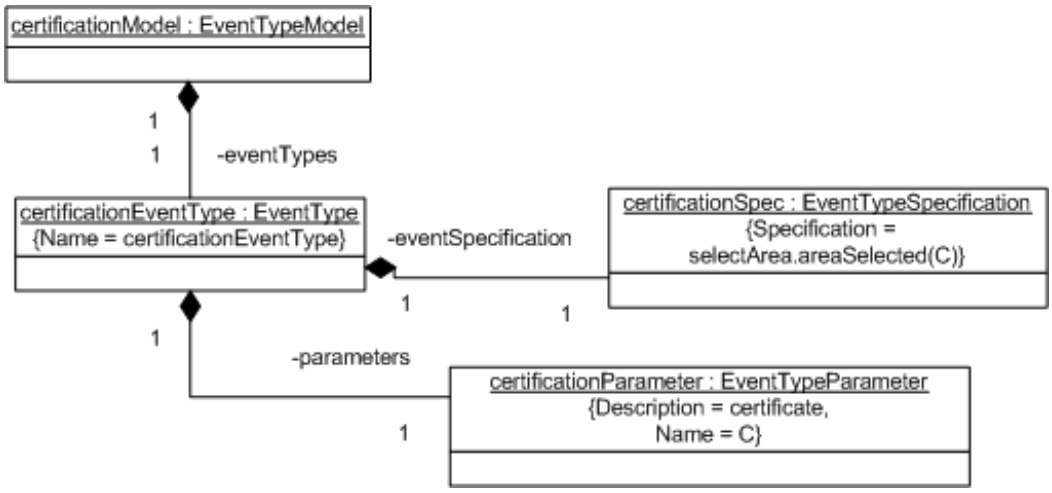


Figure 3.28: Certification Definition

3.8 Conclusions and Further Research Directions

In this chapter we presented the CONNECT Property Meta-Model we devised to express properties in a machine-processable way. The properties that can be specified are of three natures: abstract, prescriptive and descriptive. An *abstract* property indicates a generic property definition, a *descriptive* property represents a guaranteed/owned property of the system, and a *prescriptive* one indicates a system requirement, then a required property in the system. Moreover, the CPMM allows the definition of *qualitative* and *quantitative* properties. Differently from the qualitative one, the quantitative property, to be fully described, must indicate the metrics definition needed to quantify it.

We implemented CPMM by the Eclipse Modeling Framework and we generated the corresponding editor that can be used to specify property models. These are released in the Appendix-Prototypes associated with this document.

There are various directions for future work. First, since the models that conform to such a meta-model are machine-processable, we plan to define automated procedures (in form of ModelToModel or ModelToText transformations) that, from such models, instrument the monitoring Enabler to monitor properties of interest and produce suitable input for the analysis tools.

Moreover, some meta-model parts need to be refined. In particular:

- the QualitativePropertyDefinition meta-class should be refined by defining a suitable language (meta-model) that allows the specification of complex property;
- the Constraint meta-class (in the Metrics meta-model) requires a specification of a language to properly describe conditions the metric requires;
- the EventTypeSpecification meta-class should be refined by an event-based language that enables the specification of complex EventType.
- the whole definition of the ActionBaseExpression introduced to model trust at the moment is not completely defined. It will be refined as soon as the trust model will be defined and validated.

At the moment, the meta-model only allows the specification of the transition (or action)-based properties. We did not consider state-based properties, since CONNECT assumes that it can use only the information the Learning Enabler is able to observe during the Networked System execution. However, if the Networked Systems expose some additional information (specification of internal behaviors, interaction protocols), it can happen that they can provide some information on their internal state, and hence state-based properties could become of interest. To address also this last case, we plan to extend the meta-model with concepts that allow the specification of state-based properties.

As other possible future work it could be the integration of the CPMM with ontologies for non-functional requirements to introduce semantics to CPMM. Examples of suitable and interesting ontology we can refer have been discussed in Section 3.3.

We have then shown how the CONNECT Property Meta-Model can be used to instantiate a set of relevant properties for a given application scenario. We have referred to the Terrorist Alert scenario. In addition to more extensive usage of the meta-model (itself undergoing further refinements), our future research work will include the specification of Model-to-Model and Model-to-Code transformations for automatically translating the instantiated property models towards the specific notations adopted by the Enablers, e.g. the Dependability&Performance analysis or the Monitoring Enablers.

4 Dependability&Performance Analysis Enabler

In CONNECT we are interested in quantitative, probabilistic dependability and performance evaluation. To this purpose, a model-based approach is adopted, which in general terms is composed of two phases: (i) building a model from the elementary stochastic processes that represent the behaviour of the components of the system and their interactions (mainly, these elementary stochastic processes relate to failures, to repair and to service restoration); (ii) processing the model to obtain the expressions and the values of the dependability measures of the system.

Research in dependability and performance analysis has developed a variety of model-based techniques, each focusing on particular levels of abstraction and/or system characteristics. As already motivated in [32], in CONNECT we consider as evaluation techniques:

- *Stochastic model checking*, which is a formal verification technique for the analysis of stochastic systems. It is based on the construction of a probabilistic model from a precise, high-level description of a system's behaviour.
- *State-based stochastic methods*, which use state-space mathematical models expressed with probabilistic assumptions about time durations and transition behaviours. They allow explicit modeling of complex relationships (e.g., concerning failure and repair processes), and their transition structure encodes important sequencing information.

During the first year of the project, dependability and performance analysis has been applied to assess to some extent dependability and performance metrics of Networked Systems. Specifically, the activity focused on gossip protocols in presence of heterogeneity at level of node communication and computation, which is typically neglected in already existing analyses. In a gossip, each node periodically exchanges messages with randomly selected nodes in order to disseminate data, exchange membership information, or build distributed state (http://gossiplib.gforge.inria.fr/). Such kind of protocols are expected to be of interest in CONNECT as they may be considered as basic building blocks for many distributed services. In fact, gossip based protocols have been applied in a wide range of situations such as data dissemination, overlay maintenance, and, more recently, also in peer-to-peer streaming applications. We experimented both stochastic model checking and state-based stochastic methods and the two studies have shown the complementarity of the two approaches: while the former is very accurate in determining best and worst case behaviour but for small numbers of involved nodes, the latter is able to provide average values for large-scale networks.

During this second year, focus has been shifted on the dependability and performance analysis of the CONNECTed system, and specifically to assess, before deployment, if the dependability and performance requirements requested by the NSs can be satisfied by the CONNECTor being synthesised. To this purpose, we have defined the architecture of a Dependability&Performance Analysis Enabler, called DePer, that supports both stochastic model checking and state-based stochastic methods evaluation techniques. Preliminary prototypes for the two approaches, based on PRISM and Möbius assessment tools respectively, have been also developed (a first release for the latter is included in the Appendix-Prototypes). The complementarity of the two approaches has been also shown through the analysis of the CONNECT "Terrorist Alert" scenario.

The rest of this chapter describes in details progress achieved on dependability and performance analysis during this second year. Precisely, the architectural structure of the DePer is described in Section 4.1. Then, Sections 4.2 and 4.3 detail the functioning of the Möbius-based and Prism-based implementations, respectively. Finally, in Section 4.4, we demonstrate both analysis methods on the Terrorist Alert scenario. Conclusions are drawn in Section 4.5.

4.1 DePer Architecture

The *DePer* provides support to the definition of a CONNECTor that allows NSs to interact with a given level of dependability and performance properties.

Before presenting the architecture of this Enabler, we briefly discuss its relations with other Enablers of the CONNECT architecture. In Deliverable D1.2 [33], the complete overview of Enablers is provided.

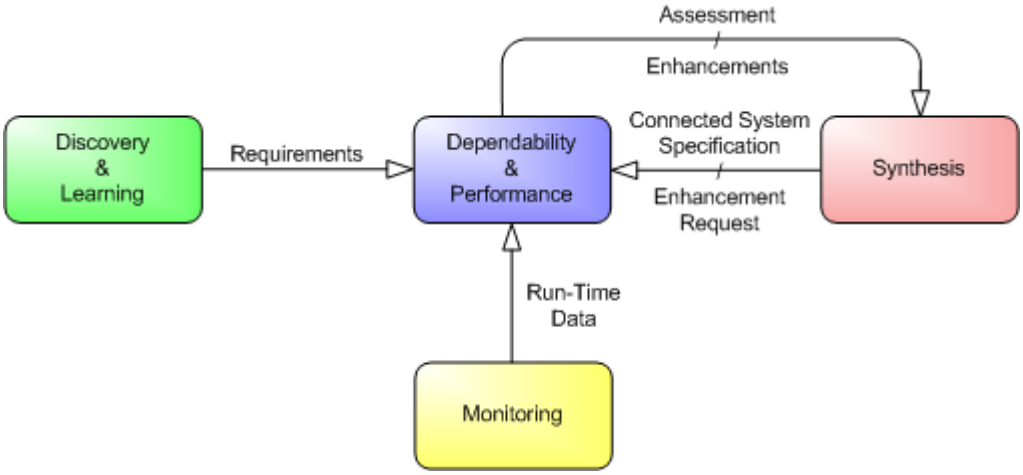


Figure 4.1: Input-Output Relations between DePer and the Other Enablers

Adopting a DePer-centric view, here we restrict to those having input-output relations with the DePer, as shown in Figure 4.1.

The role of these Enablers is synthetically recalled in the following.

Discovery. This Enabler gathers information on the NSs. Specifically, the Enabler discovers mutually interested NSs, and retrieves information on the specification of their interfaces. The unit assumes that NSs are *discovery enabled*, i.e., they provide a minimal description of their intent and functionalities.

Learning. This Enabler completes the specification through a learning procedure (e.g., via model-based testing) when NSs do not provide a sufficient description of their behaviour.

Synthesis. This Enabler performs the dynamic synthesis of mediating CONNECTors to enable interoperation among NSs willing to interact. The Enabler uses the behavioural models built by Discovery and Learning to identify mismatches between the communication protocols employed by the NSs, generates a CONNECTor that resolves the incompatibilities between the communication protocols, and deploys the CONNECTor.

Monitoring. This Enabler becomes operational when the CONNECTor is deployed. The Enabler continuously monitors the deployed CONNECTor to update the functional and non-functional specification of the CONNECTor with run-time data.

According to the CONNECT vision, a Networked System broadcasts a *connect request* whenever a new connection to a service is needed. The connect request contains a description of the requested service together with a specification of the required dependability level for the service. When Discovery detects a connect request, it looks for available Networked Systems that can provide the requested service. If such systems are found and operate a communication protocol different from that of the Networked System that made the connect request, Discovery triggers the process of creating a suitable CONNECTor that enables interoperation. The Synthesis Enabler, on the basis of the specification of the communication protocols, produces a mediating CONNECTor. Before CONNECTor deployment, Synthesis activates the DePer to evaluate if the CONNECTed system that will be obtained satisfies the non-functional requirements expressed by the Networked Systems. If the non-functional requirements are satisfied, the CONNECTor is deployed; otherwise, Synthesis is supported by the DePer in the definition of possible enhancements that can be applied. Once the CONNECTor is deployed, the Monitoring Enabler continuously updates the functional and non-functional specification of the CONNECTor with run-time data to allow the other Enablers to take into account dynamic system changes.

Therefore, as shown in Figure 4.1, the joint activity of Discovery and Learning provides the dependabil-

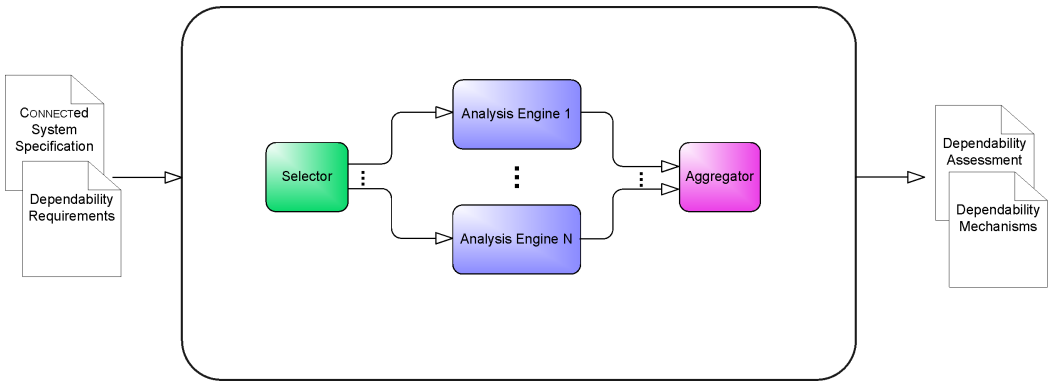


Figure 4.2: Architecture of the Dependability&Performance Analysis Enabler

ity requirements; Synthesis provides the specification of the CONNECTed system, and possibly requests a dependability enhancement; Monitoring provides run-time data on the execution of the deployed CONNECTor. The dependability and performance assessment and the enhancements produced by the DePer are used by Synthesis.

4.1.1 Architecture

The architecture of the DePer is shown in Figure 4.2 and also described in [75]. Currently, this Enabler accommodates dependability and performance analysis performed through both the stochastic state-based and the stochastic model-checking approaches. Actually, the architecture is general and other analysis methods could be easily included by specifying and implementing an appropriate Analysis Engine module. The Selector and Aggregator modules, at the entrance and exit of the architecture, allow the selection of the analysis method and the aggregation of the analyses results (in case more than one method is applied), respectively. More details on each module are provided in the following.

4.1.2 Selector

The Selector module activates, depending on the characteristics of the specification of the CONNECTed system and of the requirements, the most suitable analysis engine among those available to the Enabler. In fact, the employed engines implement different approaches to analyse dependability and performance properties of a CONNECTed system. Each approach has its own advantages regarding modeling capability, specification of properties, and scalability; hence, besides using the different engines to cross validate the results and to improve the confidence in the correctness of the models,they actually complement each other. In the study conducted during the first year [32], the characteristics of the stochastic model checking and state-based stochastic methods evaluation approaches have been already pointed out; other considerations will be provided when addressing the two implementations of the DePer in Sections 4.2 and 4.3.

4.1.3 Dependability&Performance Analysis Engine

The Dependability&Performance Analysis Engine is logically split into four main functional modules (see Figure 4.3): Builder, Analyser, Evaluator and Enhancer. The Builder module derives the dependability or performance model of the CONNECTed system from the specification provided by Synthesis. The Analyser uses the generated dependability and performance model to perform a quantitative assessment of the non-functional requirements reported by Discovery/Learning. The Evaluator checks the analysis results to determine if the non-functional requirements are met. If the requirements are satisfied, the Evaluator reports to Synthesis that the CONNECTor can be successfully deployed. If, on the other hand, the requirements are not satisfied, the Evaluator reports a warning message to Synthesis. The Synthesis unit, in turn, may reply with a request to improve the dependability and performance level of the CONNECTed system.

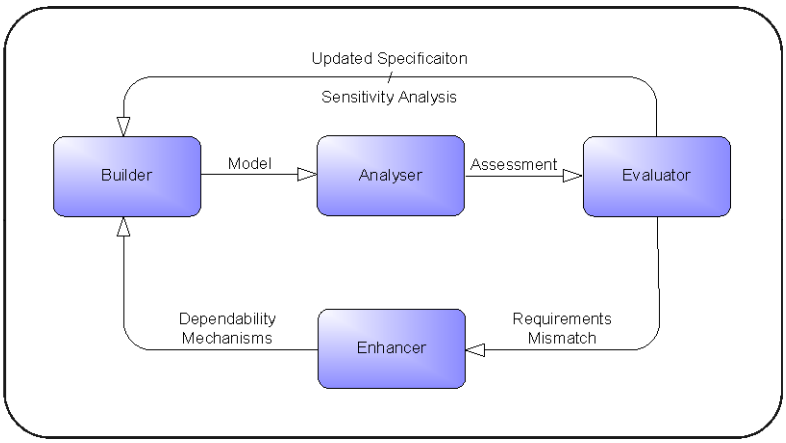


Figure 4.3: Architecture of the Dependability&Performance Analysis Engine

Upon receiving such enhancement request, the Evaluator module activates a loop in the Dependability&Performance Analysis Engine to determine solutions to improve the dependability and performance level of the CONNECTed system, e.g., use an updated specification that takes into account an alternative CONNECTor deployment or enhance the CONNECTor specification with ad hoc mechanisms. In the following, we provide more details on the functionalities of the modules of the Dependability&Performance Analysis Engine.

Builder

The Builder module takes in input the specification of the CONNECTed system from Synthesis, and the dependability and performance requirements from Discovery/Learning. The specification includes both the nominal behaviour of the CONNECTed system, i.e., in fault-free situations, and the exceptional conditions, i.e., the failure modes. The module produces in output a dependability and performance model of the CONNECTed system that contains enough details to assess the given dependability and performance requirements.

Specification of the CONNECT*ed system.* With reference to recent works on synthesis of mediating CONNECTors [94] and automata discovery/learning [89], the specification of the CONNECTed system is given with Labelled Transition Systems (LTSs) annotated with non-functional information necessary to build the dependability and performance model of the CONNECTed system. An LTS is an abstract machine that represents the sequence of actions performed by the system. Formally, an LTS is a tuple $(\mathcal{S}, \mathcal{S}_0, \mathcal{L}, \mathcal{T})$, where $\mathcal{S}$ is a set of states, $\mathcal{S}_0 \subseteq \mathcal{S}$ is a set of initial states, $\mathcal{L}$ is a set of labels, and $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{L} \times \mathcal{S}$ is a transition relation. Annotations include, for each labelled transition, the following fields: *time to complete*, *firing probability*, *failure mode*, and *failure probability*.

Dependability and performance requirements. In our architecture, the dependability and performance requirements provided by the Networked Systems are translated by Discovery/Learning into *metrics* and *guarantees*. Metrics are arithmetic expressions that describe how to obtain a quantitative assessment of the properties of interest of the CONNECTed system. In this context, metrics are expressed in terms of transitions and states of the LTS specification of the Networked Systems. Guarantees are boolean expressions that are required to be satisfied on the metrics under a set of constraints.

Dependability and performance model. The dependability and performance model of the CONNECTed system is specified with a formalism that allows to describe complex systems that have probabilistic behaviour, e.g., stochastic processes.

Analyser

The Analyser module takes in input the dependability and performance model from the Builder module and the dependability and performance requirements from Discovery/Learning. The module builds a reward model, i.e., a model that enables a quantitative assessment of the metrics of interest, and makes use of a solver engine to obtain a quantitative assessment of the dependability and performance metrics.

Reward model. The reward model is the dependability and performance model extended with reward functions. Reward functions allow to specify properties of interest: they return a value depending on the system state, and can be evaluated either at an instant of time or accumulated over a time frame.

Solver. The solver evaluates the reward functions defined in the reward model. The evaluation can be performed either through analytical approaches or through simulation. Analytical approaches are based on state space analysis, and they are applicable when the model does not face state space explosion and when timing information follow deterministic or exponential distributions. Simulation, on the other hand, is always applicable (unless extremely rare events have to be evaluated), and is expected to be particularly useful when a tradeoff between accuracy and time to perform the assessment is needed.

Evaluator

The Evaluator reports to Synthesis if the CONNECTed system satisfies the dependability and performance requirements provided by Discovery/Learning. In the case of requirements mismatch, the Evaluator sends a warning message to Synthesis, and may receive back a request to evaluate if enhancements can be applied to improve the dependability (and/or performance, depending on the received request) level of the CONNECTed system.

Requirements mismatch. If the requirements are not satisfied, the Evaluator may receive a request to explore one of the following three directions for improvements:

1. Update the specification of the CONNECTor to take into account an alternative CONNECTor deployment (e.g., a deployment that uses a communication channel with lower failure rate). Upon receiving this request, the Evaluator triggers a new analysis that considers the updated specification of the CONNECTor.

2. Enhance the specification of the CONNECTor by including dependability mechanisms, which are counter-measures that can be adopted to contrast failure modes affecting performance and/or dependability metrics (e.g., a message retransmission technique). Upon receiving this request, the Evaluator triggers a sensitivity analysis whose objective is to understand which failure modes of the CONNECTed system have highest impact on the dependability or performance metrics. Such failure modes will be the first ones to be considered for devising appropriate counter-measures capable of limiting their effects. To this end, the Evaluator builds a sensitivity analysis campaign to instruct the Builder module on the creation of dependability and performance model variants, each of which considers a specific subset of failure modes, among those foreseen. Whenever a variant is generated, the Analyser module performs the assessment of the metrics on the generated model. The Evaluator collects the analysis results and, after all variants have been analysed, produces a ranking of the failure modes. This ranking is used by the Evaluator to iteratively activate the Enhancer module as long as one of the following conditions is not met: the guarantees are satisfied, or the Enhancer signals that all possible dependability mechanisms have been explored.

3. Apply a combination of the previously mentioned enhancements.

Enhancer

The Enhancer is activated by the Evaluator when the guarantees are not satisfied and Synthesis makes a request to enhance the CONNECTor with dependability mechanisms. The Enhancer is instructed by the Evaluator module on the requirements mismatch and the failure mode that needs to be tackled. Specifically, the Enhancer performs the following actions: (i) selects a dependability mechanism that can be

employed, among those available, to contrast the failure mode indicated by the Evaluator module; (ii) if all possible applications of dependability mechanisms have already been explored without success, the Enhancer sends a warning message to the Evaluator module; otherwise, the Enhancer instructs the Builder module on the application of the selected dependability mechanism in the CONNECTed system model and triggers a new analysis.

Dependability mechanisms. Typically, dependability mechanisms are based on the application of redundancy, e.g., duplication of system channels, or retry of message transmissions over system channels. The dependability mechanism, in this context, will be embedded in the synthesised CONNECTor, because Networked Systems are not under the control of the framework. Nevertheless, the dependability mechanisms embedded in the CONNECTor can be employed to improve, to some extent, the dependability and performance level of the Networked Systems. For example, the reliability level of a transmission performed by a Networked System can be improved through timeouts or message retransmissions applied at the CONNECTor level.

4.1.4 Aggregator

The Aggregator module is in charge of selecting the analysis results to be provided in output to the Synthesis Enabler, in case more than one Analysis Engines have been activated for a CONNECTed system specification. Therefore, when only one kind of analysis is performed, based on the choice made by the Selector module, the Aggregator just conveys the analysis results to the output interface of the DePer. Instead, when more analysis methods are activated, their results are collected by the Aggregator and elaborated according to some criteria to derive the output results. The definition of this module is still in progress.
The first step to be performed is a comparison among the values provided by the different methods to check whether they are in agreement (within a certain tolerance degree, to cope with natural dissimilarities inherent to the use of different methods). In the case of a matching comparison, the reliance in each of the employed approaches is increased (cross-validation) and all the analysis results can be equally considered valid, so anyone of them can be output as final analysis values. Alternatively, some form of mediation could be made on the obtained multiple results (e.g., the average), to balance the effects of single method's approximations. A mismatch, instead, would be the symptom of erroneous/too inaccurate analysis by at least one of the applied methods. Let us recall that the DePer in CONNECT is based on an automated procedure, starting from given specifications of the CONNECTed system and of the dependability and performance requirements, partially implemented at the current stage. Therefore, once fully automated, we expect that the case of mismatch would be removed by construction; however more investigations are necessary on this issue.

4.1.5 Some Remarks

The architectural structure and implementation described in this chapter constitute an important step towards the definition of an automated procedure to provide dependability and performance analysis as a support to the synthesis of dependable CONNECTors. There are several aspects that still need to be investigated to fully reach the ambitious goal of automated dependability and performance analysis, and a discussion on the main points that need to be addressed is proposed in the following.

Possible constraints on the time allowed for the dependability and/or performance analysis to complete have not been accounted for so far. At the moment, the envisaged applications of the framework assume that Networked Systems do not suspend their activity while waiting the CONNECTor to be ready. Indeed, at the moment, timing constraints are considered only on the operations performed after CONNECTor deployment. Performing dependability and performance analysis in a complete on-line setting is a very challenging problem and techniques are needed to balance between time to produce results and their accuracy. For instance, in the automatic generation of the dependability and performance model from the specification of the CONNECTed system, a technique needs to be developed to optimise the dependability and performance model on the basis of the specific metrics that needs to be assessment.

Compositional solution methods for the dependability and performance model would be desirable, possibly reusing partly solved model, e.g., when the synthesised CONNECTor is derived as specialisation of

an already existing CONNECTor that has already been analysed, or when already analysed dependability mechanisms are introduced in the dependability and performance model. Indeed, although the addressed context is dynamic and evolving, we assume that the pace at which evolution occurs is in general considerably slower that the requested rate of an already available CONNECTor. Hence, the Networked Systems are expected to remain stable for a significant portion of their lifetime and to request services for which the same synthesised CONNECTor can be reused to satisfy interoperability.

The ontology to support enhancement of the CONNECTor model with dependability mechanisms coping with failure patterns needs to be extended over time with new mechanisms, to enhance the handling of failure modes or to contrast new failure modes. We should also take into account that the identification of the dependability mechanisms may depend on the CONNECTor deployment, i.e., the CONNECTor may share resources with some of the bridged Networked System, or the CONNECTor may have its own resources separate from those of the Networked Systems.

4.2 Möbius-based Prototype Implementation

Möbius [27] is a popular software tool that provides a comprehensive framework for model-based dependability and performance evaluation of systems. The main features of the tool include: (i) multiple high-level modeling formalisms, including, among others, Stochastic Activity Networks (SANs) [93] and PEPA fault trees [53]; (ii) a hierarchical modeling paradigm, allowing one to build complex models by first specifying the behaviour of individual components and then by combining the components to create a model of the complete system; (iii) customised measures of system properties; (iv) distributed discrete-event simulation, to evaluate measures using efficient simulation algorithms to repeatedly execute the system and gather statistical results of the measures; (v) numerical solution techniques, to obtain exact solutions for Markov models.

Modeling formalism. We model the system through Stochastic Activity Networks (SANs). SANs are stochastic extensions of Petri Nets introduced in [78] and formally defined in [93]. They have a graphical representation and consist of four primitive objects: *places*, *activities*, *input gates* and *output gates*. Places in SANs have the same interpretation as in Petri Nets, i.e., they hold tokens. The number of tokens in a place is referred to as the marking of that place, and the marking of the SAN is the set of all place markings. There are two types of activities: instantaneous and timed. Timed activities represent actions that have a duration that impacts the performance of the modelled system, e.g., message transmission time, recovery time, time to fail. The duration of each timed activity is expressed via a time distribution function. Both instantaneous and timed activities may have *case probabilities*. Each case probability stands for a possible outcome of the activity, and can be used to model probabilistic aspects of the system, e.g., probability for a component to fail. Input gates control the enabling of activities, and output gates define the state change that will occur when an activity completes.

SAN models can be composed with *Join* and *Rep* operators. Join is used to compose two or more SANs. Rep is a special case of Join, and is used to construct a model consisting of a number of replicas of a SAN. Models in a composed system interact via *Place Sharing*. Place Sharing is a composition formalism based on the notion of sharing places via an equivalence relation.

Properties of interest. Properties of interest are specified with *reward functions*. Each reward function is a C++ function that specifies how to measure a property on the basis of the marking of the SAN. There are two kinds of reward functions: *rate reward* and *impulse reward*. Rate rewards can be evaluated at any time instant. Impulse rewards are associated with specific activities and they can be evaluated only when the associated activity completes. Measurements can be conducted at specific time instants, over periods of time, or when the system reaches the steady state.

4.2.1 Builder Module

The prototype implementation of the Builder module takes in input the LTS of the CONNECTed system described with Finite State Processes (FSP) [72]. LTS annotations are expressed as C++ functions. The dependability and performance model of the system, specified through SANs, is obtained from the LTS model by using the theory of regions [47]. A region identifies a set of states in the LTS such that all transitions with the same label either enter, exit, or never cross the boundary of the region. Each region

Function	Description
$timeFrame(s) : \mathcal{S} \rightarrow \mathbb{R}^+$	returns the interval of time when the system is in state s
$minTimeStamp(tr) : \mathcal{T} \rightarrow \mathbb{R}^+$	returns the first instant of time when transition tr fires
$avgTimeStamp(tr) : \mathcal{T} \rightarrow \mathbb{R}^+$	returns the average instant of time when transition tr fires
$maxTimeStamp(tr) : \mathcal{T} \rightarrow \mathbb{R}^+$	returns the last instant of time when transition tr fires
$\#(tr, t1, t2) : \mathcal{T} \times \mathbb{R}^+ \times \mathbb{R}^+ \rightarrow \mathbb{N}$	returns number of times transition tr fires in the interval $[t1, t2]$
$\#(l, t1, t2) : \mathcal{L} \times \mathbb{R}^+ \times \mathbb{R}^+ \rightarrow \mathbb{N}$	returns number of times transitions with label l fire in the interval $[t1, t2]$

Table 4.1: Examples of Predefined Functions that Can Be Used in the Metric Expression

in the LTS corresponds to a place in the derived SAN model, and each labelled transition in the LTS corresponds to an activity in the SAN model. A similar approach has already been used in other works to translate LTSs into Petri Nets (see, for instance, [36], [21] and [23]). In our approach, in order to have a well-defined probabilistic model, non-deterministic choices among k transitions outgoing from an LTS state are mapped in the SAN model into instantaneous activities with k case probabilities.

The metric is an arithmetic expression that may contain a predefined set of functions (see Table 4.1 for some examples). The guarantee is given by a boolean expression on the metric and a set of constraints on the CONNECTed system model (e.g., constraints on the time frame of evaluation of the metric and constraints on the behaviour of the Networked Systems). Statistical operators (e.g., $mean$ and $variance$), comparison and logical operators can be used in the expression. In order to uniquely identify states and transitions of the metric expression that appear in the generated SAN model, renaming is used to constrain the definition of the LTS regions.

4.2.2 Analyser Module

The prototype implementation of the Analyser is based on Möbius.

As already mentioned, in Möbius, each reward function is a C++ function that returns a value depending on the marking of the SAN.

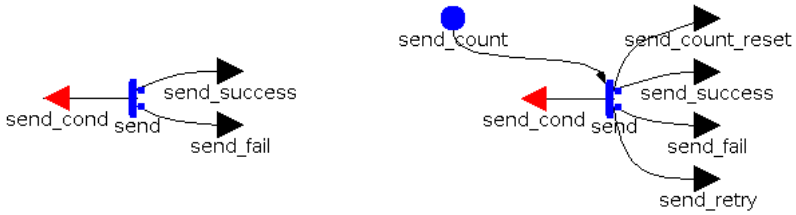
The reward functions are automatically derived from the metrics expression as follows: the metric is mapped into its syntax tree to decompose the metric into a combination of basic functions; the basic functions are translated into C++ functions by using a predefined repository of function templates. For instance, with reference to the functions shown in Table 4.1, a rate reward template is used to translate $timeFrame(s)$, while an impulse reward template is used to translate $\#(tr, t1, t2)$. A repository of function templates is planned for the next year; however, some details will be provided when presenting the case study in Section 4.4.

The Solver evaluates the reward functions via simulation. The quantitative assessment of the metric is obtained from the assessment of the reward functions by merging the results according to the arithmetic operations specified in the syntax tree of the metric expression.

4.2.3 Evaluator Module

The Evaluator compares the analysis results against the guarantees specified in the requirements, and reports a message containing a boolean value to Synthesis. The Evaluator may receive from Synthesis the following three types of enhancement requests:

- *alternative deployment*: upon receiving this request, the Evaluator triggers a new analysis that considers the updated annotated LTS specification of the CONNECTor contained in the request in the SAN model.
- *dependability mechanism*: upon receiving this request, the Evaluator triggers a sensitivity analysis that considers the impact of one failure mode at a time. Specifically, for each failure mode, a variant of the SAN models is generated such that the considered failure mode is the only one considered in the model. The ranking of the failure modes is obtained by assessing the metrics on the generated variants: the higher the impact on the guarantees dissatisfaction, the higher the ranking of the failure mode.



(a) Original model. (b) Enhanced model with message retransmissions.

Figure 4.4: Example of Dependability Mechanism and Application Rule

- *combined enhancement*: a request that allows a combined use of the previously mentioned enhancements. The combined enhancement request contains the annotated LTS specification of the CONNECTor for the alternative deployment, and an indication that the Evaluator module should also enable the evaluation of dependability mechanisms to improve the CONNECTor. They are reflected on the SAN model as indicated at the previous points. A field in the request is used to specify if the enhancements should be evaluated individually and/or in combination. When evaluated individually, an evaluation ordering is also provided.

4.2.4 Enhancer Module

The dependability mechanism suitable to contrast a given failure mode is determined through an ontology of dependability mechanisms, such as that reported in [4]. The definition of the content of this ontology is part of the work planned for the next year.

We developed ad hoc dependability and performance models for a set of relevant dependability mechanisms, and a set of rules to automate the application of the mechanisms in the SAN model of the CONNECTed system. The ad hoc models can be parametric; for instance, a retransmission mechanism is parametric with respect to the maximum number of allowed retransmissions. As an example of mechanism and application rule, in Figure 4.4 we graphically show a retransmission mechanism and how to modify the original model in order to apply message retransmissions to a send operation. Specifically, the original model contains a timed activity `send` that models the send operation, an input gate `send_cond` that specifies the enabling condition of the activity, and two output gates, `send_success` and `send_fail`, that specify the output functions in the case of correct and faulty behaviour. The enhanced model is obtained from the original model by adding the following elements: a place `send_count`, with initial marking the maximum allowed number of retransmissions; an output gate `send_count_reset`, which resets the marking of `send_count` to its initial value when the `send` succeeds; an output gate `send_retry`, which reactivates `send` as long as `send_count` contains tokens, and resets the marking of `send_count` after performing all retransmission attempts.

4.2.5 Selector Module

With reference to the Selector module included in the the architecture of the DePer (see Figure 4.2), some basic criteria that can be used to select the Möbius engine on the basis of the CONNECTed system specification and of the requirements follow.

- *Requirement expressed on* CONNECT*ed systems with large number of actors.* This information can be obtained by inspecting the constraint reported in the *guarantee expression* (a boolean expression that is required to be satisfied on the metrics under a set of constraints. This concept is better described later when dealing with the *Builder* module of the Dependability&Performance Analysis Engine).

- *Specification includes distributions different from exponential.* This information can be obtained by inspecting the distribution reported for each transitions in the non-functional part of the specification.

- *Specification mixes discrete probabilities and continuous time distributions.* This information can be obtained by inspecting the failure probability reported in the non-functional part of the specification.

4.3 PRISM-based Prototype Implementation

PRISM [56] is a popular probabilistic model checker, which can handle discrete and continuous time Markov chain models (DTMCs and CTMCs), as well as Markov decision processes (MDPs). DTMC and MDP models may be verified against probabilistic temporal logic formulae given in terms of PCTL (Probabilistic Computational Tree Logic) [54, 17], as well as cost/reward-based properties, and LTL (Linear Temporal Logic) formulae [86]. CTMCs may be verified against CSL (Continuous Stochastic Logic) [9, 10] formulae. Both states and transitions in a system can be associated with rewards, which allow for the checking of both instantaneous and cumulative properties.

So far PRISM has been applied to numerous probabilistic models, such as network protocols, security protocols, randomised distributed algorithms, biological processes, etc.

Modeling formalism. As CONNECTed systems evolve in a manner where a system stays in a state for a certain period of time and then moves to a successor state, we model such systems using CTMCs because they preserve the memoryless property. For CTMCs, the memoryless property not only requires that the probability of firing a transition totally depends on the current state, but also asks the probability to be independent of the elapsed time so far. The only continuous probability distribution exhibiting this property is the exponential distribution, which associates a *rate* to each transition in CTMCs. The rate can be understood as the average number of times we can execute the transition per unit of time. The probability of executing a transition from the current state within t time units is $1 - e^{-\lambda \cdot t}$. The rates associated with all transitions in a CTMC can be stored in a *transition rate matrix* $\mathbf{R}$, where each entry represents a rate between a pair of states. A transition can only occur from state s to state s' if $\mathbf{R}(s, s') > 0$. If more than one transition can be executed in state s, the successor state is determined by the first transition being taken. Let S be the set of states in a CTMC. The amount of time for which the system stays in s before any transition occurs is governed by an exponential distribution with rate $E(s)$ such that $E(s) \stackrel{def}{=} \sum_{s' \in S} \mathbf{R}(s, s')$. The probability of going to successor state s' from state s is calculated as follows.

$$\mathbf{P}(s, s') = \begin{cases} \mathbf{R}(s, s')/E(s) & \text{if } E(s) \neq 0, \\ 1 & \text{if } E(s) = 0 \text{ and } s = s', \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

Properties of interest. Using formula (4.1), we can compute the probability of reaching a set of target states through all paths. *Steady-state* behaviour is another interesting property for CTMC models. The steady-state probability for a state s is the probability of being in s in the long run, which can be used to infer the percentage of time that the model spends in s in the long run.

In addition to path and steady-state probabilities, we consider two additional types of reward for instantaneous and cumulative rewards separately. Every transition is associated with an instantaneous reward and every state has a cumulative reward. The former is the actual reward obtained when the system executes a transition, and the latter is the coefficient, at which the reward is computed in a state, for the amount of time spent in that state. We can define the expected reward of reaching a set of target states F through paths. The reward for a path that does not pass any target state is set to ∞. Thus, the expected reward of reaching a state in F from state s is finite if all non-zero probability paths starting from s pass a state in F.

4.3.1 Builder Module

This module translates the LTS of the CONNECTed system into a CTMC model in the format recognised by PRISM. The translation is straightforward, as a CTMC model is an LTS whose transitions are associated with an exponential distribution. Metrics defined on the model are mapped to corresponding states and transitions.

4.3.2 Analyser Module

This module augments the CTMC model with rewards according to the metrics and generates temporal logic formulae to specify the properties. The verification of the properties is done by PRISM using probabilistic model checking techniques. Currently, only the dependability and performance properties defined in Section 2.3, i.e., coverage and latency, can be dealt with. Process of general properties will be investigated next year.

4.3.3 Evaluator Module

In addition to reporting analysis results to Synthesis Enabler, this module can only respond to requests of *alternative deployment*, as CTMC models do not allow the mixture of continuous and deterministic probability.

4.3.4 Enhancer Module

Due to the same reason as above, we do not enhance CTMC models to add failure mode supported in Möbious. However, as shown in Section 5.1, a CTMC model can be converted into an MDP model, it is possible to provide support to failure mode on MDPs, which will be studied next year.

4.3.5 Selector Module

With reference to the Selector module included in the the architecture of the DePer (see Figure 4.2), some basic criteria that can be used to select the PRISM engine on the basis of the CONNECTed system specification and of the requirements follow.

- *Requirement using next / bounded until operators.* This information can be obtained by inspecting the dependability and performance properties in the specification.
- *Requirement on the steady state of the* CONNECT*ed system.* This information can be obtained by inspecting the dependability and performance properties in the specification.
- *Specification enables assume-guarantee reasoning.* Currently the assume-guarantee reasoning method is still under development in WP2. How to select this verification technique will be understood better in the future deliverable.

4.4 Application to the Terrorist Alert Scenario

In this section, we model the scenario described in Chapter 2 using PRISM and Möbius respectively, and perform various dependability and performance analysis on the models, thus showing their complementarity in assessing dependability and performance properties. A similar analysis, documented in [45], has been also previously developed during the second year with reference to a distributed market place scenario, another CONNECT scenario defined during the first year. First, both approaches are used to validate two basic dependability and performance properties. Next, extra properties are checked by the appropriate approach, selected according with its ability to cope with the specific type of analysis. Indeed, the different formalisms and tools implied by the two methods allow: (i) on the one hand, to complement the analysis from the point of view of a number of aspects, such as level of abstraction/scalability/accuracy, for which the two approaches may show different abilities to cope with; and (ii) on the other hand, through the inner diversity, provide cross-validation to enhance confidence in the correctness of the analysis itself.

4.4.1 PRISM Approach

In this section, first we show the PRISM models of the case study under analysis, then we show the stochastic verification results obtained for dependability and performance properties.

Models

The LTSs for the case study in Chapter 2 can be translated into the PRISM CTMC model in a straightforward manner. In detail, each component LTS in Figure 2.3, 2.5, and 2.6 is translated into a PRISM module in the following way. We define a variable in each module, whose domain is the set of states in the corresponding LTS and whose initial value is the initial state of the LTS. Each transition in the module has the same label as the corresponding one in the LTS. Since the LTSs do not contain information for rates, we deliberately assign rate $R1 = 1$ to all transitions between the control center and the CONNECTor, and assign $R2$ (which may vary) to those between the CONNECTor and the guards.

In order to check the dependability and performance properties specified in Section 2.3, we need to add the timeout mechanism to the model. A timeout T is introduced to model the maximum time that the CONNECTor can wait for the `eAck` message from the guards. After the CONNECTor broadcasts the `emergencyAlert` message, a timeout will be triggered after T units of time if not all guards sends back the `eAck` message. When the timeout occurs, the CONNECTor does not wait for pending responses from guards, and returns `uploadSuccess` immediately to the control center. However, the deterministic delay in timeout breaks the basic rule of CTMCs: all delays in a CTMC model respect exponential distributions, and this makes the model difficult to verify. Therefore, we use an Erlang distribution to approximate a deterministic delay T by a sequence of transitions, each of which has an exponential distribution of rate k/T, where k is the number of transitions in the sequence. The accuracy of the approximation, as well as the verification time, increases as k increases. In the experiments, we choose k to be $T \times 10$, i.e., the rate in the Erlang distribution is 10, which is a reasonable trade-off between speed and accuracy.

Stochastic Verification

Now we show the verification results for the dependability (coverage) and performance (latency) properties. For each property, we construct a set of experiments by choosing different values for timeout T, and letting $R2$ range over values 0.1, 0.5 and 1.0 respectively. This way, we can illustrate the trend as T increases.

Coverage. This property is specified by the following CSL reward formula on the reward structure *Coverage*:

$$\mathcal{R}\{\text{“}Coverage\text{”}\}_{=?}[S], \tag{4.2}$$

where S represents the reward in steady states. The structure *Coverage* associates the real value m/n to states where, among the total number n of commanders and guards, m of them send back their response to the connector within T time units after they receive the request `emergencyAlert`. The verification results are presented in Figure 4.5, which shows that as the length of waiting time T for the `eAck` messages increases, the percentage of `eAck` messages received increases and eventually approaches 100% in all cases, i.e., $R2 = 0.1, 0.5$ and 1.0. But $R2 = 1.0$ sees the fastest increase, while $R2 = 0.1$ the slowest. In other words, the transmission speed for $R2 = 1.0$ is the fastest and that for $R2 = 0.1$ is the slowest.

Latency. To verify this property, we use formula (4.3) on the reward structure *Latency*,

$$\mathcal{R}\{\text{“}Latency\text{”}\}_{=?}[Fp], \tag{4.3}$$

which assigns $1/(k/T) = 1/10$ to each transition used to approximate the timeout. The latency is measured from the moment when the control center starts to send the initial request `selectArea` to the time it receives `uploadSuccess`. The results are depicted in Figure 4.6. As T increases, the latency increases and becomes constant in the end. It is in conformance with Figure 4.5 and shows that $R2 = 1.0$ is the fastest, and hence has the shortest delay, while $R2 = 0.1$ is the slowest and has the longest delay.

It is also useful to know the trend of the amount of time spent on waiting for `eAck`, given different values of T. The formula used to check this is identical to formula 4.3 except that it is based on a different reward structure *Latency2*. The results are illustrated in Figure 4.7, which shows the same trend as in Figure 4.6 except it also takes into account in the former the time before the CONNECTor starts to wait for `eAck`.

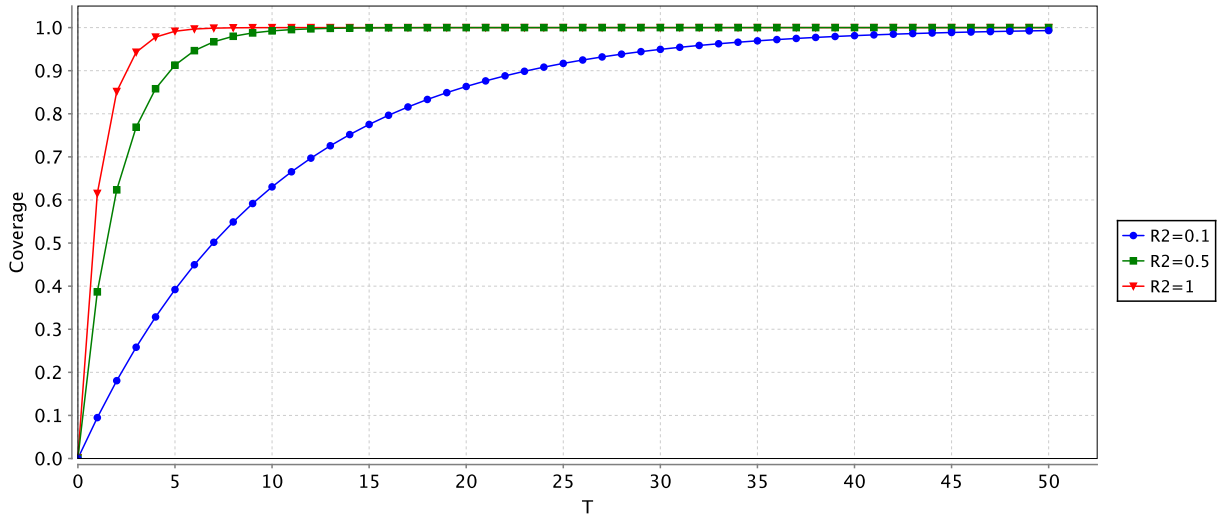


Figure 4.5: Verification Results for Coverage

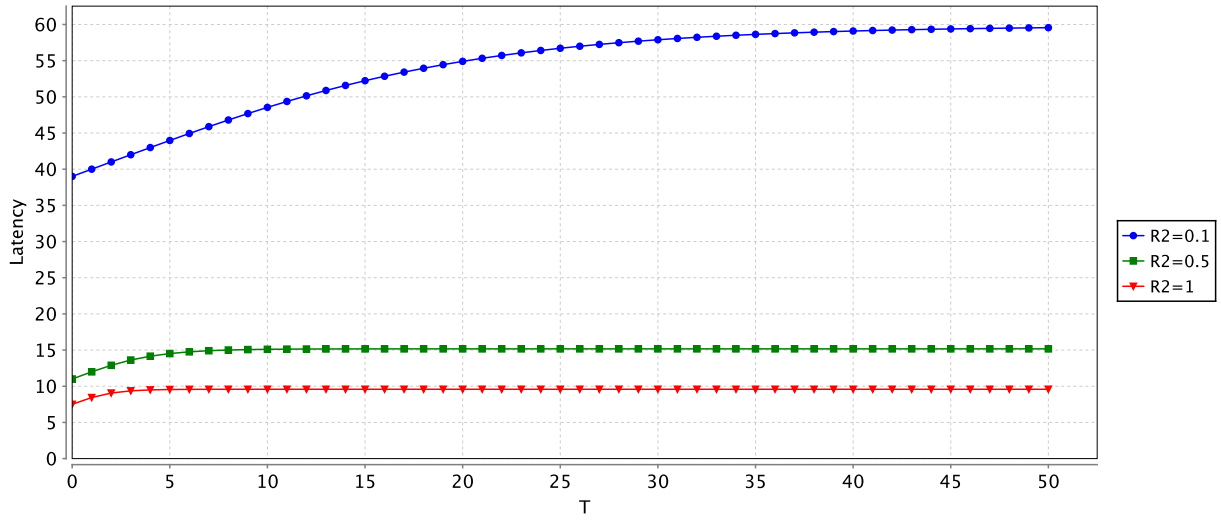


Figure 4.6: Verification Results for Latency (1)

4.4.2 Möbius Approach

In this section, first we show the SAN models of the case study under analysis, then the results of the analysis obtained through Möbius.

SAN Models

The SAN models of guard, commander, CONNECTor, and SecuredFileSharing are shown in Figure 4.8. The model of the CONNECTed system is obtained by composing, via place sharing, the SAN models of SecuredFileSharing, commander, CONNECTor and guards (the SAN model of the guards is obtained by replicating a guard with the Rep operator). There is a shared place for each pair of activities that represent send/receive actions: send activities add tokens in the shared place, while receive activities remove tokens from the shared place and use the marking of the shared place as enabling condition. Note that, in general, a send activity may control $n > 1$ receive activities (e.g., in the case of a message with multicast/broadcast addresses); in this case, the send activity will add n tokens to the shared place

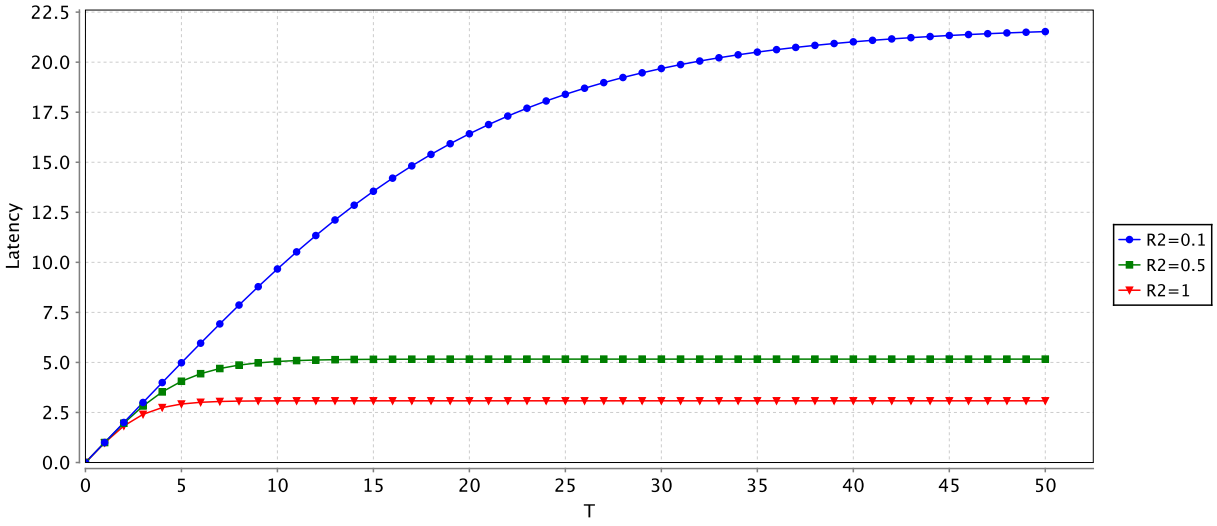


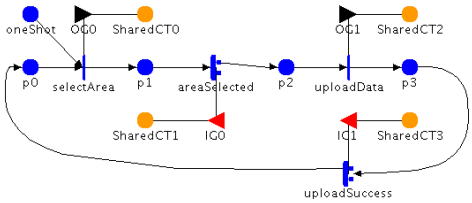
Figure 4.7: Verification Results for Latency (2)

to allow the simultaneous enabling of the receive activity of n receivers.

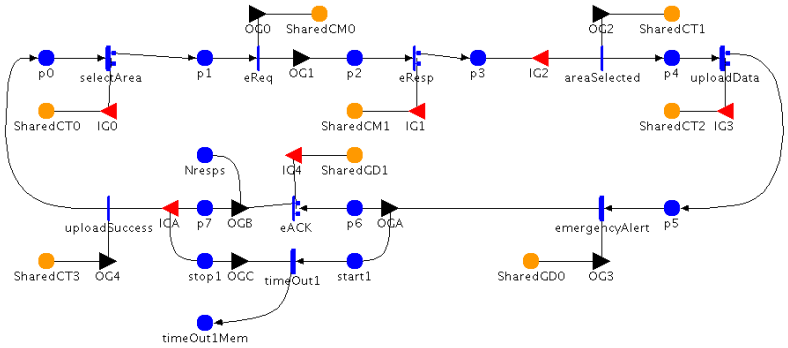
Timing aspects for send/receive actions are taken into account in the SAN models as follows: when n receive activities complete simultaneously after a send action completes, the receive activities are instantaneous and the send activity is timed; when n receive activities complete independently after a send action completes, the receive activities are timed and the send activity is instantaneous. Timeouts are modeled with timed activities that force the enabling of other activities.

In the following we describe in detail the behaviour of the model of CONNECTed system. In the description, we will use the prefixes `C`, `G`, `CON`, and `S` to disambiguate the names of local places, activities and gates of commander, guards, CONNECTor, and SecuredFileSharing.

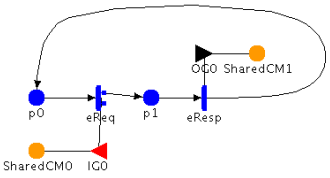
Initially, all places in the models have zero tokens, except `p0`, which contains one token in all models. The SecuredFileSharing starts the communication, because `S.selectArea` is the only enabled activity. When `S.selectArea` completes, one token is placed in `S.p1` and one token in `SharedCT0`. At this point, `S.selectArea` is enabled. When `S.selectArea` completes, one token is placed in `S.p1` and the number of tokens in `SharedCT0` is increased. The activity `CON.selectArea` is now enabled, when it completes one token is moved from `SharedCT0` to `CON.p1`, and `CON.eReq` becomes enabled. When `CON.eReq` completes, the marking changes as follows: $commNum$ tokens are placed in `SharedCM0`, because $commNum$ commanders must be involved in the communication; $commNum$ tokens are placed in `CON.p2`, because the CONNECTor must wait for one `eResp` from each commander. When the CONNECTor receive a response from each commanders, (i) for each response received one token is placed in `CON.p3`; (ii) when each commanders has sent a response `CON.areaSelected` is enabled, one token is placed in `CON.p4` and the number of tokens in `SharedCT1` is increased. At this point `S.areaSelected` is enabled, when it completes one token is moved from `SharedCT1` to `S.p2`, and `S.uploadData` becomes enabled. A token is placed in `S.p3` and the number of token in `SharedCT2` is increased. Activity `CON.uploadData` is now enabled, when it completes one token is moved from `SharedCT2` to `CON.p5`, which enables the activity `emergencyAlert`. When `emergencyAlert` completes $commNum + guardNum$ tokens are placed both in `SharedGD0` and `CON.p6`, and the number of tokens in `CON.start1` is increased. At this point activities `CON.timeOut1` and `G.emergencyAlert` are both enabled. The first one represents the CONNECTor's timeout on the maximum waiting time; while the second one enables the activity `G.eACK` which increases the number of tokens in `SharedGD1`. At this point activity `CON.eACK` is enabled and the number of tokens in `CON.p7` and `CON.Nresps` is increased, until the timed activity `CON.timeOut1` completes. The activity `uploadSuccess` becomes enabled when $commNum + guardNum$ tokens are placed in `CON.p7`, this means that the CONNECTor has received all responses, or when the number of tokens in `CON.stop1` is greater than zero, this means that the time associated to the activity `timeOut1` has elapsed, `CON.timeOut1` completes. The number of tokens in `CON.Nresps` represents the number of guards that have recived the `emergencyAlert` and have



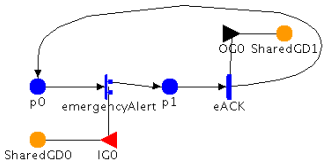
(a) SecuredFileSharing



(b) CONNECTor



(c) EmergencyCall, commander



(d) EmergencyCall, guard

Figure 4.8: SAN Models

sent back the `eACK` before the timeout.

State-based Stochastic Analysis.

The analysis performed through Möbius consists in: i) cross-validation of the results obtained by PRISM for coverage, latency, and latency2; ii) scale up to large systems with hundreds of guards; iii) coverage in case of failure, not accounted for in the previous analysis with PRISM.

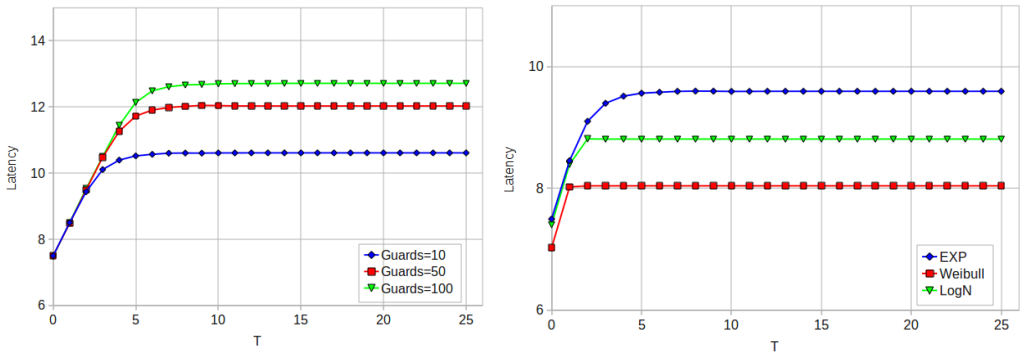
Cross validation. The reward functions are expressed as follows.

Coverage. This property is specified by accumulating over time the following impulse reward on `CON.uploadSuccess` (`guardNum` and `commNum` are two parameters of the composed model, and hold the number of guards and commanders respectively):

```
double coverage() {
 return ( (double) connector->Nresps->Mark() ) / ( guardNum + commNum );
}
```

Latency. This property is specified by accumulating over time the following rate reward function:

```
double latency() {
    if ( SecuredFileSharing->p1->Mark() > 0 || SecuredFileSharing->p2->Mark() > 0
 || SecuredFileSharing->p3->Mark() > 0 ) { return 1; }
}
```



(a) Latency for different number of guards

(b) Latency for different traffic patterns

Figure 4.9: Latency for Different System Size and Different Traffic Patterns

Latency2. This property is specified by accumulating over time the following rate reward function:

```
double latency2() {
    if ( connector->start1->Mark() > 0 && connector->p6->Mark() > 0 ) { return 1; }
}
```

We were able to successfully reproduce through Möbius the verification results of PRISM. We used simulation, and the relative difference between the average results was always below 2%.

Scalability of the models. CONNECTed systems may include an arbitrary large number of Networked Systems. Therefore, we investigated the scalability of the SAN model of the CONNECTed system by analysing large networks. The developed SAN model of the CONNECTed system is parametric with respect to the number of guards and commanders.

We successfully assessed coverage and latency for scenarios with hundreds of guards and two commanders. Figure 4.9(a) shows the analysis results for latency in scenarios with at most 100 guards. The number of batches needed to reach a confidence level of 95% and a confidence interval of 10% for the considered models was always below $10K$, because the models are relatively simple.

Latency for different traffic patterns. CONNECTed systems are expected to be a mix of heterogeneous user applications, each of which may have different characteristics and requirements. Currently, there is no single traffic distribution that can efficiently capture the traffic characteristics of all types of networks under every possible situation. A large number of empirical studies have shown that network traffic is self-similar and that it generally exhibits multiple time-scale behaviour [69]. These aspects can be modeled with subexponential distributions, such as Weibull and Lognormal.

We investigated the effect of different subexponential distributions on latency by changing the probability distribution function of the timed activities. For a fair comparison, we have chosen distribution parameters that allow the same mean value in all cases. The analysis results are shown in Figure 4.9(b). We can notice that different traffic patterns lead to different latency profiles.

Coverage in the case of failures. Communication in the real-world can be subject to failures. Therefore, failure modes need to be accounted for when setting up the system model. Failure modes can pertain the value domain (e.g., wrong output), and/or the time domain (e.g., omission). In this section, we assess coverage in the case of omission failure of the messages sent and received in the EmergencyCall application. Figure 4.10 shows the coverage profiles for different probability *P(ECallFailure)* of failures of `EmergencyCall` communications. The analysis is performed with two commanders and two guards.

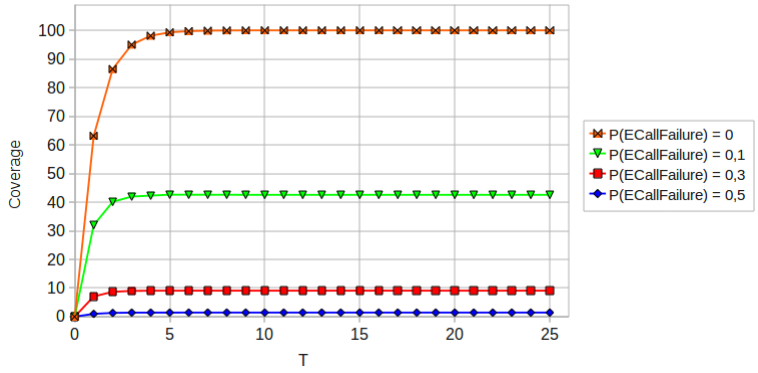


Figure 4.10: Coverage for Different *P(ECallFailure)* of Failures of `EmergencyCall` Communications

4.5 Conclusions and Further Research Directions

During this second year, the work on dependability and performance analysis concentrated on the definition and partial implementation of the DePer Enabler, which employs the two approaches of state-based stochastic model and stochastic model-checking to provide probabilistic quantitative assessment of non-functional metrics of CONNECTed systems. Initial prototypes based on PRISM and Möbius have been developed and practically applied to the analysis of a few metrics for the "Terrorist Alert" scenario, through which the complementarity of the two approaches has been shown (refer to the Appendix-Prototypes for the URL from which a first release of DePer is made available).

The enhancements/extensions planned for the third year mainly consist in: i) progressing in the implementation of the Enhancer module of DePer, by taking into account in the analysis basic fault-tolerance mechanisms/patterns, among an available set, to react to selected failure modes experienced by the CONNECTed system. This is a very important feedback to the Synthesis Enabler towards the synthesis of a dependable CONNECTor; ii) progressing in completing the loop between the DePer and the Synthesis Enabler according to Figure 4.1, so that the Synthesis can profitably embed in the synthesis of the CONNECTor the indications coming from the analysis; iii) progressing in the interactions between the DePer and the Monitoring Enabler (presented in the next chapter), to refine the analysis along time using real data/events observed at run-time.

5 Incremental and Runtime V&V

The analysis at the early stage of the CONNECT synthesis process, as described in Chapter 4, is of paramount importance to achieve the required functional and non-functional properties. Nevertheless, the incomplete a priori knowledge about the middleware and environment, as well as the continuous evolution of the context and the services with which a NS interacts, unavoidably undermine the accuracy of the considered parameters and information and, hence, may invalidate the analysis results.

Model-based analysis, such as the one supported by the approaches described in the previous chapter, is a well-established method to assess the dependability of a system before deployment. However, it is well known that, in highly dynamic contexts, such as the one we operate in CONNECT, the accuracy of the analysis results can be limited or become obsolete, because unpredictable phenomena may affect the system during its operation. Therefore, the analysis typically needs to be refined with data obtained from real system executions.

In this chapter, we report on two directions currently undertaken for the progressive and continuous refinement of dependability analysis. More precisely, we present first in Section 5.1 an approach for incremental verification in order to improve the performance of verification at run time by reusing results from previous verifications to obtain fast accurate results during the evolution of a CONNECTed system. The main contribution is that (1) we improve the efficiency of SCC-based probabilistic model checking [26], and (2) we minimise the computation cross multiple verifications using the improved SCC-based probabilistic model checking technique. Then, in Section 5.2 we show how –through the interplay between *DePer* Enabler and the GLIMPSE Monitor (introduced in [34])– we can refine model-based dependability analysis in CONNECT. We focus on the synergic use of an automated approach to perform model-based dependability analysis with runtime monitoring of CONNECTor executions. We have released prototypes for both incremental verification and GLIMPSE; their respective URLs and related info can be found in the Appendix-Prototypes.

Finally, for both incremental verification and runtime analysis, examples of application are shown for the Terrorist Alert scenario (see Sections 5.1.4 and 5.2.3, respectively). Chapter conclusions are drawn in Section 5.3.

5.1 Incremental Verification of CONNECTed Systems

In the previous chapter, we have applied stochastic model checking to the analysis of CONNECTed systems for the verification of dependability properties. As a CONNECTed system may evolve, the analysis has to be repeated in order to incorporate changes in the system provided by the monitor Enabler (see Section 5.2.2 for more detail). If the analysis is always started from scratch for an evolved system, i.e., using offline verification techniques summarised in the deliverable D2.1 [29], the significant time overhead makes the analysis less usable, particularly when subsequent changes occur before the verification terminates.

In this section, we propose an incremental verification technique that can reuse the existing verification results when the verification needs to be done again due to the changes in the system. This technique is based on the offline verification techniques, but reduces the computation time for unchanged part of a system during evolution. Therefore, it is more suitable than the latter for run-time analysis. In the rest of the section, we focus on *Markov decision processes* (MDPs), a common modeling formalism for systems that exhibit both probabilistic and nondeterministic behaviour, and formulate incremental stochastic verification techniques. Our methods at present target CONNECTed systems which are subject to changes in probability values at run time. This technique also applies to CTMC models, which are used in the previous chapter, as the verification of a CTMC is performed by converting the CTMC into a MDP first and then model checking the MDP. The detail can be seen in Deliverable D2.1 [29].

Offline stochastic verification techniques typically require a combination of graph-based analysis techniques and iterative numerical solution methods. Our method also improves their efficiency, and consequently, makes the run-time analysis even faster. In [26], improvements to efficiency have been proposed based on a decomposition of the model into *strongly connected components* (SCCs). We improve on the work of [26] by showing that SCC decomposition can be used to optimise the verification of an MDP in two ways: (i) by reducing the amount of graph-based analysis required; and (ii) by parallelising the analysis

of separate SCCs. Note that the technique applies to both probability and reward properties, but we only present the detail for the former for simplicity. The results reported in the section for experiments are produced using the reward formula presented in Section 4.4.1.

5.1.1 Preliminaries

We let $Dist(S)$ be the set of all discrete probability distributions over S, i.e., the set of functions $\mu : S \to [0,1]$ such that $\sum_{s \in S} \mu(s) = 1$.

Markov Decision Processes

Markov decision processes (MDPs) capture both probabilistic and nondeterministic behaviour. They can be very useful to model concurrency, e.g., several probabilistic processes operating in parallel, or when the exact probability distribution of some behaviour is not known or is not relevant. An MDP is formally defined as follows.

Definition 1 (MDP) *An MDP is a tuple* $\mathcal{M} = (S, \overline{s}, Act, Steps, L)$ *where*

- S *is a finite set of* states*,*
- $\overline{s} \in S$ *is the* initial state*,*
- Act *is a set of* actions*,*
- $Steps : S \to 2^{Act \times Dist(S)}$ *is the* probabilistic transition function*.*
- $L : S \to 2^{AP}$ *is a* labelling function*,*

The transition probability function $Steps$ maps each state $s \in S$ to a finite non-empty set $Steps(s)$ of action-distribution pairs.[1] There are two steps to determine the successor of a state s in the MDP: first, an action-distribution pair (a, μ) is chosen non-deterministically from the set $Steps(s)$; second, the next state s' is chosen randomly according to μ, i.e. the probability of moving to each state s' is given by $\mu(s')$.

A *path* in an MDP is a non-empty (finite or infinite) sequence of the form

$$s_0 \xrightarrow{(a_0,\mu_0)} s_1 \xrightarrow{(a_1,\mu_1)} s_2 \ldots$$

where $s_i \in S$, $(a_i, \mu_i) \in Steps(s_i)$ and $\mu_i(s_{i+1}) > 0$ for all $i \geq 0$. We use $\omega(i)$ to denote the ith state in the path ω, and $step(\omega, i)$ for the ith action in ω. For a finite path ω_{fin}, we denote the last state in ω_{fin} as $last(\omega_{fin})$. $Path_s^{fin}$ ($Path_s$) are the sets of all finite (infinite) paths starting in state s.

An *end component* [43] $\mathcal{M}'$ of $\mathcal{M}$ is an MDP $\mathcal{M}' = (S', \overline{s'}, Act', Steps', L')$ where $S' \subseteq S$, $Act' \subseteq Act$, and for each $s \in S'$, we have $L'(s) = L(s)$, $Steps'(s) \subseteq Steps(s)$, satisfying the following two conditions:

1. for each $(a, \mu) \in Steps'(s)$, $\forall s' \in S \,.\, (\mu(s') > 0 \to s' \in S')$.
2. for each $s' \in S'$, there exists a path in $\mathcal{M}'$ from s to s'.

Note that we do not care about $\overline{s'}$ in $\mathcal{M}'$.

To resolve the non-deterministic choices when we execute an MDP, we employ an *adversary* to select an action-distribution pair based on the history of choices made so far.

Definition 2 (Adversary) *An adversary* A *of an MDP* $\mathcal{M}$ *is a function mapping every finite path* ω_{fin} *onto an element* $A(\omega_{fin})$ *of the set* $Steps(last(\omega_{fin}))$*. Let* $Adv_{\mathcal{M}}$ *denote the set of all possible adversaries of the MDP and, for any adversary* A*, let* $Path_s^A$ *denote the subset of* $Path_s$ *which corresponds to* A*.*

Model checking MDPs

Usually, properties to be verified against MDPs are expressed in temporal logics, such as PCTL [54, 17] and LTL [86, 41]. The most important task for model checking such logics is to compute the *minimum* or

[1] In some presentations, it is assumed that each element of $Steps(s)$ has a unique action. In this setting, this distinction is not important.

maximum reachability probabilities, i.e. the minimum or maximum probability that a path through the MDP eventually reaches a state in some target set $F \subseteq S$, quantified over all possible adversaries:

$$\begin{aligned} p_s^{\min} &= \inf_{A \in Adv_{\mathcal{M}}} \left(Prob_s^A(\{\omega \in Path_s^A \mid \omega(i) \in F \text{ for some } i\}) \right) \\ p_s^{\max} &= \sup_{A \in Adv_{\mathcal{M}}} \left(Prob_s^A(\{\omega \in Path_s^A \mid \omega(i) \in F \text{ for some } i\}) \right) \end{aligned}$$

We use $\underline{p}^{\max}$ to denote the vector of probabilities $p_s^{\max}$ for all states $s \in S$.

In the remainder of Section 5.1, we will focus on the case of maximum probabilities, $p_s^{\max}$, but our techniques can easily be adapted to the case of minimum probabilities, Reachability probabilities can be computed using a variety of standard techniques, including value iteration, linear programming and policy iteration (see e.g. [87] for details.)

In practice, value iteration techniques have been adopted most widely since they scale better to large MDPs. In this method, the transition relation is encoded by a matrix and the probabilities for states are stored in a vector. The vector is iteratively updated by multiplying the matrix and vector, and applying the maximum function to decide the maximum probability value for states.

Definition 3 (Value iteration) *Let $\mathcal{M} = (S, \overline{s}, Act, Steps, L)$ be an MDP and $F \subseteq S$ a set of target states. The sequence of vectors $\langle \underline{p}^{\max,n} \rangle_{n \in \mathbb{N}}$, given below, converges to the vector $\underline{p}^{\max}(F)$ of maximum reachability probabilities for F in $\mathcal{M}$. We define, for any state $s \in S$:*

$$p_s^{\max,n} = \begin{cases} 1 & \text{if } s \in F \\ 0 & \text{if } s \notin F \text{ and } n = 0 \\ \max\limits_{(a,\mu) \in Steps(s)} \sum_{s' \in S} \mu(s') \cdot p_{s'}^{\max,n-1} & \text{otherwise.} \end{cases}$$

In practice, value iteration is carried out by computing the vector of required values for increasing values of n, until a pre-specified convergence criterion is met. One common approach is to check that the maximum difference between the corresponding elements of successive vectors is below some fixed threshold (also known as maximum absolute difference) δ. Another is to use the maximum relative difference of vector elements.

It is also common in practice to use *precomputation* techniques, which partition S into sets S^{no}, S^{yes} and $S^{?}$, containing states for which the probability is 0, 1 or in $(0, 1)$, respectively. Subsequently, value iteration (or other techniques) can be applied to determine the exact values for states in $S^{?}$. For p_s^{max}, we have

$$\begin{aligned} S^{no} &= Prob0A(Sat(\phi)), \\ S^{yes} &= Prob1E(Sat(\phi)), \\ S^{?} &= S \backslash (S^{yes} \cup S^{no}). \end{aligned}$$

Algorithm $Prob0A$ [17] first computes the set of states, each of which can reach a state in $Sat(\phi)$ with probability greater than zero. Then it returns the complement of this set under S as S^{no}. For each state $s \in S^{no}$, no path in $Path_s^A$ satisfies the formula $F\ \phi$ with non-zero probability under any adversary A.

Algorithm 1: $Prob0A(Sat(\phi))$

1: $R := Sat(\phi)$; $done := \mathbf{false}$
2: **while** $done = \mathbf{false}$ **do**
3: $\quad R' := R \cup \{s \in S \mid \exists (a, \mu) \in Steps(s)\ .\ \exists s' \in R\ .\ \mu(s') > 0\}$
4: $\quad$ **if** $R' = R$ **then** $done := \mathbf{true}$ **end if**
5: $\quad R := R'$
6: **end while**
7: **return** $S \backslash R$

Algorithm $Prob1E$ [43] computes a double fixpoint to return the set S^{yes} of states, each of which has $p_s^A(F\ \phi) = 1$ for some adversary A. The outer loop identifies states from which no adversary can make $p_s^A(F\ \phi) = 1$, and remove those states from S. The inner loop collects states from which one cannot reach a state in $Sat(\phi)$ without passing through a state already removed from S.

Algorithm 2: $Prob1E(Sat(\phi))$

$R := S$; $done := \mathbf{false}$
while $done = \mathbf{false}$ **do**
  $R' := Sat(\phi)$; $done' := \mathbf{false}$
  while $done' = \mathbf{false}$ **do**
    $R'' := R' \cup \{s \in S \mid \exists (a,\mu) \in Steps(s) \,.\, (\forall s' \in S \,.\, \mu(s') > 0 \rightarrow s' \in R) \wedge (\exists s' \in R' \,.\, \mu(s') > 0)\}$
    if $R'' = R'$ **then** $done' := \mathbf{true}$ **end if**
    $R' := R''$
  end while
  if $R' = R$ **then** $done := \mathbf{true}$ **end if**
  $R := R'$
end while
return R

SCC-based value iteration

In this section, we introduce an optimisation presented in [26] for value iteration. The first step is to compress maximal end components (MECs) in the model in question. An MEC $\mathcal{M}'$ with the set S' of states is an EC that is not contained in any larger EC $\mathcal{M}''$ with the set S'' of states such that $S' \subset S''$. It has been proved in [26] that all states in an MEC have the same probability value when value iteration terminates. Therefore, we can replace each MEC by a single state. A self-loop is an action that, with probability one, reaches the state where it is enabled. In the rest of Section 5.1, we assume there are no MECs nor self-loops in an MDP. However, there might still be *strongly connected components* (SCCs) in the model. An SCC is a set of states such that there exists a path between any two states in the SCC. From now on, we assume each SCC is maximal.

SCCs are particularly important in value iteration. Let SCC be an SCC, and $\overline{SCC} \subseteq S \backslash SCC$ be the set of states that can reach SCC. Any change of a state's probability value in SCC affects probability values of all other states in SCC, as well as those of states in $\overline{SCC}$. Furthermore, until the probability values of the states in SCC converge, the probability values of states in $\overline{SCC}$ cannot be stabilised. Therefore, the computation of probability value for states in $\overline{SCC}$ can be postponed until the probability values in SCC converge.

The set S of states in M is partitioned into SCCs. Let $\overline{S}_{SCC} = \{SCC_1, \ldots, SCC_m\}$ be the partition. The *successor set* $Succ(SCC_i)$ of SCC_i is a maximal set of states such that $Succ(SCC_i) \cap SCC_i = \emptyset$, and, for each state $s \in Succ(SCC_i)$, there exists a state $s' \in SCC_i$ such that $\exists (a,\mu) \in Steps(s') \,.\, \mu(s) > 0$. We say that SCC_i *depends* on SCC_j if $Succ(SCC_i) \cap SCC_j \neq \emptyset$. As there is no cyclic dependence among SCCs, we generate a *reversed topological order* $\overline{S}_{SCC}$ among SCCs such that SCC_j will appear before SCC_i in $\overline{S}_{SCC}$ if and only if SCC_i depends on SCC_j.

The SCC-based value iteration starts to process SCCs along the order $\overline{S}_{SCC}$ and terminates when it reaches the end of the order. For each state $s \in SCC_i$, let $p_s^{\max,k}$ be the maximum probability after the k-th iteration, $p_s^{\max}$ the final value after the iteration on SCC_i terminates, and $p_s^{\max,0}$ the initial value, which is one if $s \in F$, or zero otherwise. The value iteration on SCC_i is split into two steps: the first iteration and the remaining iterations. The first iteration is performed as follows:

$$p_s^{\max,1} = \begin{cases} \max\limits_{(a,\mu) \in Steps'(s)} \sum_{s' \in S} \mu(s') \cdot p'_{s'} & \text{if } p_s^{\max,0} < 1 \text{ and } Steps'(s) \neq \emptyset \\ p_s^{\max,0} & \text{otherwise,} \end{cases}$$

where $Steps'(s) = \{(a,\mu) \in Steps(s) \mid \exists s' \in Succ(SCC_i) \,.\, \big(\mu(s') > 0 \wedge p_{s'}^{\max} > 0\big)\}$ and $p'_{s'}$ is $p_{s'}^{\max}$ if $s' \in Succ(SCC_i)$, or $p_{s'}^{\max,0}$ otherwise.

In the remaining iterations, we only update probabilities for those states that are affected by the previous iteration. Other states simply keep their probability from the previous iteration. The k-th ($k > 1$) iteration is described in Algorithm 3.

In the above algorithm, $p'_{s'} = p_{s'}^{\max}$ if $s' \in Succ(SCC_i)$; else $p'_{s'} = p_{s'}^{\max,k-1}$. The iteration on SCC_i terminates at the k-th iteration when X in Algorithm 3 is empty. Note that Algorithm 3 also works when we use δ as maximum relative difference, e.g., the condition $p_x^{\max,k-1} - p_x^{\max,k-2} \geq \delta$ in Algorithm 3 can

Algorithm 3: The k-th Iteration

1: $X \Leftarrow \{x \in SCC_i \mid p_x^{\max,k-1} - p_x^{\max,k-2} \geq \delta\}$
2: **for all** $x \in X$ **do**
3: $\quad Y \Leftarrow \{y \in SCC_i \mid p_y^{\max,k-1} < 1$ and $\exists(a,\mu) \in Steps(y)\ .\ \mu(x) > 0\}$
4: $\quad$ **for all** $y \in Y$ **do**
5: $\quad\quad Steps'(y) \Leftarrow \{(a,\mu) \in Steps(y) \mid \mu(x) > 0\}$
6: $\quad\quad p_y^{\max,k} \Leftarrow \max_{(a,\mu)\in Steps'(y)} \sum_{s'\in S} \mu(s') \cdot p'_{s'}$
7: $\quad$ **end for**
8: **end for**

be replaced by $\frac{p_x^{\max,k-1} - p_x^{\max,k-2}}{p_x^{\max,k-2}} \geq \delta$. Note that this condition is slightly different from the convergence criterion $|\frac{p_x^{\max,k-1} - p_x^{\max,k-2}}{p_x^{\max,k-1}}| < \delta$ used in PRISM. But the impact on the probability is negligible as the experiments in Section 5.1.4 show.

5.1.2 Acceleration on SCC-based Matrix Iteration

The experimental results in [26] have demonstrated the efficiency of SCC-based value iteration. We propose further improvements for it in this section.

Eliminating precomputations

The precomputation step presented in Section 5.1.1 can speed up value iteration in many cases, and, most importantly, it reduces the approximation error of the computation. However, it requires fixpoint computation over the set of states, and, therefore, is time-consuming. Here we show that this step can be eliminated for SCC-based value iteration. By Lemma 1, we can safely remove $Prob0A$ without losing accuracy.

Lemma 1 *A state in an SCC has maximum probability zero iff all states in the successor set $Succ(SCC)$ of SCC have maximum probability zero.*

Proof 1 *$\Leftarrow$. This is trivial.*

$\Rightarrow$. Suppose $s \in SCC$ has maximum probability zero and $\sigma \in Succ(SCC)$ has a non-zero maximum probability. There exists a state $s' \in SCC$ such that $\exists(a,\mu) \in Step(s')\ .\ \mu(\sigma) > 0$. Apparently, s' has non-zero maximum probability. By definition of SCC, there exists a path $\omega = s_1 \ldots s_n$ in SCC such that $s_1 = s$ and $s_n = s'$. We can deduce backwards the path that $p_{s_{n-1}}^{\max} > 0$, $p_{s_{n-2}}^{\max} > 0$, ..., $p_{s_1}^{\max} = p_s^{\max} > 0$. □

According to Lemma 1, we do not need to perform any special computation to enforce $prob0A$, as for such an SCC the iteration process terminates after the first iteration.

Lemma 2 *Given an SCC and its successor set $Succ(SCC)$, let $suc^0 = \{x \in Succ(SCC) \mid p_x^{\max} < 1.0\}$. If either*

1. *suc^0 is empty and $Succ(SCC)$ is not, or*
2. *suc^0 is non-empty and there does not exist a state $s \in SCC$ such that $\forall(a,\mu) \in Step(s)\ .\ \exists s' \in suc^0\ .\ \mu(s') > 0$,*

then all states in SCC have maximum probability one.

Proof 2 *First recall that there is no MEC nor self-loop in SCC. By removing every action from states in SCC such that it has a transition reaching suc^0 with probability greater than zero, we obtain a partition of SCC where each block forms a connected graph and no connection is among blocks. In each block $B \subseteq SCC$, each state only has transitions either leading to states in the same block or in $suc^1 = Succ(SCC) \backslash suc^0$. For all state states $s \in B$, if the maximum probability of reaching suc^1 is less than one, there exists an infinite path ω starting at s and only passing states in B. Let $inft(\omega)$ be the set of*

state-action pairs that occurs infinitely often in ω. Then according to [43, Theorem 3.2, page 46], $inft(\omega)$ is an end component, which contradicts to the premise.

□

By Lemma 2, $Prob1E$ can be removed by inserting a check before the first iteration of each SCC. This check is simpler than $Prob1E$ because it does not require double fixpoint computation. However, Lemma 2 only gives a sufficient condition to detect states with maximum probability one, which means that, in some cases, it is not possible to identify the whole set S^{yes}. However, for the experiments we performed, Lemma 2 returns the same set of states as $Prob1E$ does, and more importantly, runs much faster.

Remark. The precomputation for minimum probability can be eliminated by Lemma 3 and 4, which are the dual of Lemma 1 and 2 respectively.

Lemma 3 *A state in an SCC has minimum probability one iff all states in the successor set $Succ(SCC)$ of SCC, i.e., , have minimum probability one.*

Lemma 4 *Given an SCC and its successor set $Succ(SCC)$, let $suc^1 = \{x \in Succ(SCC) \mid p_x^{\max} > 0\}$. If either*

1. *suc^1 is empty, or*
2. *suc^1 is non-empty and there does not exist a state $s \in SCC$ such that $\forall (a, \mu) \in Step(s)\ .\ \exists s' \in suc^1\ .\ \mu(s') > 0$,*

then all states in SCC have minimum probability zero.

Parallel computation

As multi-core architectures become the mainstream of CPU design, it is very interesting to utilise the power of parallel computation to accelerate value iterations. The topological order among SCCs provides a natural structure for parallel computation. At any step, all SCCs whose successor set has been processed completely can be processed independently, and thus, in parallel. To achieve this, we need a queue to store SCCs that are ready to be processed. Initially, all SCCs that have an empty successor set are put in the queue. Each computation thread takes one SCC from the queue to process, and when it is done, it puts SCCs that newly become ready into the queue. The whole process terminates when the queue is empty. Let $\overline{Succ}(SCC)$ be a copy of the successor set $Succ(SCC)$ of an SCC in $\overline{S}_{SCC}$. Algorithm 4 shows the procedure for parallel computation. Note that in the **while** loop, only line 4 can be executed in parallel.

Algorithm 4: The Parallel Computation

```
1:  Queue ⇐ {SCC_i ∈ S̄_SCC | Succ̄(SCC_i) = ∅}; S̄_SCC ⇐ S̄_SCC \ Queue
2:  while Queue ≠ ∅ do
3:      scc ⇐ the head of Queue
4:      compute maximum probabilities for states in scc
5:      for all SCC ∈ S̄_SCC . Succ̄(SCC) ∩ scc ≠ ∅ do
6:          Succ̄(SCC) ⇐ Succ̄(SCC) \ scc
7:          if Succ̄(SCC) = ∅ then
8:              S̄_SCC ⇐ S̄_SCC \ {SCC}; Queue ⇐ Queue ∪ {SCC}
9:          end if
10:     end for
11: end while
```

In addition, finding all MECs can be done using multi-cores. The idea is that to partition a model into SCCs first, and secondly, searching for MECs in each SCC in parallel.

5.1.3 Incremental Value Iteration

Incremental verification is very useful when verification has to be repeated for a model which has undergone minor changes, as it allows to reuse the results from previous rounds of verification in a later round. Incremental verification is particularly vital for online monitoring or run-time analysis due to the speed requirement of the latter. For example, [22] proposed a self-adaption framework to monitor IT systems modelled as CTMCs/DTMCs, which takes a snapshot of a system regularly and automatically adjusts system parameters according to the quantitative verification result of each snapshot. In [22], the verification tasks for the snapshots were not performed incrementally, resulting in slow performance.

In this section, we target MDPs and explore the cases where some probability distributions in a model can change as time passes, while the transition structure of the model remains untouched. This means that transitions with probability one or zero cannot be changed; otherwise, some transitions with non-zero probability would be added or deleted from the model.

When some probabilities in $Steps$ are changed, it might be not necessary to recompute probability values for all states. Let $Steps'$ be the updated transition relation in the model. We first identify the set $\overline{S}'_{SCC}$ of SCCs that have been affected by the changes. It can be generated using Algorithm 5.

At the beginning, $\overline{S}'_{SCC}$ is initialised to an empty set. Then we scan the SCC partition in the reversed topological order and add SCC_i to $\overline{S}'_{SCC}$ if SCC_i satisfies one of two conditions:

1. There exists a state $s \in SCC_i$ that one action enabled in s is involved in the changes;
2. There exists an $SCC \in \overline{S}'_{SCC}$ that SCC_i depends on.

Algorithm 5: Generate $\overline{S}'_{SCC}$

1: $\overline{S}'_{SCC} \Leftarrow \emptyset$
2: **for all** $i \in 1, \ldots, m$ **do**
3: **if** $\exists s \in SCC_i \,.\, Steps(s) \neq Steps'(s)$ **or** $\exists SCC \in \overline{S}'_{SCC} \,.\, Succ(SCC_i) \cap SCC \neq \emptyset$ **then**
4: $\overline{S}'_{SCC} \Leftarrow \overline{S}'_{SCC} \cup \{SCC_i\}$
5: **end if**
6: **end for**

Let $p_s^{\max}$ be the probability computed for state s initially and $\overline{p}_s^{\max}$ the one we need to compute after the changes occur. The SCC-based value iteration can be adapted to handle changes in probabilities by replacing $\overline{S}_{SCC}$ by $\overline{S}'_{SCC}$ and initialising $\overline{p}_s^{\max}$ as follows:

$$\overline{p}_s^{\max,0} = \begin{cases} 1 & \text{if } s \in F \\ p_s^{\max} & \text{if } \exists SCC \in \overline{S}_{SCC} \backslash \overline{S}'_{SCC} \,.\, s \in SCC \\ 0 & \text{otherwise} \end{cases}$$

In addition, before we recompute the probability for an SCC in $\overline{S}'_{SCC}$, we perform a test on its successor set $Succ(SCC)$. This test checks the following conditions:

1. for every state $s \in Succ(SCC)$, its probability is not affected by the changes, i.e.,
$$\forall s \in Succ(SCC) \,.\, \overline{p}_s^{\max} = p_s^{\max}, \tag{5.1}$$
2. all actions enabled in a state in SCC are not affected by the changes, i.e.,
$$\forall s \in SCC \,.\, Steps'(s) = Steps(s).$$

If both conditions hold, there is no need to perform recomputation in this SCC, i.e.,

$$\forall s \in SCC \,.\, \overline{p}_s^{\max} = p_s^{\max}.$$

Though the above test can eliminate unnecessary recomputation for SCCs that might be affected by the changes, condition (5.1) is quite restrictive, as it requires all states in the successor set have the same probability as before the changes occurred. Recomputation is executed even if, for all states in $Succ(SCC)$, there are only tiny changes between $\overline{p}_s^{\max}$ and $p_s^{\max}$, e.g.,

$$\overline{p}_s^{\max} \neq p_s^{\max} \wedge |\overline{p}_s^{\max} - p_s^{\max}| < \epsilon.$$

In this case, the change of the probability for a state in SCC is bounded by ϵ with respect to its original value. If ϵ is less than the required accuracy, we can use $p_s^{\max}$ as $\overline{p}_s^{\max}$ for state s in SCC, which speeds up the recomputation by introducing a small approximation error. Lemma 5 formalises this idea.

Lemma 5 *1. If the condition $\overline{p}_s^{\max} = p_s^{\max}$ in condition (5.1) is replaced by $|\overline{p}_s^{\max} - p_s^{\max}| < \epsilon$ and the test succeeds, then*

$$\forall x \in SCC \,.\, |\overline{p}_x^{\max} - p_x^{\max}| < \epsilon. \tag{5.2}$$

2. If condition $\overline{p}_s^{\max} = p_s^{\max}$ is replaced by $|\frac{\overline{p}_s^{\max} - p_s^{\max}}{p_s^{\max}}| < \epsilon$ and the above test succeeds, then

$$\forall x \in SCC \,.\, |\frac{\overline{p}_x^{\max} - p_x^{\max}}{p_x^{\max}}| < \epsilon. \tag{5.3}$$

Proof 3 *Consider the base case where $s_0 \xrightarrow{a,p} s_1$ and $s_0 \xrightarrow{a,1-p} s_2$. Let v_0, v_1 and v_2 be the probability values in state s_0, s_1 and s_2 respectively. Note that $v = p \cdot v_1 + (1-p) \cdot v_2$. Let v_0', v_1' and v_2' be the new probability value for s_0, s_1 and s_2 respectively after some probabilities in the model are changed (but p is not changed). Here we discard the absolute value in formulae (5.2) and (5.3) and consider $v_1' \geq v_1$ and $v_2' \geq v_2$ only.*

Maximum absolute difference. *If $v_1' - v_1 < \epsilon$ and $v_2' - v_2 < \epsilon$, we have*

$$\begin{aligned} v_0' - v_0 &= \big(p \cdot v_1' + (1-p) \cdot v_2'\big) - \big(p \cdot v_1 + (1-p) \cdot v_2\big) \\ &= p \cdot (v_1' - v_1) + (1-p) \cdot (v_2' - v_2) \\ &< p \cdot \epsilon + (1-p) \cdot \epsilon \\ &= \epsilon. \end{aligned}$$

Maximum relative difference. *Assume $\frac{v_1' - v_1}{v_1} < \epsilon$ and $\frac{v_2' - v_2}{v_2} < \epsilon$. We rearrange the inequalities to obtain*

$$v_1' < (1+\epsilon) \cdot v_1 \text{ and } v_2' < (1+\epsilon) \cdot v_2.$$

Then we have

$$\begin{aligned} v_0' - v_0 &= \frac{\big(p \cdot v_1' + (1-p) \cdot v_2'\big) - \big(p \cdot v_1 + (1-p) \cdot v_2\big)}{p \cdot v_1 + (1-p) \cdot v_2} \\ &= \frac{p \cdot (v_1' - v_1) + (1-p) \cdot (v_2' - v_2)}{p \cdot v_1 + (1-p) \cdot v_2} \\ &< \frac{p \cdot \big((1+\epsilon) \cdot v_1 - v_1\big) + (1-p) \cdot \big((1+\epsilon) \cdot v_2 - v_2\big)}{p \cdot v_1 + (1-p) \cdot v_2} \\ &= \frac{p \cdot \epsilon \cdot v_1 + (1-p) \cdot \epsilon \cdot v_2}{p \cdot v_1 + (1-p) \cdot v_2} \\ &= \frac{\epsilon \cdot \big(p \cdot v_1 + (1-p) \cdot v_2\big)}{p \cdot v_1 + (1-p) \cdot v_2} \\ &= \epsilon. \end{aligned}$$

Note that if $s_1 = s$ or $s_2 = s$, the above reasoning still holds. For example, let $s_1 = s$. We have $v = p \cdot v + (1-p) \cdot v_2$, which is simplified to $v = v_2$. If $v_2' - v_2 < \epsilon$, then $v' - v < \epsilon$; If $\frac{v_2' - v_2}{v_2} < \epsilon$, then $\frac{v' - v}{v} < \epsilon$.

Lemma 5 is proved by generalisation of the base case. □

In practice, we can use δ, the maximum absolute difference or maximum relative difference, as ϵ, or a smaller value than δ to increase the accuracy with possible longer computation time.

5.1.4 Experiments

We have implemented the incremental verification technique in an extension of PRISM, using explicit-state data structures, and investigated performance for the coverage on the Terrorist Alert scenario[2]. In

[2] To apply the technique, we converted the CTMC model into an MDP model.

Table 5.1: Incremental Verification Results

N, M	States	Original	SCC-based			Incremental	
		Total time (s)	SCC comp time (s)	Sequential time (s)	Parallel time (s)	Sequential time (s)	Parallel time (s)
$2, 2$	8370	3.241	0.184	0.488	0.421	0.022	0.032
$2, 3$	24894	12.921	0.322	0.947	0.629	0.108	0.093
$2, 4$	74466	49.781	0.951	1.856	1.232	0.055	0.432
$2, 5$	223182	186.505	2.928	6.984	5.222	0.212	2.503
$2, 6$	669330	671.963	12.465	27.564	27.675	0.368	18.281

the experiment, we fixed the number N of commanders to be two, and choose different numbers M of other guards. We also set the time out deadline to be 10 and $R2$, the exponential distribution rate for the communication between the CONNECTor and guards, to be 0.1.

The experiment was performed on an AMD Phenom(tm) 9600B Quad-Core Processor with 8GB memory running Fedora 12 x86_64 Linux. Our experimental results are presented in two parts. The first covers the optimisations to accelerate SCC-based value iteration from Section 5.1.2; the second one focuses on the incremental quantitative verification techniques of Section 5.1.3.

Accelerating SCC-based value iteration.

In the first part, we compare running times of three verification approaches:

1. the original version of value iteration,
2. SCC-based value iteration, without precomputation
3. parallel SCC-based value iteration, without precomputation (using 4 threads, one per core).

The times are shown under the columns 'Original' and 'SCC-based' in Table 5.1. For the former, we report the total time spent on the verification. For the two SCC-based approaches, i.e. sequential and parallel, we give the time spent computing SCCs (including identification and compression of maximal end components), which is required for both approaches, as well as for SCC-based value iteration.

In this scenario, the SCC-based value iteration is much faster than the original one. The parallel version also shows improvement with respect to the sequential one: in the best case, the speed up is about 1.5. Although this is lower than the number of threads, numerical computations such as value iteration are known to be hard to parallelise so this remains a very encouraging result. There are several factors preventing further speed-ups for the parallel version. The major ones are: (1) at some point in the process, there are fewer independent SCCs than threads; (2) the synchronisation overhead is comparably heavy for SCCs that contain only one state; In addition, the implementation could be further tuned to alleviate memory contention among threads.

Incremental value iteration. To demonstrate the incremental verification algorithm without bias, we randomly choose three states that are not in any MEC, and have a distribution with probabilistic choices. For each state s, we pick such a distribution $\mu \in Steps(s)$ and modify the probability distribution as follows. Assume there exist m ($m > 1$) states $s_1, \ldots, s_m \in S$ such that $\mu(s_i) > 0$ for $1 \leq i \leq m$. The new distribution μ' in a is such that, for $1 \leq i \leq m-1$, we keep half of the value, i.e., $\mu'(s_i) = \mu(s)/2$; for $i = m$, we increase the value such that $\mu'(s_m) = \mu(s_m) + \sum_{i=1}^{m-1} \mu(s_i)/2$.

Times for the incremental value iteration algorithm, described in Section 5.1.3, are reported in the final two columns of Table 5.1. This includes both the sequential and parallel versions. We do not consider the time for SCC computation, since this does not need to repeated. Even when ignoring this, we see that the times for incremental value iteration represent significant speed-ups compared to the non-incremental (SCC-based) version: they always faster, up to 50 times faster in some cases. The sequential version works particularly well; for models where a small number of SCCs need to be updated, the gains for the parallel version are less impressive due to the synchronisation overhead.

5.2 Runtime Analysis via Monitoring

A key concern of CONNECT is to ensure CONNECTability, in spite of the continuous evolution of the Networked Systems, which modify their behaviour at run time to tackle the continuous changes in the unpredictable open-world settings [11]. In such partially unknown and evolving context, also dependability&performance analysis calls for on-line support to enhance the accuracy of preliminary estimates performed at design time.

The DePer presented in the previous chapter (see Section 4.1.3) is in charge of performing model-based dependability and performance analysis to assess, before deployment, if the dependability and performance requirements requested by the NSs can be satisfied by the synthesised CONNECTor. In this section we present a preliminary study on how DePer can interact with the CONNECT Monitoring Enabler to dynamically refine its assessment at runtime, after a CONNECTor is deployed and running.

5.2.1 Monitoring Enabler

The Monitoring Enabler is in charge of performing complex event recognition, as well as observing and notifying specific events occurrences.

The architecture of the CONNECT Monitoring Enabler, called GLIMPSE[3], has been presented in Deliverable D4.2 [34]. To make presentation self-contained, we outline a brief summary of its components and report its illustration in Figure 5.1. GLIMPSE has been conceived with a highly flexible architecture [15] decoupling high-level event specification from the underlying observation and analysis mechanisms, so to yield the greatest generality and facility of use.

The Monitoring Enabler is driven by the Manager component (see Figure 5.1). The collection of raw data coming on the Monitoring Bus from the observed components is provided by Probes. The essential characteristic of this model is that producers (Probes) do not need to know any information about who are the event consumers (Dependability & Performance Enabler for instance). They send information about their own state only, precluding any assumptions on consumer functionality. The information sent by Probes is structured on a `ConnectBaseEvent` object whose interface is shown in Figure 5.2. Probes may be realized either by injecting code into an existing software, or by using proxies that intercept events to be analysed and send them to the Monitoring Bus. The communication backbone of the Monitoring Enabler is the Monitoring Bus, to which all the information (events, requests, responses) are sent on by: Probes, CONNECT Enablers, Complex Event Processor (CEP) and by any other services joining GLIMPSE.

The Complex Event Processor, instructed with the information provided by the Dependability & Performance Enabler, is implemented using JBoss Drools Fusion [2], a rule engine based on Charles Forgy's Rete Algorithm [49], able to perform the analysis of the events streaming on the Monitoring Bus. All the messages exchanged between units are JMS Messages. The latter is implemented using ServiceMix4 [3] and ActiveMQ [1] technologies.

5.2.2 Enablers Interactions

The event-based mode of interaction is also known as implicit invocation. The interactions between Dependability & Performance and Monitoring Enablers start after the Dependability & Performance Enabler determines that the synthesised CONNECTor satisfies the required dependability level. Specifically, after the analysis phase, Dependability & Performance Enabler informs the Monitoring Enabler on which are the parameters (among those used in the dependability analysis) relative to CONNECTor and NSs, that must be kept under observation at run-time. The Monitoring Enabler, upon receiving the request, properly instructs the probes embedded in the CONNECTor.

Once the CONNECTor is deployed, the data derived from evaluation of real executions are sent from Monitoring Enabler to Dependability & Performance Enabler. In this way, DePer can perform statistical analyses based on the real data coming from the monitored observations and can use such information to continuously check the accuracy of the analysed model. If the model parameters are found to be inaccurate, Dependability & Performance updates the model with the new values, and performs a new

[3] Generic fLexIble Monitoring based on a Publish-Subscribe infrastructurE, http://labse.isti.cnr.it/tools/glimpse

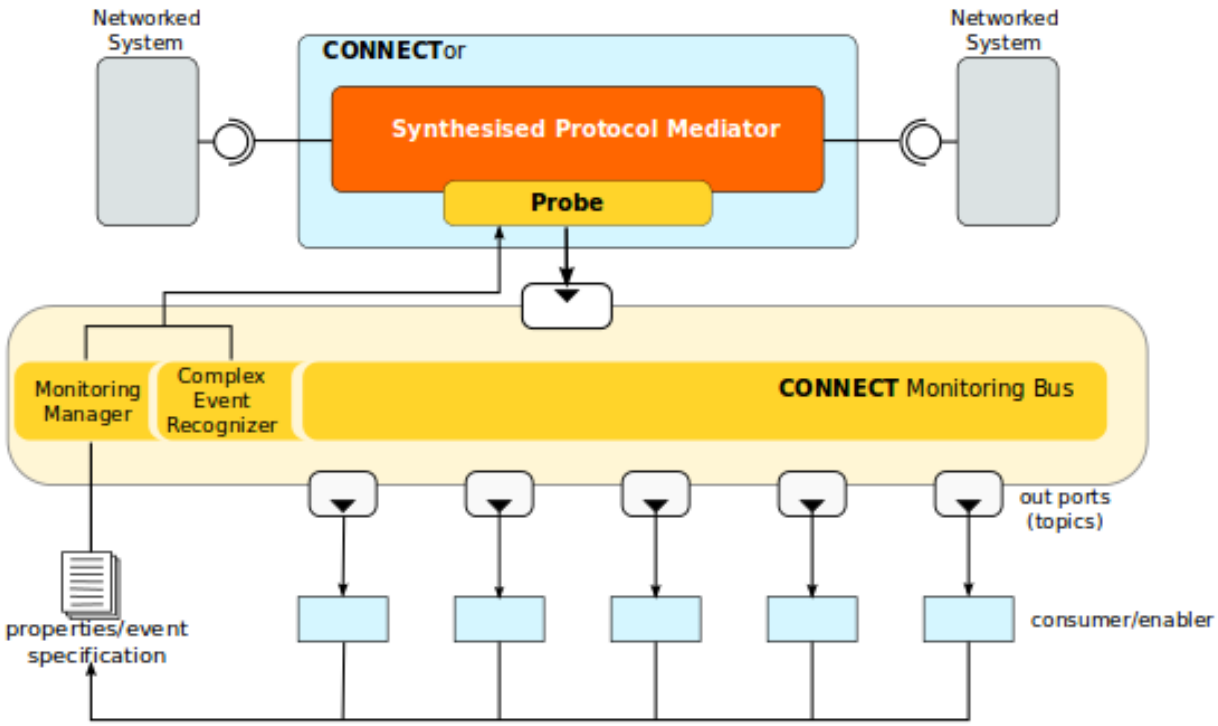


Figure 5.1: Monitoring Enabler Architecture

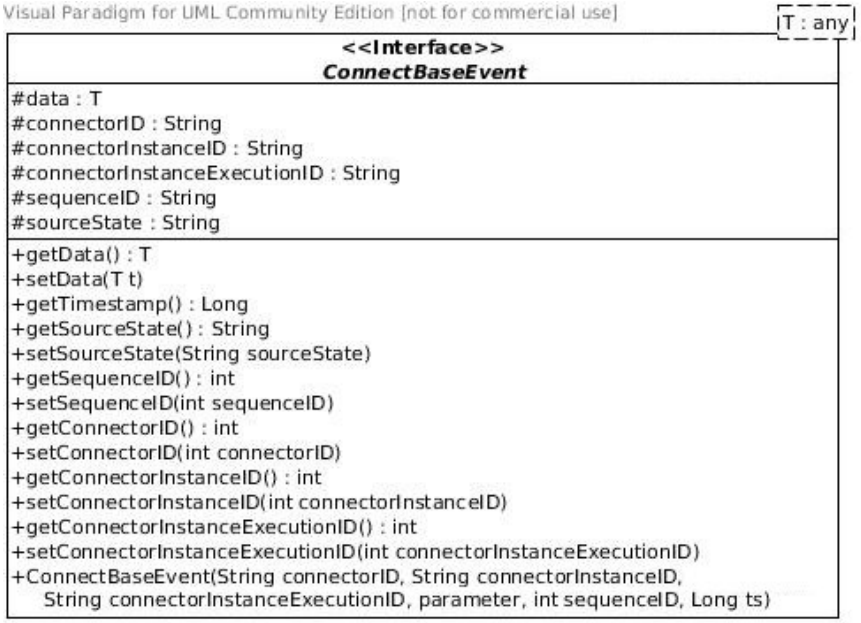


Figure 5.2: CONNECT Base Event Interface

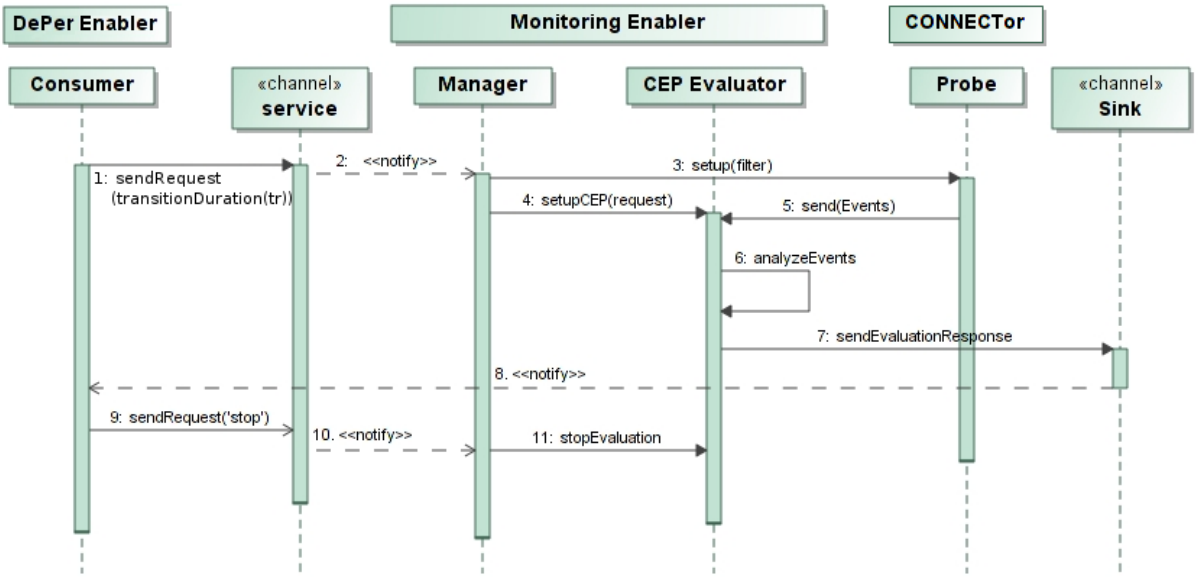


Figure 5.3: Sequence Diagram of the Basic Interaction Pattern between DePer and Monitoring

analysis. If the new analysis evidences that the deployed CONNECTor needs adjustments, in cooperation with the Synthesis Enabler a new synthesis–analysis cycle is started.

With more details, Dependability & Performance Enabler and Monitoring Enabler interact by using a Publish/Subscribe protocol. The interaction pattern is shown as a sequence diagram in Figure 5.3 where we intentionally left out system start-up operations (for a more detailed sequence diagram, see Deliverable D4.2 [34]). Whenever Monitoring Enabler receives a request message on the service channel (see message 2 on Figure 5.3), a new channel dedicated to the requesting Enabler is set up to communicate the monitored values.

GLIMPSE sends response messages to Dependability & Performance Enabler as soon as the aspect of interest is available (see message 8 on Figure 5.3).

The two Enablers exchange JMS messages whose payload is expressed in XML language. The payload of the XML, contains a `ComplexEventRuleActionList` xml object, which determines a lists of possible actions to execute on the Monitoring Enabler knowledge base. The schema of ComplexEventRuleActionList is shown in Listing 5.1

The Complex EventRule ActionList specification supports the use of heterogeneous rule languages, as it is natively unbound to any. In our example, into the field RuleBody, the Dependability & Performance Enabler, will set the Drools rule for the requested evaluation. In a basic interaction between Monitoring and Dependability & Performance Enabler, another Enabler is involved, the Deployment Enabler. The information about the CONNECTor: identity, instance, execution, are stored into the Deployment Enabler that can be prompted by the Monitoring Enabler to retrieve such information.

5.2.3 Application Example

In the following, we focus on the Enablers interactions only, leaving out of scope the actions taken by Dependability & Performance Enabler once it obtains the values observed at runtime from the Monitoring Enabler. To show a basic interaction between Dependability & Performance Enabler and Monitoring Enablers, we refer to the Terrorist Alert Scenario [35]. As summarised in Section 2.1, the scenario considers the interactions between police and patrolling civil guards. It is assumed that policemen and guards are both equipped with mobile devices, but they use different communication protocols. Hence, we intend to use the CONNECT infrastructure to enable the direct interoperation between a policeman and guards in the zone.

Following CONNECT work on synthesis of mediating CONNECTors [94] and automata discovery/learning [89], the specification of the CONNECTed system is given with Labelled Transition Systems (LTSs) [63].

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://labse.
      isti.cnr.it/glimpse/xml/ComplexEventRule" xmlns:tns="http://labse.isti.cnr.it/
      glimpse/xml/ComplexEventRule" elementFormDefault="qualified">
3
4
5     <element name="ComplexEventRuleActionList" type="tns:
          ComplexEventRuleActionType"></element>
6
7     <complexType name="ComplexEventRuleActionType">
8       <sequence>
9         <element name="Insert" type="tns:ComplexEventRuleType"
10          maxOccurs="unbounded" minOccurs="0">
11        </element>
12        <element name="Delete" type="tns:ComplexEventRuleType"
13          maxOccurs="unbounded" minOccurs="0">
14        </element>
15        <element name="Start" type="tns:ComplexEventRuleType"
16          maxOccurs="unbounded" minOccurs="0">
17        </element>
18        <element name="Stop" type="tns:ComplexEventRuleType"
19          maxOccurs="unbounded" minOccurs="0">
20        </element>
21        <element name="Restart" type="tns:ComplexEventRuleType"
22          maxOccurs="unbounded" minOccurs="0">
23        </element>
24      </sequence>
25    </complexType>
26
27    <complexType name="ComplexEventRuleType">
28      <sequence>
29        <element name="RuleName" type="string" maxOccurs="1" minOccurs="1"></
              element>
30        <element name="RuleBody" type="string" maxOccurs="1" minOccurs="0"></
              element>
31      </sequence>
32      <attribute name="RuleType" type="string"></attribute>
33    </complexType>
34 </schema>
```

Listing 5.1: ComplexEventRuleActionList Schema

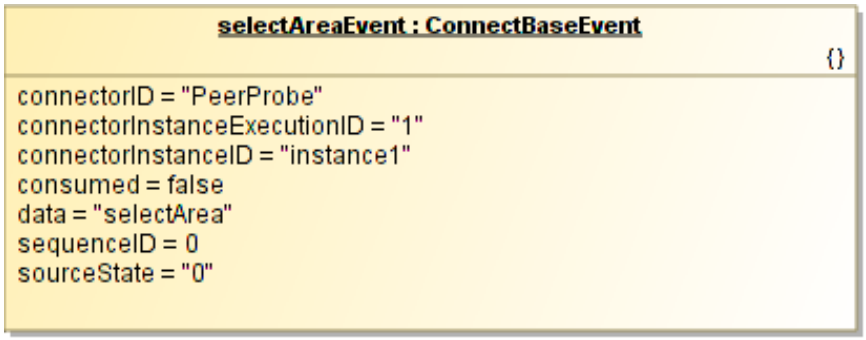


Figure 5.4: The selectArea Event Sent from Peer Probe

What we want to monitor at runtime is the latency between two states of the LTS. Specifically, we want to monitor latency of two transitions from the LTS shown in Figure 2.3 (Chapter 2).

The parameters under monitoring are the duration of the transitions executed by the NS requesting the communication, on which time-outs have been setup in the CONNECTor specification to limit the waiting periods. Therefore, having feedbacks on real executions is useful to improve the time-out calibration.

From Figure 2.3, the events to be monitored are the consumer transitions `selectArea` and `areaSelected`. The request messages sent by Dependability & Performance Enabler to Monitoring are shown in Listing 5.2.

The Monitoring infrastructure, more specifically, the Manager component, receives the Dependability & Performance requests and sets up the ComplexEventProcessor with the provided rule.

The events flowing in from Probes are structured on a `ConnectBaseEvent` object (see Figure 5.2), that provides all the necessary informations for an accurate pattern recognition.

According to the scenario, the peer that initiates the communication sends a broadcast message `selectArea` to selected peers (the Police control center or policemen) operating in a specified area of interest.

The event generated from the Probe instrumented into the peer software component is shown in Figure 5.4 and flows in into the Monitoring infrastructure stream of events.

When the selected peer replies (Police control center or policemen), another event is fired and sent on the Monitoring Bus, the `areaSelected` event.

Using the timestamp contained into the two different events, checking and matching connectorID, sequenceID, ConnectorInstanceID, ConnectorInstanceExecutionID between the two events, the CEP is able to calculate the latency between the two events and provide it to the Dependability & Performance Enabler.

With those results, the Dependability & Performance Enabler, is able now to evaluate the behaviour of the CONNECTor and if this is not compliant to the expected values, it may contact the Syntesis Enabler requiring a new synthesis process.

5.3 Conclusions and Further Research Directions

In this chapter, we have presented two approaches for evolving dependability analysis at runtime. They include first a technique for optimising stochastic verification of the MDP model for CONNECTed systems, based on a decomposition into strongly CONNECTed components. This technique reduces the amount of graph-based computation required and provide opportunities for parallelisation. In particular, we also focused on the applicability of this to incremental verification: re-analysing an MDP after small changes in its probability values, by re-using existing verification results. In the future, we plan to develop these techniques further, for example, considering also the case where the structure of the model changes.

Second, this chapter has also shown work in-progress on the synergic use of stochastic model-based dependability & performance analysis, conducted at system design-time, with run-time monitoring to improve the accuracy of property estimates. The basic interplay between Dependability & Performance and Monitoring Enablers has been discussed, highlighting the benefits of the monitoring feedbacks on the

```
<?xml version="1.0" encoding="UTF-8"?>
<ComplexEventRuleActionList xmlns="http://labse.isti.cnr.it/glimpse/xml/
    ComplexEventRule"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://labse.isti.cnr.it/glimpse/xml/ComplexEventRule ./
     ComplexEventRule.xsd">
    <Insert RuleType="drools">
        <RuleName>transitionDurationRule</RuleName>
        <RuleBody>
  import it.cnr.isti.labse.glimpse.event.SimpleEvent;
  import it.cnr.isti.labse.glimpse.manager.ResponseDispatcher;
  import it.cnr.isti.labse.glimpse.settings.DroolsUtils;

  rule "transitionDuration"
  no-loop
  salience 999
  dialect "java"
  when
    $aEvent : SimpleEvent(this.data == "selectArea",
          this.getConsumed == false);

    $bEvent : SimpleEvent(this.data == "areaSelected",
          this.getConsumed == false,
          this.getConnectorID == $aEvent.getConnectorID,
          this.getConnectorInstanceID == $aEvent.getConnectorInstanceID,
          this.getConnectorInstanceExecutionID == $aEvent.
              getConnectorInstanceExecutionID,
          this after $aEvent);
  then
    $aEvent.setConsumed(true);
    $bEvent.setConsumed(true);
    update( $aEvent );
    update( $bEvent );
    retract( $aEvent );
    retract( $bEvent );
     ResponseDispatcher.NotifyMe(drools.getRule().getName(),"
         DependabilityAnalisysEnabler1",DroolsUtils.latency($aEvent.getTimestamp(),
          $bEvent.getTimestamp()));
    end
        </RuleBody>
    </Insert>
</ComplexEventRuleActionList>
```

Listing 5.2: Sample Request from Dependability & Performance Enabler

model-based analysis. A simple application example has been also presented, which shows the operative steps of the Enablers interactions. This activity is also documented in [14]. As already mentioned, this is work in progress and further developments are needed, including: (i) to finalise the implementation of statistical analysis on observed data, (ii) to set-up an appropriate data model to store the monitored data, (iii) to possibly extend monitoring to the analysed dependability and performance properties, to validate the correctness of the specification of CONNECTor and NSs used by the Dependability & Performance Enabler, and (iv) to integrate the CPMM model presented in Chapter 3 with the Monitor infrastructure creating a self-contained suite that will provide metrics rule templates for all the Enablers that need to evaluate non-functional properties at runtime.

6 Trust Management Interoperability

The general CONNECT Trust model introduced in Deliverable D5.1 [32] highlighted three categories of trust relations, for assessing the trustworthiness of: CONNECTors, Enablers, and Networked Systems, respectively.

However, as a first step to supporting trust management in the CONNECT architecture, we assume that CONNECTors and Enablers are trusted and thereby focus on interoperable trust management among heterogeneous Networked Systems through the CONNECT Trust Enabler. Specifically, we define a trust meta-model that allows the rigorous specification of trust models as well as their composition. The resulting composite trust models enable heterogeneous trust management systems to interoperate transparently through mediators.

In effect, with people getting increasingly connected virtually, trust management is becoming a central element of today's open distributed digital environment. However, existing trust management systems are customized according to specific application domains, hence implementing different trust models. As a result, it is nearly impossible to exploit established trust relations across systems. However, while a trust relation holding in one system does not systematically translate into a similar relation in another system, it is still a valuable knowledge, especially if the systems relate to the same application domains (e.g., e-commerce, social network). This is such an issue that we are addressing in our work.

To the best of our knowledge, little work investigates interoperability between heterogeneous trust models. The closest to our concern is the work of [95], which describes a trust management architecture that enables dealing with a variety of trust metrics and mapping between them. However, the architecture deals with the composition at the level of trust values and do not account for the variety of trust models. In particular, one may want to differentiate between direct trust values and reputation-based ones when composing them. In general, what is needed is a way to formalize heterogeneous trust models and their composition. Such a concern is in particular addressed in [61, 98], which introduce trust meta-models based on state of the art trust management systems. Nevertheless, little detail is given and the paper does not describe how to exploit the meta-model for composing heterogeneous trust models and thereby achieving interoperability. Dealing with the heterogeneity of trust models is also investigated in [51, 96]. However, the study is for the sake of comparison and further concentrates on reputation-based models. Summarizing, while the literature is increasingly rich of trust models, dealing with their composition remains a challenge.

This chapter specifically introduces the foundations of the CONNECT Trust Enabler. We recall the key entities associated with trust management systems (Section 6.1). Then,w we formally define a trust meta-model that enables the specification of various trust models (Section 6.2). We further describe different mapping mechanisms and associated algorithms in order to support the composition of heterogeneous trust models and thus enable a common and systematic trust assessment across models (Section 6.3).

Next step is to integrate the proposed trust modeling and associated model composition in the CONNECT architecture, through the concrete development of the Security and Trust (SxT) Enabler and its integration with the introduced CONNECT Property Meta-Model (see Chapter 3).

6.1 Trust Model Definition

As defined in [52]: *A trustor trusts a trustee with regard to its ability to perform a specific action or to provide a specific service*. Hence, any trust model may basically be defined in terms of the three following elements:

1. *Trust roles* abstract the representative behaviors of stakeholders from the standpoint of trust management, in a way similar to role-based access control model [48].
2. *Trust relations* serve specifying trust relationships holding among stakeholders, and
3. *Trust assessment* define how to compute the trustworthiness of stakeholders.

We further define trust relations and assessment below.

6.1.1 Trust relations

We identify two types of trust relationships, i.e., *direct* and *indirect*, depending on the number of stakeholders that are involved to build the trust relationship (see Figure 6.1).

Direct trust: A direct trust relationship represents a trust assertion of a subject (i.e., trustor) about another subject (i.e., trustee). It is thus a *one-to-one trust relation* (denoted *1:1*) since it defines a direct link from *1* trustor to *1* trustee. One-to-one trust relations are maintained locally by trustors and represent the trustors' personal opinion regarding their trustees [62]. For example, a one-to-one relation may represent a belonging relationship (e.g., employees trust their company), a social relationship (e.g., trust among friends), or a profit-driven relationship (e.g., a person trusts a trader for managing its portfolio).

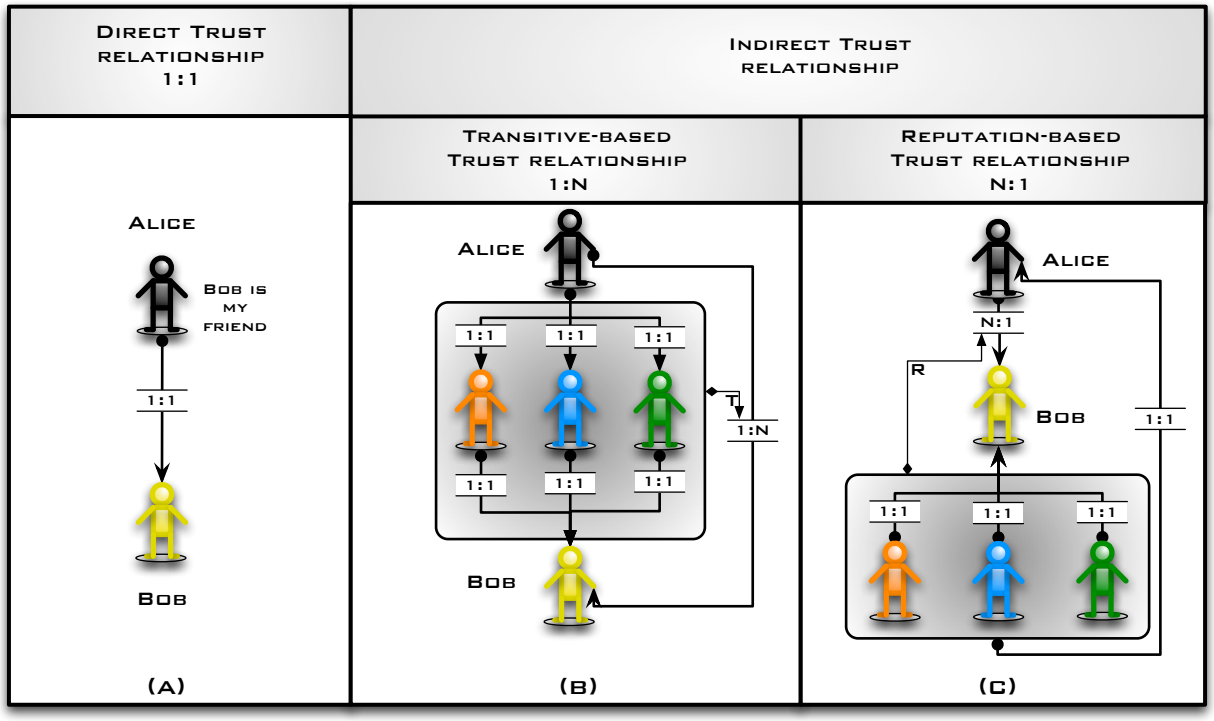


Figure 6.1: Trust Relations

Indirect trust: As opposed to a direct trust relationship, an indirect trust relationship represents a subject's trustworthiness based on a third party's recommendation(s). This can be either (i) transitive-based or (ii) reputation-based.

Transitive-based trust relations are *one-to-many* (denoted *1:N*). Such a relation enables *1* trustor (e.g., Alice in Figure 6.1(B)) to indirectly assess the trustworthiness of an unknown trustee (e.g., Bob in Figure 6.1(B)) through the *recommendations* of a group of trustees (*N*). Hence, the computation of *1:N* relations results from the concatenation and/or aggregation of many *1:1* trust relations (see arrow T in Figure 6.1). The concatenation of *1:1* trust relations usually represents a transitive trust path, where each entity can trust unknown entities based on the recommendation of its trustees. Thus, this relationship is built by composing personal trust relations [5, 92]. Furthermore, in the case where there exist several trust paths that link the trustor to the recommended trustee, the aggregation can be used to aggregate all given trust recommendations [59].

Reputation-based trust relations are *many-to-one* (denoted *N:1*). These relations result from the aggregation of many personal trust relationships of recommenders having the same trustee (see arrow R

in Figure 6.1). In other words, the group of recommenders (N) trust a specific subject (e.g., Alice in Figure 6.1(C)) to take their personal opinions and to maintain and provide the *reputation* of trustees (e.g., Bob in Figure 6.1(C)). In the literature, reputation systems are divided into two categories depending on whether they are (i) *Centralized* or (ii) *Distributed*. With the former, the reputation of each participant is collected and made publicly available at a centralized server (e.g., eBay, Amazon, Google, [80]). With the latter, reputation is spread throughout the network and each networked entity is responsible to manage the reputation of other entities (e.g., [59, 101]).

Figure 6.2 illustrates roles and relations of a representative trust model associated with the Terrorist Alert use case. Precisely, we introduce the specification of a centralized (Guard trust model, see Figure 6.2(A)) and fully distributed (Police trust model, see Figure 6.2(B)) trust models. Trust models aim at assessing *Policeman*/*Guard* security accreditation/reputation.

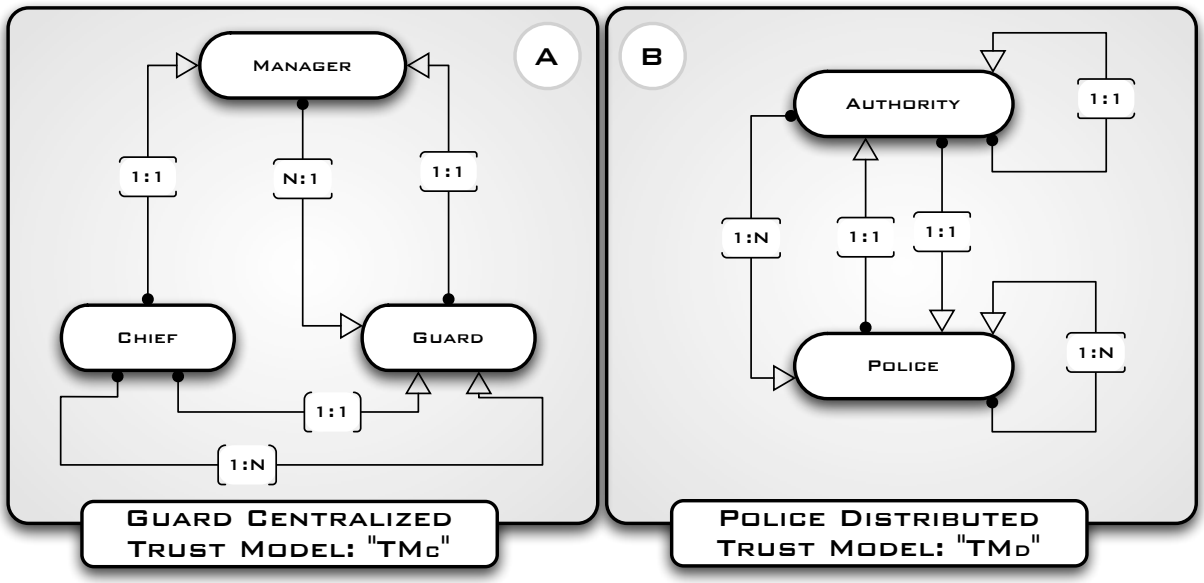


Figure 6.2: Terrorist Alert Trust Models TM_C and TM_D.

The centralized trust model (see Figure 6.2(A)) shows three roles, namely: the *Guard*, the *Chief*, and the *Manager*. The *Chief* and the *Guard* trust (i.e., *1:1* trust relation between a *Chief*/*Guard* and a *Manager*) the *Manager* to maintain the *Guards*' reputation (i.e., *N:1* trust relation between a *Manager* and a *Guard*). This *Guard* reputation is updated according to personal opinion of *Chiefs* (i.e., *1:1* trust relation between a *Chief* and a *Guard*). Hence, any *Chief* can assess the trustworthiness of a given *Guard* transitively through the *Manager* (i.e., *1:N* trust relation between a *Chief* and a *Guard*).

Regarding the distributed model, *Policemen* have a direct trust relationship with *Authorities*. *Authorities* also can collaborate and trust each other directly (*1:1*) or transitively (*1:N*). *Authorities* trust their *Policemen*, which can be represented by a direct relationship, or transitively if they pertain to a trusted *Authority* (*1:N*). In this way Policemen can check the level of accreditation of each other through their *Authority* (*1:N*).

6.1.2 Trust Management

Trust management, i.e., assigning values to trust relationships, relies on the definition of: (i) trust metrics characterizing how trust is measured and (ii) operations for assessing and composing trust relations.

Trust metrics: Different metrics have been defined to measure trust. This is due to the fact that one trust metric may be more or less suitable to a certain context. Thus, there is no widely recognized way to assign trust values. Some systems assume only binary values. In [102], trust is quantified by qualitative

labels (e.g., high trust, low trust etc.). Other solutions represent trust by a numerical range. For instance, this range can be defined by the interval [-1..1] (e.g., [74]), [0..n] (e.g., [5, 92]) or [0..1](e.g., [59]. A trust value can also be described in many dimensions, such as: (Belief, Disbelief, Uncertainty) [59].

In addition, several definitions exist about the semantics of trust metrics. This is for instance illustrated by the meaning of zero and negative values. For example, zero may indicate lack of trust (but not distrust), lack of information, or deep distrust. Negative values, if allowed, usually indicate distrust, but there is a doubt whether distrust is simply trust with a negative sign, or a phenomenon of its own.

Assessment operations: We define four main operations for the computation of trust values associated with the trust relations given in Section 6.1.1 (see table 6.1): *Setting*, *aggregation*, and *concatenation*.

	Setting		Aggregation	Concatenation
	Bootstrapping	Updating		
One-to-One (*1:1*)	X	X		
One-to-Many (*1:N*)		X	X	X
Many-to-One (*N:1*)	X	X	X	

Table 6.1: Trust Assessment Operations

The *setting* operations are mainly performed by trustors to $bootstrap$ (i.e., initialize) *1:1* and *N:1* trust relationships or to *update* these relationships after receiving personal feedback or third parties recommendation. The *bootstrapping* operation initializes the *a priori* values of *1:1* and *N:1* trust relations. Trust bootstrapping consists of deciding how to initialize trust relations in order to efficiently start the system and also allow newcomers to join the running system [88]. Most existing solutions simply initialize trust relation with a fixed value (e.g., 0.5 [55], a uniform Beta probabilistic distribution [60], etc.). Other approaches include among others: initializing existing trust relations according to given peers recommendations [90]; applying a sorting mechanism instead of assigning fixed values [92]; and assessing trustees into different contexts (e.g., fixing a car, babysitting, etc.) and then inferring unknown trust values from known ones of similar or correlate contexts [88, 6].

All the solutions dealing with *1:N* trust assessment mainly define the *concatenation* and the *aggregation* operations, in order to concatenate and to aggregate trust recommendations by computing the average [92], the minimum or the product [5] of all the intermediary trust values. In the case of Web service composition, some approaches (e.g., [84]) evaluate the recommendation for each service by evaluating its provider, whereas other approaches (e.g., [65]) evaluate the service itself in terms of its previous invocations, performance, reliability, etc. Then, trust is composed and/or aggregated according to the service composition flow (sequence, concurrent, conditional and loop). *Aggregation* operations such as Bayesian probability (e.g., [79]) are often used for the assessment of *N:1* (reputation-based) trust relations. Trust values are then represented by a beta Probability Density Function [60], which takes binary ratings as inputs (i.e., positive or negative) from all trustors. Thus, the reputation score is refreshed from the previous reputation score and the new rating [80]. The advantage of Bayesian systems is that they provide a theoretically sound basis for computing reputation scores and can also be used to predict future behavior.

6.2 Trust Meta-Model

Following the above, we formally define the trust meta-model as: $TM =< \mathbb{R}, \mathbb{L}, \mathbb{M}, \mathbb{O} >$, where $\mathbb{R}$, $\mathbb{L}$, $\mathbb{M}$ and $\mathbb{O}$ are the finite sets of trust roles, relations, metrics and operations, respectively.

In the following, we use the notations:

- Id to characterize an identifier which can be a string or be specified by an ontological concept although the latter is not longer discussed in this chapter.
- $\Diamond V$ to characterize a specific value from the the set of values V.
- $\veebar V$ to characterize an exclusive disjunction of values (only one of the values) from the set of values V, for instance, $v_1 \veebar v_2 \veebar v_3$ where $v_1, v_2, v_3 \in V$.

- $\wedge V$ to characterize a conjunction of: (i) values or (ii) an exclusive disjunction of values (one or more elements) from the set of values V, for instance, $v_1 \wedge v_2 \wedge (v_3 \veebar v_4)$ where $v_1, v_2, v_3, v_4 \in V$.
- $e \sqsubset v$ returns true if e is in v, for instance: having $v = v_1 \wedge v_2 \wedge (v_3 \veebar v_4)$, we can say that $v_1 \sqsubset v = true$.
- If a value v has the type $\wedge V$ (i.e., $v :: \wedge V$), we note v[i] to return the i^{th} element of v and $|v|$ to get the number of elements of v. For instance, having $v = v_1 \wedge v_2 \wedge (v_3 \veebar v_4)$, we can say that $|v| = 3$, $v[1] = v_2$, and $v[|v|] = v_3 \veebar v_4$.

Role set $\mathbb{R}$: The role set defines all the roles r played by the stakeholders of the trust model. A role r of $\mathbb{R}$ is simply denoted by its name:

$$r =< \text{name}::Id > \tag{6.1}$$

In our meta-model, a stakeholder is represented as a *Subject* s, playing a number of roles, r_1, r_2...and r_n, which is denoted as $s \blacktriangleright r_1, r_2 ... r_n$.

The following tables define of the roles of the Guard and the Police trust models.

Guard Trust Model		
Role set $\mathbb{R}$		
r_C	=	<name="*Chief*">
r_G	=	<name="*Guard*">
r_M	=	<name="*Manager*">

Police Trust Model		
Role set $\mathbb{R}$		
r_P	=	<name="*Policeman*">
r_A	=	<name="*Authority*">

Metric set $\mathbb{M}$: The metric set describes all the trust metrics that can be manipulated by the trust model. A metric is formally denoted as a pair:

$$m =< \text{name}::Id, \text{type}::Id > \tag{6.2}$$

where: *name* and *type* respectively define the name and the type (e.g., probability, rate, etc.) of the metric.

Given the example depicted in Figure 6.2, in the Guard trust model, we can identify three metrics, as follows:

Guard Trust Model		
Metric set $\mathbb{M}$		
m_{Rep}	=	<name="Reputation", type="Probability">
m_{Rec}	=	<name="Recommendation", type="Probability">
m_{Rat}	=	<name="Rate", type= "Labels">

The metrics m_{rep} and m_{rec} are defined to describe the trust value of a given reputation and recommendation respectively, both metrics define a probability value that falls in the interval $[0..1]$. Whereas, m_{rat} defines the trust feedback given by a *Chief* about the performance of the *Guards* under his command.

Concerning the police trust model, we identify two metrics that are defined through trust level (see Table below). These metrics (m_{acc} and m_{aut}), respectively, describe the level of accreditation of the *Policeman* and the level of trust of an *Authority*.

Police Trust Model		
Metric set $\mathbb{M}$		
m_{Acc}	=	<name="Accreditation", type="Levels">
m_{Aut}	=	<name="*Authority*", type="Levels">

Relation set $\mathbb{L}$: A relation set $\mathbb{L}$ contains all the trust relations that are specified by the trust model. We specifically denote a trust relation as a tuple:

$$\begin{aligned} l =< \text{name}::Id, \text{ctx}::Id, \text{type}::\Diamond Tl, \text{trustor}::\veebar\mathbb{R}, \text{trustee}::\veebar\mathbb{R}, \text{metric}::\Diamond\mathbb{M} > \\ \text{with } Tl = \{\text{"}\textit{1:1}\text{"}, \text{"}\textit{1:N}\text{"}, \text{"}\textit{N:1}\text{"}\} \end{aligned} \tag{6.3}$$

where:

- *name* identifies the relation;
- *ctx* describes the context of the relationship in terms of the application domain (e.g., security);
- *type* represents the type of the relation as given by Tl;
- *trustor* and *trustee* are roles where a trust relation relates a *trustor* role with a *trustee* role. Different trustors can establish the same type of relationship with different trustees. Thus, as a trust relation is binary and between a *trustor* role and a *trustee*, the exclusive combination of roles (e.g., $r_1 \veebar r_2 \veebar r_3$) is used to describe these elements.
- *metric* is an element from the metric set and thus reflects the trust measure given by the trustor to the trustee through this relation.

According to the above, we set the following definitions of the relations of the Guard and the Police trust models (see Figure 6.2).

Guard Trust Model		
Relation set $\mathbb{L}$		
l_{dCM}	=	$<$ name="ServerRecommendation", ctx= "Security", type="*1:1*", trustor="$(r_C \veebar r_G)$", trustee="r_M", metric="m_{rec}" $>$
l_{tCG}	=	$<$ name= "GuardTrustworthiness", ctx= "Security", type="*1:N*", trustor="r_C", trustee="r_G", metric="m_{Rec}" $>$
l_{dCG}	=	$<$ name="ChiefFeedback", ctx= "Security", type=*1:1*, trustor=r_C, trustee=r_G, metric=m_{Rat} $>$
l_{rMG}	=	$<$ name="GuardReputation", ctx= "Security", type=*N:1*, trustor=r_G, trustee=r_M, metric=m_{Rep} $>$

Police Trust Model		
Relation set $\mathbb{L}$		
l_{dAA}	=	$<$ name="TrustAuthority", ctx= "Security", type="*1:1*", trustor="r_A, trustee=r_A", metric="m_{Aut}" $>$
l_{dPA}	=	$<$ name="PoliceAuthority", ctx= "Security", type="*1:1*", trustor="r_P", trustee=r_A, metric="m_{Aut}" $>$
l_{dAP}	=	$<$ name="PoliceMember", ctx= "Security", type="*1:1*", trustor="r_A", trustee="r_P", metric="m_{Acc}" $>$
l_{tPP}	=	$<$ name="Colleague", ctx= "Security", type="*1:N*", trustor="r_P", trustee="r_P", metric="m_{Acc}" $>$
l_{tAP}	=	$<$ name="RemoteTrustPolice", ctx= "Security", type="*1:N*", trustor="r_A", trustee="r_P", metric="m_{Acc}" $>$

Operation set $\mathbb{O}$: The operation set specifies the operations that can be performed over relations by a subject to assess the trustworthiness of another subject. As defined in Section 6.1, trust assessment relies on the setting, the aggregation, and the concatenation operations.

An assessment operation is formally denoted as:

$$o =< \text{name::}Id, \text{host::}\veebar\mathbb{R}, \text{type::}\Diamond To, \text{input::}\wedge\mathbb{L}, \text{output::}\Diamond\mathbb{L}, \text{call::}\wedge\mathbb{O} > \quad \text{with } To = \{"setting", "aggregation", "concatenation"\} \tag{6.4}$$

where: (i) *name* identifies uniquely an operation;

- *host* specifies the role(s) that executes the operation;
- *type* defines the type of operation denoted by an element from the set To;
- *input* gives the trust relations that are required to perform an assessment operation;
- *output* gives the resulting trust relation;
- *call* denotes a continuation, i.e., a trust operation possibly triggers (an)other operation(s).

We define certain constraints on the operations as follows:

- A host executes an operation in order to assess its managed relations and only those ones (i.e., the relation where the host is a trustor), formally: $(\forall o \in \mathbb{O})\ o.host \sqsubset o.output.trustor$.
- The concatenation operation is defined to represent the assessment of a path of relations, namely, (i) each trustor of the output relation has to be a trustor of the first input relation, (ii) each trustee of an input relation has to be present as a trustor in a next input relation, and finally, (iii) each trustee from the output relation has to be present as a trustee of the last input relation, formally:

$$(\forall o \in \mathbb{O})\ o.type = concatenation \Rightarrow \begin{cases} (i) & (\forall l \sqsubset o.input[1])\ l.output.trustor \sqsubset l.input.trustor \\ (ii) & (\forall i \in [1..|o.input|[])(\forall l_x \sqsubset o.input[i])... \\ & \quad ...\exists l_y \sqsubset o.input[i+1]\ /\ l_x.trustee \sqsubset l_y.trustor \\ (iii) & (\forall l \sqsubset o.input[|o.input|])\ l.output.trustee \sqsubset l.input.trustee \end{cases}$$

- The aggregation operation aims at aggregating relationships that have the same trustee, formally: $(\forall o \in \mathbb{O})\ o.type = concatenation \Rightarrow (\forall l \sqsubset o.input)\ o.output.trustee = l.trustee$.

We give below the operations defined by the Guard and the Police trust models.

Guard Trust Model		
Operation set $\mathbb{O}$		
o_{sMT}	=	<name="setManagerTrustworthiness", host="$r_C \trianglerighteq r_G$", type="setting", in="l_{dCM}",out="l_{dCM}" >
o_{aGT}	=	<name="assessGuardTrustworthiness", host="r_C", type="concatenation", in="$(l_{dCM} \wedge l_{rMG})$" , out="l_{tCG}", call="o_{sGT}" >
o_{sGT}	=	<name= "setGuardTrustworthiness", host="r_C", type="setting", in="l_{tCG}", out="l_{tCG}" >
o_{sCF}	=	<name= "setChiefFeedback", host="r_C", type="setting", out="l_{dCG}", call="o_{aGR}" >
o_{aGR}	=	<name="assessGuardReputation", host="r_M", type="aggregation", in="l_{dCG}", out="l_{rMG}", call="o_{sGR}" >
o_{sGR}	=	<name="setGuardReputation", host=r_M, type=setting, in=l_{rMG}, out=l_{rMG} >

Police Trust Model		
Operation set $\mathbb{O}$		
o_{sTA}	=	<name="setTrustedAuthority", host=r_A, type=setting, in=l_{dAA},out=l_{dAA} >
o_{sPA}	=	<name="setPoliceAuthority", host=r_P, type=setting, in=l_{dPA}, out=l_{dPA} >
o_{sPM}	=	<name="setPoliceMember", host=r_A, type=setting, in=l_{dAP}, out=l_{dAP} >
o_{cRP}	=	<name="concatRemotePolice", host=r_A, type=concatenation, in=$l_{dAA} \wedge (l_{dAP} \trianglerighteq l_{tAP})$ out=l_{tAP}, call=o_{aRP} >
o_{aRP}	=	<name="aggregateRemotePolice", host=r_A, type=aggregation, in=l_{tAP} out=l_{tAP} >
o_{sRP}	=	<name="setRemotePolice", host=r_A, type=setting, in=l_{tAP} out=l_{tAP} >
o_{aPC}	=	<name="assessPoliceAccreditation ", host=r_P, type=concatenation, in=$l_{dPA} \wedge (l_{dAP} \trianglerighteq l_{tAP})$ out=l_{tPP}, call=o_{sPC} >
o_{sPC}	=	<name="setPoliceAccreditation", host=r_P, type=setting, in=l_{tPP} out=l_{tPP} >

Focusing on the Guard trust model, we can see that the operation o_{sCF} is performed by the *Chief* to set the relation l_{dCG} with its feedback. This operation will then trigger the operation o_{aGR} that is performed by the *Manager* to compute (by aggregation) the reputation of the *Manager*, which then leads to update the corresponding *N:1* relation o_{sGR}. In the Police trust model, each *Policeman* is able to assess the accreditation of another *Policeman* through its *Authority* ($o_{aPC} \rightarrow o_{sPC}$). An *Authority* can manage the accreditation of its *Policeman* (o_{sPM}) or compute transitively the *Policeman* accreditation through trustee *Authorities* ($o_{cRP} \rightarrow o_{aRP} \rightarrow o_{sRP}$).

Trust graph TG: We associate the definition of a *trust graph* with any trust model TM for the sake of graphical representation. Specifically, the trust graph $TG(\mathbb{R}, \mathbb{E})$ associated with a given TM is a directed graph with the vertices representing the set of roles $\mathbb{R}$ of TM, and the set of edges $\mathbb{E}$ representing the relationship between roles according to $\mathbb{L}$. Hence, each edge is labeled by the referenced relation l from the set of relations $\mathbb{L}$ and the type of that relation, i.e., *1:1*, *1:N* or *N:1* (see Figure 6.3).

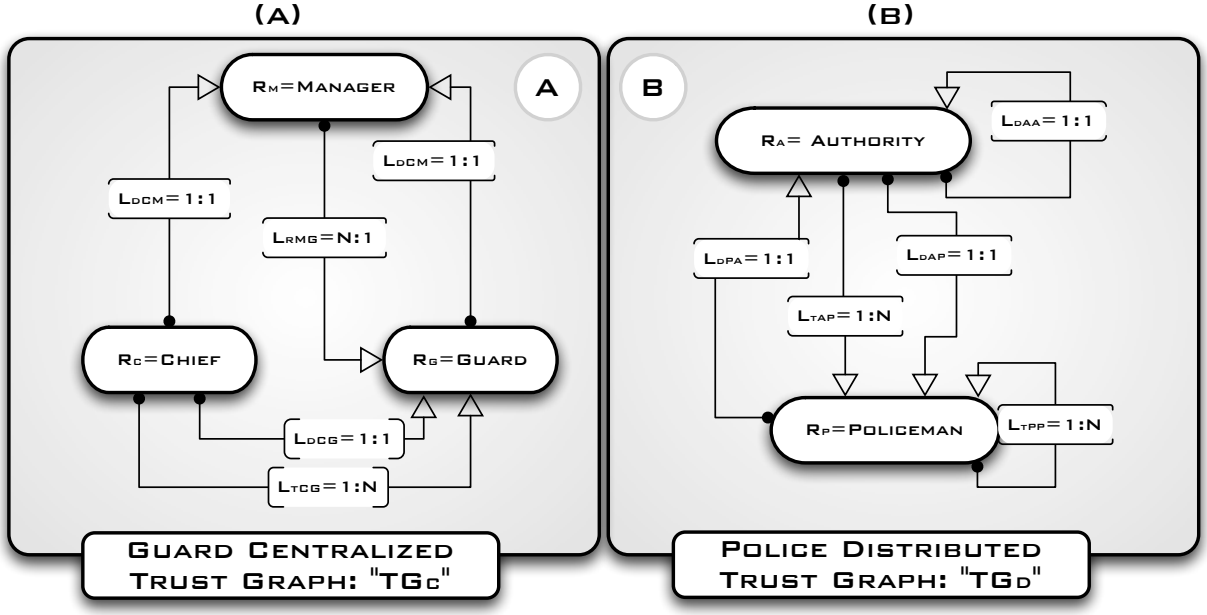


Figure 6.3: Terrorist Alert Trust Graph TG_C and TG_D.

6.3 Composing Trust Models

The aim of composing different trust models is to provide roles of a given model (which we call target model T) the ability to assess roles that pertains to another trust models (which we call source model S). For instance, according to the terrorist alert scenario, *Guards* should help *Policemen* to find suspects. This requires that any *Policeman* should be able to assess the accreditation of *Guards*. To do so, mapping of roles from two distinct trust models (TM_x and TM_y) is explicitly defined through a set of mapping rules Ψ_{xy}, as follows:

$$\begin{aligned} \Psi_{xy} = \{\psi^k_{ST} / \psi^k_{ST} = (r_s : TM_S) \rhd (r_t : TM_T)\} \\ Where\ S, T \in \{x, y\},\ S \neq T \end{aligned} \tag{6.5}$$

Each mapping rule associates a source role with a target role so as to define that the source role r_s of TM_S plays the target role r_t in TM_T. In fact, in order to initiate any relationship across trust models, roles of different models have to be mapped into local roles, since roles of a given trust model are designed to communicate only with roles from the same trust model. For instance, in order to enable *Policemen* to assess *Guards* (see Figure 6.4, Step 1), we identify which role in the *Policeman* trust model corresponds to the *Guard*. Intuitively, we can define the rule: $(r_G : TM_C) \rhd (r_P : TM_D)$, which means that *Guards* (i.e., $r_G : TM_C$) of the centralized trust model are seen by the distributed trust model (TM_D) as *Policemen* ($r_P : TM_D$).

Given the specification of trust models, their composition relies on mapping their respective roles. Then, the existing trust relations and operations are extended to relate roles from the composed models, and new assessment operations are required to map trust relations from one model to another. Finally, the resulting mapping and extensions are implemented through mediation [100] so as to make composition transparent to existing systems, which leads us to introduce the corresponding *mediator role*.

Formally, the composition, denoted by $\bigodot_{\Psi_{xy}}$, of two trust models TM_x and TM_y, which introduces the trust model TM_{xy}, is defined as follows:

$$\begin{aligned} TM_{xy} &= TM_x \bigodot_{\Psi_{xy}} TM_y \\ &= < \mathbb{R}_x, \mathbb{M}_x, \mathbb{L}_x, \mathbb{O}_x > \bigodot_{\Psi^{xy}} < \mathbb{R}_y, \mathbb{M}_y, \mathbb{L}_y, \mathbb{O}_y > \\ &= \left\langle \begin{array}{lcl} \mathbb{R}_{xy} & = & \mathbb{R}_x \cup \mathbb{R}_y \cup \mu\mathbb{R}_{xy} \\ \mathbb{M}_{xy} & = & \mathbb{M}_x \cup \mathbb{M}_y \\ \mathbb{L}_{xy} & = & \mathbb{L}^+_x \cup \mathbb{L}^+_y \\ \mathbb{O}_{xy} & = & \mathbb{O}^+_x \cup \mathbb{O}^+_y \cup \mu\mathbb{O}_{xy} \end{array} \right\rangle \end{aligned} \tag{6.6}$$

where:

- Ψ_{xy} is the set of mapping rules over roles that enables the composition of TM_x and TM_y;
- $\mu\mathbb{R}_{xy}$ and $\mu\mathbb{O}_{xy}$ are the new sets of mediator roles and mediation operations, respectively;
- ($\mathbb{L}^+_x$ and $\mathbb{L}^+_y$) and ($\mathbb{O}^+_x$ and $\mathbb{O}^+_y$) are the extended relations and operations, respectively.

As illustrated in Figure 6.4, the composition process is performed in two steps:

1. *Role Mapping*: Processes the mapping rules of Ψ_{xy} in order to enable the target model to manage the source roles, for instance: the *Guard* is perceived as a trustee *Policeman* into the Police trust model.
2. *Recommendation mediation*: If indirect relations are extended, we introduce a mediator to play the role of a recommender from the target model, which is able to assess the source role from the source trust model. For instance, the mediator can play the role of an *Authority* that recommends to other *Authorities*, a given *Guard*, according to its reputation. To do so, the mediator will further play the role of a *Guard* to retrieve that reputation.

In the following, we elaborate, *role mapping* and *recommendation mediation* to generate the sets of mediator roles, and mediation operations (i.e., $\mu\mathbb{R}_{xy}$, and $\mu\mathbb{O}_{xy}$) and extended relations and operations (i.e., $\mathbb{L}^+_x$, $\mathbb{L}^+_y$, $\mathbb{O}^+_x$, $\mathbb{O}^+_y$).

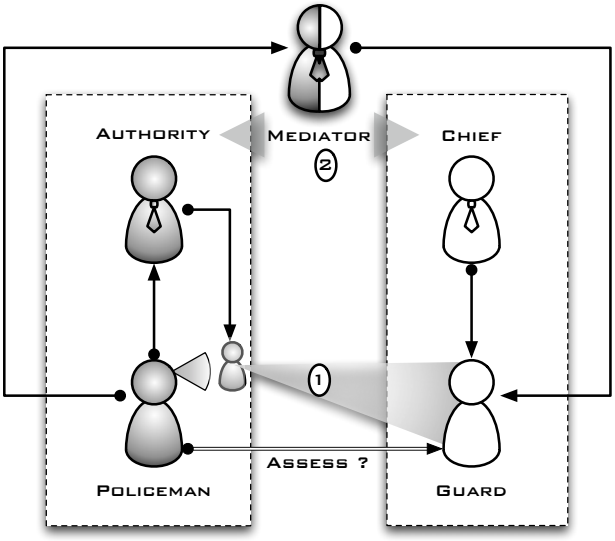


Figure 6.4: Composition Process

6.3.1 Role Mapping

The computation of the composition of trust models TM_x and TM_y is detailed in Algorithm 6. The algorithm iterates on mapping rules (Line 4) for each of which it extends relation sets, $\mathbb{L}_x^+$ and $\mathbb{L}_y^+$ (Lines 6-10). The aim of extending relations is to enable those relations to handle trustors and trustees from another trust model. More precisely, for any trust relation: $l =<$ name, ctx, type, trustor, trustee, metric$>$ of $\mathbb{L}_x$ and $\mathbb{L}_y$ of the composed models TM_x and TM_y, its *trustee* and *trustor* elements are possibly extended to account for mapping between roles.

Algorithm 6 details the corresponding extension where, as shown in Lines 6 to 10, the extension of trust relations relies on each mapping rule. Each rule defines which local trustee (target role r_t) corresponds to the source role (r_s). Therefore, all the relations l_i (from the target trust model) that consider the target role as a trustee (Line 7) are extended with the source role (Line 8).

For instance, applying Algorithm 6 sto compose the Guard TM_C and the Police TM_D trust models, with the mapping rule $r_G : TM_C \rhd r_P : TM_D$ leads to extend TM_D as follows:

Police Trust Model		
l_{dAA}	=	$<$ name="TrustAuthority", ctx= "Security", type="*1:1*", trustor="r_A", trustee="r_A", metric="m_{Aut}" $>$
l_{dPA}	=	$<$ name="PoliceAuthority", ctx= "Security", type="*1:1*", trustor="r_P", trustee="r_A", metric="m_{Aut}" $>$
l_{dAP}	=	$<$ name="PoliceMember", ctx= "Security", type="*1:1*", trustor="r_A", trustee="$r_P \trianglerighteq r_G$", metric="m_{Acc}" $>$
l_{tPP}	=	$<$ name="Colleague", ctx= "Security", type="*1:N*", trustor="r_P", trustee="$r_P \trianglerighteq r_G$", metric="m_{Acc}" $>$
l_{tAP}	=	$<$ name="RemoteTrustPolice", ctx= "Security", type="*1:N*", trustor="r_A", trustee="$r_P \trianglerighteq r_G$", metric="m_{Acc}" $>$

All the relations that are depicted in gray in the table above sink to the *Policeman* role. Thus, thanks to the given mapping rule, these relations are extended with the *Guard* role as a trustee.

As stated earlier, the next composition step processes the extended indirect relations (the set $i\mathbb{L}$ -Line 5 and 10-) that go across trust models. The goal is to introduce a mediator as a potential recommender of each indirect relation in order to help the target model to set (bootstrap and update) the source role trustworthiness according to the trust knowledge of the source trust model. Algorithm 7 is hence trigged to process each indirect relation that is extended. However, before that, the set $i\mathbb{L}$ is parsed in order to avoid redundancy, namely, any extended indirect relation that needs another extended indirect relation to be assessed. In this case, solving the required indirect relation will solve those that needs that relation. To do so, Algorithm 6 searches for the operations that assess indirect relations (Lines 11-12), and removes from $i\mathbb{L}$ (Line 17-18) all the relations requiring an extended indirect relation as input to be assessed(Lines 13-16).

Algorithm 6: Trust_Models_Composition(TM_x, TM_y, Ψ^{xy})

Input(s) : Trust models $TM_x =< \mathbb{R}_x, \mathbb{M}_x, \mathbb{L}_x, \mathbb{O}_x >$ and $TM_y =< \mathbb{R}_y, \mathbb{M}_y, \mathbb{L}_y, \mathbb{O}_y >$
The set of Mapping rules Ψ^{xy}

Output(s): The trust model composition $TM_{xy} =< \mathbb{R}_{xy}, \mathbb{M}_{xy}, \mathbb{L}_{xy}, \mathbb{O}_{xy} >$

```
1  begin
       // Initialize trust models sets for composition
2      L_x^+ = L_x ; L_y^+ = L_y
3      O_x^+ = O_x ; O_y^+ = O_y
4      foreach (ψ_k^xy = (r_s : TM_{S∈{x,y}}) ▷ (r_t : TM_{T∈{x,y},S≠T})) ∈ Ψ^xy) do
5          iL = ∅                                   /* Set of indirect relations that are extended */
           // Extend target model relation with r_s
6          foreach (l_i ∈ L_T^+) do                 /* Find relations with the trustee r_t */
7              if r_t ⊏ l_i.trustee then
8                  l_i.trustee = l_i.trustee ⊻ r_s   /* Add r_s as a trustee */
9                  if l_i.type ≠ 1:1 then            /* the extended relation is indirect */
10                     iL = iL ∪ {l_i}
           // Delete redundant extended indirect relations
11         foreach (o_i ∈ O_T^+) do                 /* Find operations that assess indirect relations */
12             if o_i.output ∈ iL then
13                 redundant = false
14                 foreach (l_j ⊏ o_i.input) do
15                     if (l_j ≠ o_i.output) and (l_j ∈ iL) then
16                         redundant = true
17                 if redundant = true then
18                     iL = iL − {l_j}
19         μr_k = null
20         Relation_Mediation( r_s, μr_k, iL, L_S^+, L_T^+, O_S^+, O_T^+, μO_xy)
21         if μr_k ≠ null) then                     /* At least one indirect relation was been mediated */
22             μR_xy = μR_xy ∪ {μr_k}
23     R_xy = R_x ∪ R_y ∪ μR_xy
24     M_xy = M_x ∪ M_y
25     L_xy = L_x^+ ∪ L_y^+
26     O_xy = O_x^+ ∪ O_y^+ ∪ μO_xy
27 end
```

6.3.2 Recommendation Mediation

In this section we describe the mediation process over recommendations, which enables a mediator to recommend the source role to the target trust model, from the trust assessment that is provided by the source model. This process is performed by Algorithms 7 and 8 in four steps:

- *Step 1: Find a target recommender* r_{rec_t} of each extended indirect relation l_s that should maintain a direct relation with the source role l_{rec_t} (see Figure 6.5(a)(b)). This step is described in Algorithm 7 (Lines 2-6). The operation set of the target model is parsed in order to retrieve for each extended indirect relation a potential recommender (line 6).
- *Step 2: Find a source requestor* r_{req_s} that is able to indirectly assess the source role r_s (i.e., r_{req_s} has a *1:N* or *N:1* trust relation with r_s) (see Figure 6.5(c)(d)). We are looking for an indirect trust relation (l_{req_s}) because the mediator is passive and it is defined to retrieve (i.e., indirectly) trust values and not to give its opinion. Moreover, transitive based-trust relations (i.e., *1:N*) have more priority than reputation based-trust relations (Lines 10 and 14), since *1:N* relation is the less restrictive in terms of managing and storing the history of the relation. Another constraint consists to find a relation that is applied in the same or similar context of the recommended relation l_{rec_t} (Lines 11-13 and 15-17). Once, Algorithm 7 finds the most appropriate relation, we can choose one of its trustor as requestor

role r_{req_s} (Line 19).

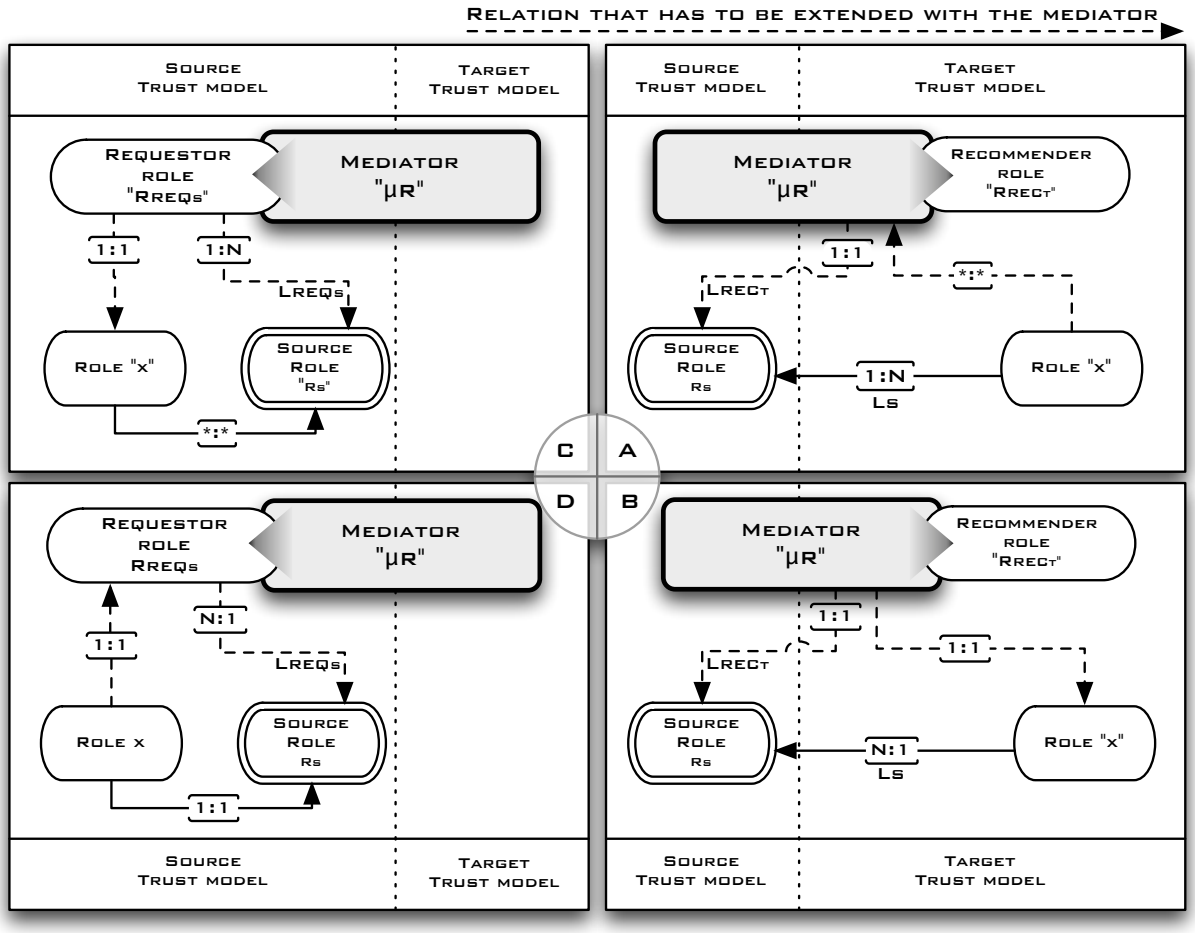


Figure 6.5: Relation Mediation

- *Step 3: Extend the relations of the source and the target models* in order to enable the mediator μr to play the roles r_{rec_t} and r_{req_s}. Furthermore, in the source model, if the relation l_s is a reputation based-trust relationship (see Figure 6.5(b)), the mediator has to establish a trust relationship with the trustor of the source relation in order to be able to send its recommendation (see Algorithm 7, Lines 27-30). Similarly, in the target model, if the requested relation l_{req_s} is of type *1:N* (see Figure 6.5(c)), the mediator has to establish a relationship with at least one recommender of this relation (see Algorithm 7, Lines 31-34).
- Step 4: Extend the operations of each model with the mediator μr, in order to allow it to perform translation over new mediation operations. In fact, the mediator will have to perform all the operations that assess its managed relations (i.e., the relations where the mediator is a trustor). In other words, each operation $o =<$ name, host, type, input, output, call$>$ of O_x^+ and O_y^+, that has to be performed by the mediator (i.e., $\mu r \sqsubset output.trustor$) will be extended with the mediator role (Algorithm 8 Lines 2-5). Then, in order to link and translate requested relation l_{req_s} to the recommended relation l_{rec_t}, we define a new type of operation called *Mediation* operation (μo). More precisely, consider a requested and a recommended relations (l_{req_s} and l_{req_t} respectively):

Algorithm 7: Relation_Mediation($r_s, \mu r, i\mathbb{L}, L_S^+, L_T^+, \mathbb{O}_S^+, \mathbb{O}_T^+, \mu\mathbb{O}_{xy}$)

Input(s) : Role source r_s ; Mediator role μr ;
Set of the extended indirect relation $i\mathbb{L}$; Extended relation sets $\mathbb{L}_s^+$ and $\mathbb{L}_t^+$;
Extended operation sets $\mathbb{O}_s^+$ and $\mathbb{O}_t^+$; Mediation operation set $\mu\mathbb{O}_{xy}$

Output(s): Extended relation sets $\mathbb{L}_s^+$ and $\mathbb{L}_t^+$;
Extended operation sets $\mathbb{O}_s^+$ and $\mathbb{O}_t^+$ and Mediation operation set $\mu\mathbb{O}_{xy}$

```
1  begin
2    foreach (o_i ∈ O_T^+ / o_i.output ∈ iL) do        /* Find operations that assess indirect relations */
3      l_t = o_i.output
4      if (l_t.type=1:N and o_i.type=concatenation) or (l_t.type=N:1 and o_i.type=aggregation) then
         // Select the input relations that sink to r_s
5        foreach (l_rec_t ⊏ o_i.input / r_s ⊏ l_rec_t.trustee ) do
6          r_rec_t = l_rec_t.trustor[1]        /* Select the first trustor as a potential recommender */
           // Find similar relation in the source trust model
7          l_req_s = null
8          sim^0 = TH^0
9          foreach (l_i ∈ L_S^+ / (l_i.type ≠ 1:1 and r_s ⊏ l_i.trustee)) do        /* Parse L_S^+ */
10           if (l_req_s = null or l_req_s.type = l_i.type) then
11             if Sim(l_i.ctx, l_rec_t.ctx) ⩾ sim^0 then
12               l_req_s = l_i
13               sim^0 = Sim(l_i.ctx, l_rec_t.ctx)
14           else if (l_req_s ≠ null and l_req_s.type =N:1 and l_i.type =1:N) then
               // 1:N has higher priority than, N:1
15             if Sim(l_i.ctx, l_rec_t.ctx) ⩾ TH^0 then
16               l_req_s = l_i
17               sim^0 = Sim(l_i.ctx, l_rec_t.ctx)
18         if l_req_s ≠ null then        /* A similar relation is found */
19           r_req_s = l_req_s.trustor[1]        /* Select the first trustor as a potential requestor */
20           if μr = null then
21             μr = new role
22           l_req_s.trustor = l_req_s.trustor ⩒ μr  /* Extend requested relation with μr as a trustor */
23           l_rec_t.trustor = l_rec_t.trustor ⩒ μr  /* Extend recommended relation with μr as a trustor
             */
             // Extend relations to enable μr to play r_rec_t and r_req_s
24           foreach (l_i ∈ L_T^+ ∪ L_S^+) / ((r_rec_t ⩒ r_req_s) ⊏ l_i.trustee) do
25             if (¬(μr ⊏ l_i.trustee) then
26               l_i.trustee = l_i.trustee ⩒ μr        /* Add μr as a trustee */
27           if l_t.type =N:1 then        /* Link the mediator to the reputation Manager */
28             foreach (l_i ∈ L_T^+) / (l_i.type =1:1 and l_i.trustee ⊔ l_t.trustor ≠ ∅)) do
29               if (r_rec_t ⊏ l_i.trustor) and ¬(μr ⊏ l_i.trustor) then
30                 l_i.trustor = l_i.trustor ⩒ μr        /* Add μr as a trustor */
31           else if l_req_s.type =1:N then        /* Link the mediator to the a source recommender */
32             foreach (o_i ∈ O_S^+) do
33               foreach (l_j ∈ o_i.input[1]) do
34                 l_j.trustor = l_j.trustor ⩒ μr ; Break
               // Call the mapping operation algorithm
35           Operation_Mediation(μr, l_rec_t, l_req_s, L_S^+, L_T^+, O_S^+, O_T^+, μO_xy )
36 end
```

l_{req_s} =<name="ns", ctx="cs", type="*1:N*" or "*N:1*", trustor="$r_{req_s} \trianglerighteq \mu r$", trustee="$r_s$", metric="ms"> of TM_s where $l_{req_s} \in \mathbb{L}_s^+$, $r_{req_s} \in \mathbb{R}_s$, and $ms \in \mathbb{M}_s$.

l_{req_t} =<name="nt", ctx="ct", type="*1:1*", trustor="$r_{rec_t} \trianglerighteq \mu r$", trustee="$r_s$", metric="mt"> of TM_t where $l_{rec_t} \in \mathbb{L}_t^+$, $r_{rec_t} \in \mathbb{R}_t$, and $mt \in \mathbb{M}_t$.

The mediation operation μo (see Algorithm 8 Lines 10-13) will mainly have to translate and normalize the metric value ms of l_{req_s} into the metric value mr of l_{rec_t},

i.e., $\mu o =<$ name, host= μr, type=Mediation, in=l_{req_s}, out = $l_{rec_t} >$.

To do so, we consider that the mediator (μr) embeds a library of mediation functions that translate and normalize heterogeneous trust metrics, which are invoked by the mediation operation μo.

After that, we create, through the mediator, a continuation from the source model to the target model. The mediator will call the mediation operation after assessing and setting ($o_{set_s} \rightarrow \mu o$) the requested relation l_{req_s} (see Algorithm 8 Line 14). In turn, after performing the translation, the mediation operation can call the operation o_{set_t} ($\mu o \rightarrow o_{set_t}$) in order to set the recommended relation l_{rec_t} (see Algorithm 8 Line 15).

Algorithm 8: Operation_Mediation($\mu r, l_{rec_t}, l_{req_s}, L_S^+, L_T^+, \mathbb{O}_S^+, \mathbb{O}_T^+, \mu\mathbb{O}_{xy}$

Input(s) : The mediator role μr;
The recommended relation l_{rec_t} and the requested relation l_{req_s};
Extended relation sets $\mathbb{L}_s^+$ and $\mathbb{L}_t^+$;
Extended operation sets $\mathbb{O}_s^+$ and $\mathbb{O}_t^+$;
Mediation operation set $\mu\mathbb{O}_{xy}$.

Output(s): The source, the target and the mediationw operation sets: $\mathbb{O}_s^+$, $\mathbb{O}_t^+$ and $\mu\mathbb{O}_{st}$

1 **begin**
  // Extend mediator operations
2 **foreach** *($o_i \in \{O_S^+ \cup O_T^+\}$) / ($\mu r \sqsubset o_i.output.trustor$)* **do**
3 $o_i.host = o_i.host \trianglerighteq \mu r$
4 **if** $(o_i.output = l_{rec_t})$ *and* $(o_i.type = setting)$ **then**
5 $o_{set_t} = o_j$
6 **if** $(o_i.output = l_{req_s})$ *and* $(o_i.type = setting)$ **then**
7 $o_{set_s} = o_j$
8 $\mu o = new\ operation$
9 $\mu o.type = mediation$
10 $\mu o.input = l_{req_s}$
11 $\mu o.output = l_{rec_t}$
12 $o_{update_s}.call = o_{update_s}.call \wedge \mu o$
13 $\mu o.call = \mu o.call \wedge o_{update_t}$
14 $\mu\mathbb{O}_{xy} = \mu\mathbb{O}_{xy} \cup \{\mu o\}$
15 **end**

In Figure 6.6, we illustrate the composition of TM_C (the centralized Guard Trust Model) and TM_D (the distributed Police Trust Model) resulting from the following mapping rules:

- $(r_G : TM_C) \triangleright (r_P : TM_D)$ (see Figure 6.6(A))
- $(r_P : TM_D) \triangleright (r_G : TM_C)$ (see Figure 6.6(B))

As a consequence of applying Algorithm 7 with the first rule, *Policemen* are able to transitively assess the accreditation of *Guards* through the mediator that maintains a direct relationship with a *Guard* ($l_{rec_t} = l_{dAP}$). This relation is set according to the reputation of that *Guard* into TM_C. The mediator gets this value by playing the role of a *Chief* that assesses the trustworthiness of a *Guard* transitively through the *Manager* ($l_{req_s} = l_{tCG}$).

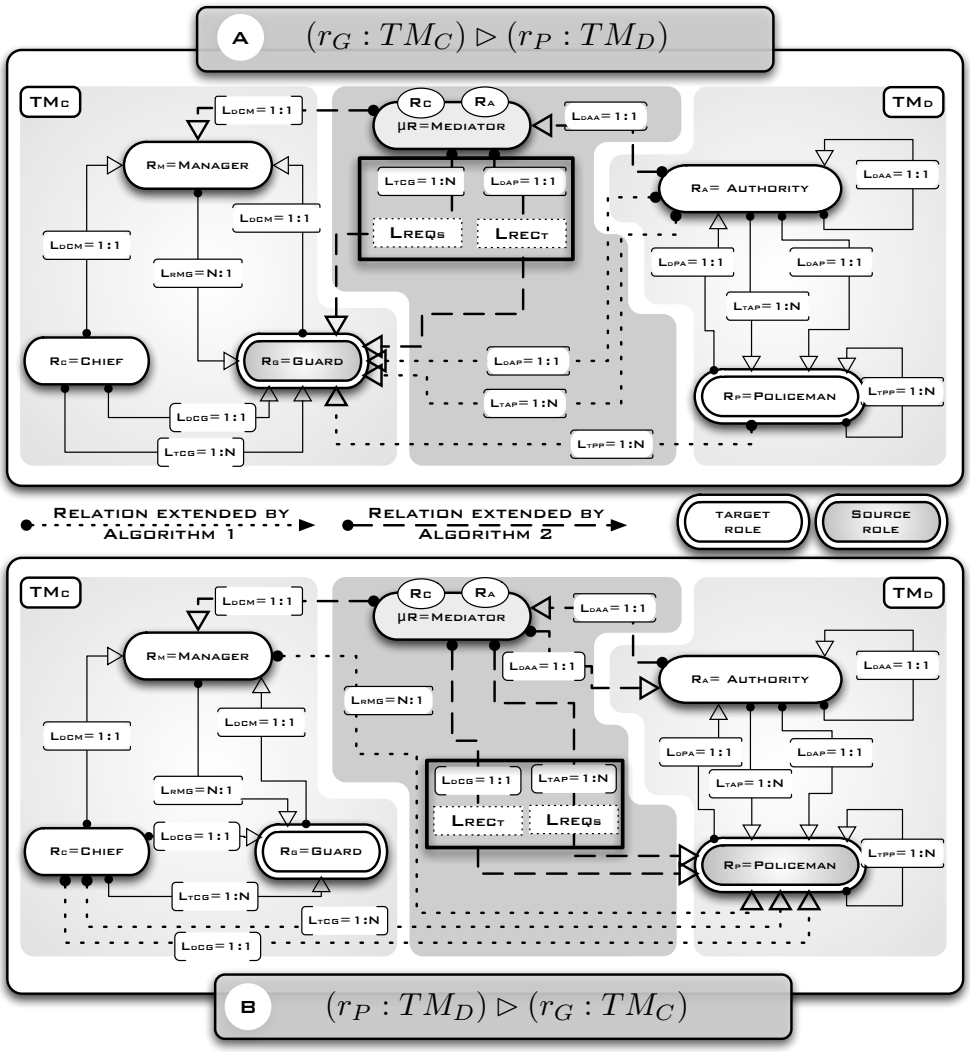


Figure 6.6: Trust Graph Composition of the Guard and the Police Trust Models

Concerning the second rule, Algorithm 7 enables the *Manager* to bootstrap and update the reputation of the *Manager* through the mediator that plays the role of a *Chief* that gives it feedback ($l_{rec_t} = l_{dCG}$). To do so, the mediator plays the role of an *Authority* that assesses transitively the accreditation of an unknown *Policeman* ($l_{req_s} = l_{tAP}$).

Then, as depicted in the table below, by applying Algorithm 8, all operations highlighted in gray are extended to handle the extended relations that have to be performed by μr. Moreover, two mediation operations are defined, namely: μo_{mGP} and μo_{mPG} in order to respectively convert probability values to accreditation levels and conversely .

Guard Trust Model		
Operation set $\mathbb{O}$		
o_{sMT}	=	<name="setManagerTrustworthiness", host="$r_C \veebar r_G \veebar \mu r$", type="setting", in="$l_{dCM}$",out="$l_{dCM}$" >
o_{aGT}	=	<name="assessGuardTrustworthiness", host="$r_C \veebar \mu r$", type="concatenation", in="$(l_{dCM} \wedge l_{rMG})$", out="l_{tCG}, call=o_{sGT}" >
o_{sGT}	=	<name= "setGuardTrustworthiness", host="$r_C \veebar \mu r$", type="setting", in="l_{tCG}", out="l_{tCG}", call="μo_{mGP}" >
o_{sCF}	=	<name= "setChiefFeedback", host="r_C", type="setting", out="l_{dCG}", call="o_{aGR}" >
o_{aGR}	=	<name="assessGuardReputation", host="r_M", type="aggregation", in="l_{dCG}", out="l_{rMG}", call="o_{sGR}" >
o_{sGR}	=	<name="setGuardReputation", host="r_M", "type=setting", in="l_{rMG}", out="l_{rMG}" >
μo_{mGP}	=	<name="mediateGuardPolice", host="μr", type="mediation", in="l_{tCG}", out="l_{dAP}", call="o_{sPM}" >

Police Trust Model		
Operation set $\mathbb{O}$		
o_{sTA}	=	<name="setTrustedAuthority", host=$r_A \trianglerighteq \mu r$, type=setting, in=$l_{dAA}$,out=$l_{dAA}$ >
o_{sPA}	=	<name="setPoliceAuthority", host="r_P", type=setting, in="l_{dPA}", out="l_{dPA}" >
o_{sPM}	=	<name="setPoliceMember", host="r_A", type="setting", in="l_{dAP}", out="l_{dAP}" >
o_{cRP}	=	<name="concatRemotePolice", host=$r_A \trianglerighteq \mu r$, type=concatenation, in="$l_{dAA} \wedge (l_{dAP} \trianglerighteq l_{tAP})$", out="$l_{tAP}$", call="$o_{aRP}$" >
o_{aRP}	=	<name="aggregateRemotePolice", host="$r_A \trianglerighteq \mu r$", type="aggregation", in="l_{tAP}", out="l_{tAP}" >
o_{sRP}	=	<name="setRemotePolice", host="r_A", type="setting", in="$l_{tAP \trianglerighteq \mu r}$" out="$l_{tAP}$", call="$\mu o_{mGP}$" >
o_{aPC}	=	<name="assessPoliceAccreditation ", host="r_P", type="concatenation", in="$l_{dPA} \wedge (l_{dAP} \trianglerighteq l_{tAP})$", out="$l_{tPP}$", call="$o_{sPC}$" >
o_{sPC}	=	<name="setPoliceAccreditation", host="r_P", type="setting", in="l_{tPP}", out="l_{tPP}" >
μo_{mPG}	=	<name="mediatePoliceGuard", host="μr", type="mediation", in="l_{dCG}", out="l_{tAP}", call="o_{sCF}" >

6.4 Conclusions and Further Research Directions

In this chapter, we have introduced a trust meta-model as the basis to express and to compose a wide range of trust models. We have illustrated the proposed meta-model through a concrete terrorist alert scenario, where two heterogeneous models (i.e., Guard and Police) are modeled, composed and mediated. Such a composition is specified in terms of mapping rules between roles. Rules are then processed by a set of mediation algorithms to overcome the heterogeneity between the trust metrics, relations and operations associated with the composed trust models.

Next chapter tackles CONNECT security requirements, and introduces trust-based access control solutions based on contract and policies. To do so, these security solutions refer to the trust composition approach (that is provided in this chapter) to overcome heterogeneity of NSs' trust model.

Currently, we are actively working with the partners that are developing the CPMM meta-model to extend it with our trust meta-model. We are further designing and implementing the Security and Trust (SXT) Enabler.

We are also investigating the use of ontologies to specify the semantics of trust model elements and thus possibly infer the mapping rules as well as infer the similarity of trust relations from the semantics.

7 Security Aspects

To study security aspects, we distinguish two levels: the CONNECT infrastructure level, consisting of the enablers, and the Networked System level, consisting of the entities that ask for communicating.

With an eye at the new assumptions and at the architecture developed in this second year (D1.2) [33], in the following (Section 7.1) we extend the analysis of threat models of the CONNECT world that we have preliminarily described in D5.1. Then, while in WP3 we focus on security aspect at CONNECTor synthesis and deployment time, here, in WP5, we deal with security aspects at run-time. In particular, according to the security policies that have to be guaranteed and referring to the considered threat model, we describe the following two frameworks:

- Security-by-Contract-with-Trust (in Section 7.2), and
- Access control negotiation with trust (in Section 7.3).

We have released this year a prototype of the former, please refer to the Appendix-Prototypes for getting related information.

Note that, both these frameworks deal also with trust. Indeed, during the second year of the project, some work has been done for integrating the two concepts of security and trust. In particular, the Security-by-Contract-with-Trust mechanism, as we will show after, is a run-time enforcement mechanism driven by both security and trust aspects. On the other hand, the access control negotiation with trust framework is not yet fully integrated with the trust meta-model described in Chapter 6. We plan to integrate these two approaches as future work within the next year of the project.

As for the other WP5 approaches, we discuss (in Section 7.4) the application of both frameworks to the case study of The Terrorist Alert and draw conclusions (in Section 7.5).

7.1 Analysis of the Threat Models

Within the second year of the CONNECT project, several hypothesis are made upon the behaviour and features of CONNECT infrastructure, Networked systems, and CONNECTors. Indeed, according to the D1.2 [33]:

- **Networked Systems** are systems that manifest the will to connect to other systems for fulfilling some intent identified by their users and the applications executing upon them.
- **Enablers** are networked entities in the environment of networked systems that incorporate all the intelligence and logic offered by CONNECT for enabling connection between heterogeneous networked systems. Enablers constitute the CONNECT enabling architecture.
- **CONNECTors** are the emergent connectors produced by the action of enablers.

The result of the interaction of these three agents is a **CONNECTed** systems, that is actually the outcome of the successful creation and deployment of CONNECTors.

In order to guarantee security in the CONNECT framework, we identify the possible threat models of the CONNECT scenario (Table 7.1) by investigating all possible relations that may be established among the listed entities.

To this aim we identify three different security layers:

The CONNECT Infrastructure : The CONNECT infrastructure is composed by several enablers whose functions are integrated in order to obtain a CONNECTor compliant to the requirements of the Networked Systems that have to communicate. Once a communication request is received by the CONNECT framework, and processed by the Learning Enabler and the Discovery Enabler, all the information are elaborated and a CONNECTor is synthesized by the Synthesis Enabler. Sharing information among different enablers is a security relevant aspects. Indeed, some enabler could be malicious and could provide wrong information to the other ones in order to prevent the communication.

Between the CONNECT Infrastructure and the NSs : When two Networked Systems ask to establish a communication, they have to authenticate themselves to the CONNECT infrastructure in such a way that CONNECT is able to discover their interface protocols and their communication constraints

	CONNECT Infrastructure	Between CONNECT and NSs	CONNECTed system
Synthesis Time	i)All Enablers are safe; ii)Some Enabler is malicious	i)The NSs are safe; ii) At least one NS is malicious	
Deployment Time		i)The Nss are safe; ii) At least one NSs is malicious	
Execution Time	i)All Enablers are safe; ii)Some Enabler could be malicious	i)The Nss are safe; ii) At least one NSs is malicious	i)Each NS could be a malicious agent ii) CONNECT does not trust NSs

Table 7.1: Security in CONNECT According to the CONNECTor Life-cycle

as, *e.g.*, , security policies, dependability requirements, and so on. In this layer we have to foresee mechanisms for guaranteeing that this *subscribe* phase is performed in a secure way. In this scenario we assume that the CONNECT infrastructure cannot behave as a malicious entity, *i.e.*, it never tries to damage the Networked Systems security. However, no assumptions are done on the Networked Systems. The following threat models can be pointed out:

- the NSs behave as expected with respect to the CONNECT infrastructure.
- There is at least one NS that tries to attack the CONNECT infrastructure.

Note that when these threats arise, the CONNECTor has not yet been synthesized.

The CONNECTed System : Once the CONNECTor is synthesized, the communication among the two Networked systems is established. From the security prospective, we have to guarantee that the CONNECTed system, i.e., the system composed by the two Networked Systems and the CONNECTor, is secure. Also in this case we can distinguish different threat models. Starting from the assumption that the CONNECTor is secure by construction, we have that:

- At least one of the two NSs involved in the communication is malicious with respect to the CONNECTor.
- The two NSs try to attack each other.

Another meaningful distinction needed for guaranteeing security in CONNECT is the one based on the CONNECTor life-cycle. Indeed, we can distinguish three phases as follows:

- Synthesis time. At this stage, the CONNECT infrastructure collects all the NSs requests and characteristics (*affordance* [33])and synthesizes through the Synthesis enabler a CONNECTor that satisfies both functional and non-functional requirements.
- Deployment time. Referring to [33], the Deployment enabler is in charge of composing the required functionality to ensure that CONNECTors will communicate with the legacy networked systems and of deploying and managing the executable code of the CONNECTors in the network, *i.e.*, if it is distributed to the components that can verify the (fragment of) CONNECTor they received or it is run in a centralized way.
- Execution time. At this stage, the CONNECTor is executed on each component that can also apply a run-time enforcement mechanism in order to assure that local security policies are satisfied [40].

In this deliverable, we mainly focus on security at execution time. Security at synthesis and deployment time is dealt with in WP3. Hence, in the following we will show how and when we are able to guarantee security policy in the CONNECT framework. To do this we investigate separately the two cases of decentralized and centralized deployment and execution of the CONNECTor.

7.2 Security of the CONNECTed System at Execution Time

Let us suppose that each Networked System sets a *local policy* that has to be locally satisfied inside the Networked System itself. Such security policy is the combination of a *private* policy and a *public* one. A

private policy is a security policy that a Networked System does not declare to the external world, *e.g.*, a policy that speaks about sensitive data. On the contrary, a public policy is a policy that the Networked System considers freely deliverable, *e.g.*, a policy that does not involve information that the Networked System considers private. Clearly two limit cases exist: i) a Networked System has only a private policy or ii) it sets only a public policy.

This can speak about local security relevant actions (*directly enforceable policy*) or also about information sent to the external world (*delegated enforceable policy*), *e.g.*, to the other Networked System.

Definition 4 *A local security policy P_1 of the NS NS_1 is a policy that can always be locally enforced.*

Example of local policies are the *sanding-box* policies, *i.e.*, do not access to ports different from the declared ones", *access control* policies, and so on.

On the other hand, we refer to *global policy* whenever we speak about a security policy that involves both the NSs and the CONNECTor itself, *i.e.*, a global security policy is a security policy of the CONNECTed system as a whole.

In order to establish a secure communication, each NS sends to the CONNECT infrastructure the communication request and each possible public security policy. The CONNECT infrastructure receives, attached to the communication request, the security policies and is in charge to generate, verify, and deploy the CONNECTor.

As said, here we focus on security aspects at execution time, *i.e.*, we present, whether it is possible, how certain security policies are guaranteed while two Networked Systems communicate through the CONNECTor.

In this scenario, assuming that the CONNECTor does not intentionally attack the NSs, the possible threat models we obtain are the following ones:

- One of the two NSs involved in the communication is malicious with respect to the CONNECTor;
- Both NSs try to attack each other.

We now present different possible solutions to guarantee security under these assumptions, according to the CONNECTor deployment strategy.

7.2.1 Centralized Deployment of the CONNECTor

Let us now assume that the deployment of the CONNECTor is centralized. This means that the CONNECTor is run by a deputy enabler (for instance the Security & Trust enabler), *i.e.*, it is run into the CONNECT infrastructure.

Since we are working under the assumption that the CONNECT infrastructure is secure and trusted, we provide solution for dealing with security requirements at synthesis time more than at run-time. Indeed, both the CONNECTor and the platform on which it is deployed and run are assumed to be safe.

By the way, it is important to notice that, some security requirements can not be satisfied under these assumption. Indeed, let us consider that one of the two NS or both, requires that the communication is established only on secure channel, from the security point of view, we have to guarantee that the two NSs are able to encrypt and decrypt the communication messages. If (at leas one of) the two NSs is not provided of the cryptographic primitive, it is not able to satisfy this request because, even if the CONNECTor is able to encrypt and decrypt messages, during the communication between the two NSs some message is sent in plain text because the CONNECTor does not run on the NSs, so can not encrypt and decrypt messages before the send and receive actions.

7.2.2 Decentralized Deployment of the CONNECTor

Let us suppose that at least one of the two NS is able to run a (piece of) CONNECTor. Generally, let us assume that a CONNECTor is made of three part, each of them is deployed on the two NS and on the *security Enabler*, *i.e.*, the enabler of the CONNECT infrastructure that it is charge of running the CONNECTor and, eventually, enforces its behaviour according to the requested security policies.

We assume that each NS does not share with the other NS its local public security policies before the CONNECTor has been deployed. The public security policies are send only to the CONNECT infrastructure that is considered secure by both the NSs.

Security-by-Contract-with-Trust

Each Networked System runs its own part of the CONNECTor. Two possible scenarios arise:

- the NSs are provided of the Security-by-Contract-with-Trust ($\times$C$\times$T for short) mechanism.
- the NSs are not provided of the S$\times$C$\times$T. In this case we are in the same case of the centralized on that we describe later because we do not have any possibility of controlling the code that run on the NSs

In the following we consider the first case, *i.e.*, that the NSs are able to run the S$\times$C$\times$T for enforcing security policies. In particular, the S$\times$C$\times$T paradigm is applied for guaranteeing at run-time that:

- local private policies, that are not shared neither with the CONNECT infrastructure, are satisfied. Example of local private policies are private policies regarding, for instance the GPS coordinates of the NS itself, the right for accessing to the private data stored on the device of the NS, and so on.
- none NS tries to attack the other NS by manipulating the part of the CONNECTor running on it. Indeed, let us suppose that one of the two NS is a malicious agent. When it receives the part of CONNECTor that it is in charge to run, the NS manipulates it in order to attack the other NS during the communication.

In both these cases, even if the CONNECTor is synthesized in such a way it is not malicious, once it is deployed on a possible malicious platform, the CONNECTor becomes a possible malicious agent. Under this assumption, each NS able to run the S$\times$C$\times$T paradigm can enforce its own private and public security policies. Here we briefly describe how the S$\times$C$\times$T paradigm works.

The S$\times$C$\times$T Workflow The S$\times$C$\times$T workflow [38, 37] results as in Figure 7.1. It consists of the following steps:

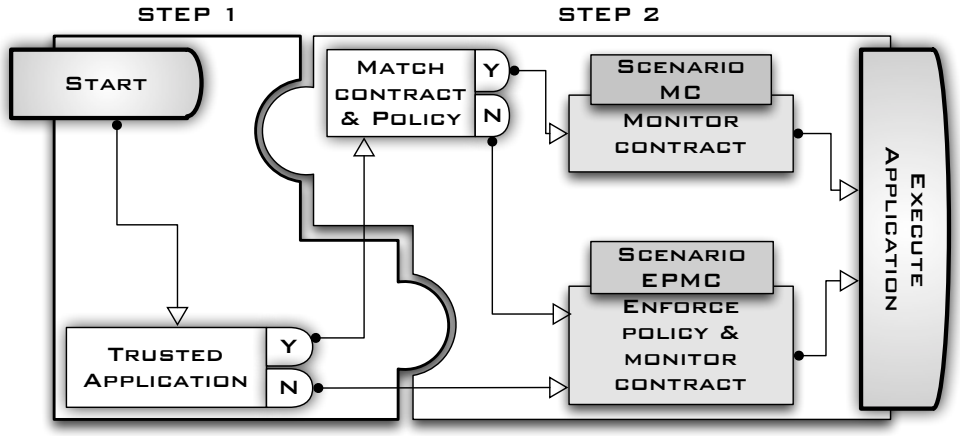


Figure 7.1: The Extended Security-by-Contract Application Workflow

- **Step 1-*Trust Assessment*:** Once (part of) the CONNECTor is downloaded on each Networked System, before executing it, the trust module decides if it trusts or not that the execution of the CONNECTor satisfies its contract. Since NSs may implement different trust models, we apply the trust model composition introduced in Chapter 6. So that, both NSs that aim at being connected can assess the trustworthiness of each other
- **Step 2-*Contract Driven Deployment*:** According to this trust measure, the security module decides if just monitoring the contract (Scenario MC, where MC stands for Monitor of the Contract) or both enforcing the policy and monitoring[1] the contract (Scenario EPMC, where EPMC stands for Enforce the Policy and Monitor the Contract), thus going into one on the scenarios described in Step 3. Indeed, we have two cases:

[1] Note that the the monitoring mechanisms referred here is different from the one described in Chapter 5. Indeed, it is an active and synchronous run-time monitoring running by the NS itself and it is not run by the monitoring enabler.

Scenario MC The contract satisfies the policy. In this case our monitoring/enforcement infrastructure is required to monitor only the CONNECTor contract. Indeed, under these conditions, contract adherence also implies policy compliance. If no violation is detected then the CONNECTor worked as expected. Otherwise, we discovered that a trusted party provided us with a fake contract. The CONNECT infrastructure reacts to this event by reducing the level of trust of the indicted provider and switching to the policy enforcement modality.

Scenario EPMC The contract does not satisfy the policy. Since the contract declares some potentially undesired behaviour, policy enforcement is turned on. Similarly to a pure enforcement framework, our system guarantees that executions are policy-compliant. However, monitoring contract during these executions can provide a useful feedback for better tuning the trust vector. Hence, our framework also allows for a mixed monitoring and enforcement configuration. This configuration is activated on a statistical base.

Let us notice that, in both the previous scenarios, contract monitoring plays a central role. Indeed, a contract violation denotes that a trusted provider released a fake contract.

- **Step 3-*Contract Monitoring vs Policy Enforcement Scenarios***: Depending on the chosen scenario the security module is in charge to monitor either the policy or the contract and save the execution traces (logs).
- **Step 4-*Trust Feedback Inference*:** Finally, the trust module parses the S×C×T produced logs and infers trust feedback.

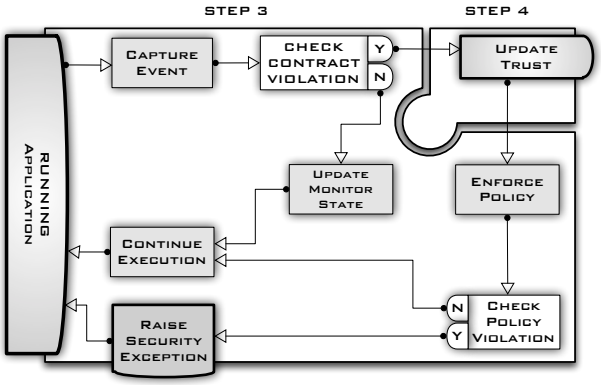


Figure 7.2: The Contract Monitoring Configuration in the MC Scenario

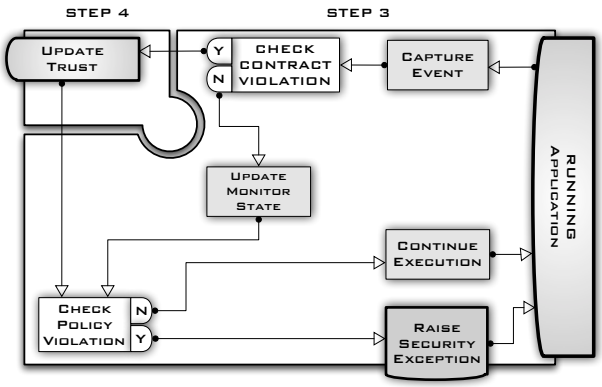


Figure 7.3: The Contract Monitoring Configuration in the EPMC Scenario

Trust measures assigned to security assertions can be adjusted as a result of a contract monitoring strategy. Indeed, trust measures associated with the NS concern on the contract goodness mainly. Up-

dated trust measures will influence on future interactions of the CONNECT infrastructure with that particular NS. In other words, our system penalizes the NS more when the contract does not specify CONNECTor's behaviour correctly, rather when the CONNECTor itself contradicts user's security policy. This is because, since the CONNECTor is secure by construction, if there is an execution of the CONNECTor that violates its contract, this means that the NS that runs it has modified the CONNECTor behaviour. If it occurs, the level of trust of the NS is downgraded.

In [39] a monitoring infrastructure consists in a *policy decision point* (PDP) that holds the actual security state and is responsible for accepting or refusing new actions and *Policy Enforcement Points* (PEPs) that are both in charge of intercepting actions to be dispatched to the PDP and preventing the execution of not allowed operations.

Starting from this model, we extend it by making the PDP also responsible for the contract monitoring operations and for the trust vector updating. Following [24, 39], we assume that both contracts and policies are specified through the same formalism. Hence, the policy enforcement configuration of the PDP keeps unchanged. The PDP must load application contracts as well as security policies dynamically. Moreover, it must be able to run under the two different execution scenarios in Figure 7.1.

EPMC Scenario Both the policy enforcement and the contract monitoring are active. During the execution, contract violations are checked for updating trust levels and policy enforcement is activated in order to guarantee that the application does not violate the security policy on the device. Indeed, the contract monitoring receives event signals from the executing code and keeps trace of the execution trace. When a signal arrives, its consistency with respect to the monitored contract is checked. If the contract is respected then its internal monitoring state is updated and the operation is allowed, and a good behaviour is logged (*i.e.*, contract respected). Otherwise, if a violation attempt happens, a security error occurs and a violation feedback is logged for the trust module. The policy enforcer is only in charge of following the execution of the application and whenever it attempts to violate the security policy of the device, the enforcement mechanism halts the execution in such a way that the security policy is satisfied.

MC Scenario The contract monitoring is performed. It works according to the following strategy: the contract monitoring receives event signals from the executing code. The execution trace is kept in memory. When a signal arrives, its consistency with respect to the monitored contract is checked. As in the previous case, if the contract is respected then its internal monitoring state is updated and the operation is allowed, and a good behaviour is logged (*i.e.*, contract respected). Otherwise, if a violation attempt happens, a security error occurs, and a bad feedback is trigged (*i.e.*, contract violation), and the system switches from contract monitoring to policy enforcement configuration in order to guarantee that the security policy is satisfied. Since an instance of the policy is always present, this operation does not imply a serious computational overhead.

Summing up, both execution scenarios check contract violations through the contract monitoring strategy described above and update providers' trust level according to the contract monitoring feedback.

A Prototype Implementation A prototype of the S×C×T application is implemented in Java Micro Edition (J2ME). It follows the architecture depicted in Figure 7.1.

Basically, the prototype consists of a suite of tools. The main components are: the *policy/contract matching verifier*, the *security checks in-liner*, the *trust manager system* (TMS) and the *policy decision point* (PDP).

The system is triggered whenever a new application package is downloaded on the platform before its actual installation. At this stage, the application package contains the MIDlet resources, *i.e.*, the files needed to install and execute the application, and its contract, *i.e.*, an XML-based specification of the MIDlet behaviour. Firstly, the system assigns a trust value to the MIDlet by retrieving from the TMS the trust measure associated to the MIDlet provider. In particular, the TMS is implemented as a local service that answers to special, authenticated requests. Then, the S×C×T framework proceeds according to the MIDlet trust level. If the MIDlet is considered to be untrusted, *i.e.*, the trust value is less than an predefined acceptance threshold, the MIDlet is instrumented with local checks (in-lining). Otherwise, two tasks run in parallel: the in-lining procedure and the policy/contract matching. The instrumentation process produces a modified version of the MIDlet in which security checks have been inserted inside the instructions flow

in order to wrap potentially dangerous operations. Note that the set of these operations contains (at least) all the actions the policy and the contract refer to. The second task compares the MIDlet contract with the platform policy through a simulation process. The result of this computation is a positive flag if the contract is policy compliant, *i.e.*, if the set of behaviours denoted by the contract is a subclass of those allowed by the policy. If it is not the case, a negative response is returned.

At this point the modified MIDlet is installed on the mobile device and can be actually executed. At execution time, the PDP applies the standard policy enforcement strategy to the MIDlets that are untrusted or have an illegal contract. Instead, it watches the trusted MIDlets having a valid contract, with respect to the security policy, by verifying that the actual behaviour complies with the declared contract. Well behaving MIDlets may then be slightly rewarded according to the trust model presented before. On the contrary, misbehaving MIDlets may receive a drastic trust penalty. If such a contract violation is detected, the PDP immediately switches to the policy enforcing configuration, eventually stopping the execution. The mixed PDP strategy, *i.e.*, contract monitoring and policy enforcement, is obtained by composing the MIDlet contract with the system policy. Since both of them use the same specification formalism, *i.e.*, the XML-based syntax, the two requirements can be composed provided that the policy is more general than the contract, *i.e.*, the contract policy matching has been successfully passed.

7.3 Access Control Policy Negotiation with Trust Levels

In the previous chapter (Chapter 6) a generic trust-meta-model is depicted, for dealing with the (possible) heterogeneity of the trust models adopted by different entities in the communication, *e.g.*, by different Networked Systems, Enablers, CONNECTors, and so on. Here we present an access control policy negotiation mechanism for allowing two NSs to state and enforce their access control policies to their services. These access control policies can also be enriched with trust values (for instance calculated through the accreditation process defined in Chapter 6). For sake of simplicity, in order to develop our negotiation mechanism, we assume that both the NSs agree on the same access control language and trust model for the credentials.

Indeed, in this framework we can express that a NS has an access control policy for allowing another NS to interact with it. In order to access the NS service, one NS has to present a given set of credentials that must match the access control policies of the other NS.

A policy negotiation phase arises if one NS does not present the appropriate access credentials to the other NS. In this case, it can start a negotiation phase where credentials are interactively requested, in a step by step procedure. The interesting part of the work we did in this area is the capability to do this negotiation strategy in an access control language where credentials stating access control rules and facts are enriched with trust measures.

We recall that two basic logical services can be used inside authorization systems. One is used to establish if a NS has presented credentials sufficient to have granted the access, the other is a way to help the client (the NS requesting the service of the other NS) to understand where are the missing credentials to gain the access.

The first logical approach useful in access control is represented by *deduction* [66, 76]: given a policy and a set of additional facts and events, the service finds out all consequences (actions or obligations) of the policy and the facts, *i.e.*, whether granting the request can be deduced from the policy and the current facts.

Current access control policy languages based on credentials (for both expressing facts and access policy rules) uses several foundational approaches. However, facts and access rules are not so crisp in the real complex world. For example, each piece of information could have a confidence value associated with it and representing a reliability estimation, or a fuzzy preference level or a cost to be taken in account.

The feedback final value, obtained by aggregating all the ground facts together, can be then used to improve the decision support system by basing on this preference level instead of a plain “yes or no” result (*e.g.*, see [25, 20]).

In this scenario, a credential could state that the referred entity is a “guards” or a “civilian guards ” with a probability of 80% because her/his identity of guard is based on what an accreditation authority asserts with a certain trust level/precision.

Here we are mainly interested to work with a credential-based language enriched with weights for expressing trust measures. In order to give full semantics to these family of languages, we use a weighted version of Datalog where the rules are enhanced with values taken from a proper c-semiring structure [18, 19], in order to model the preference/cost system. Datalog is a restricted form of logic programming with variables, predicates, and constants, but without function symbols. Facts and rules are represented as Horn clauses. Then, we use it as the basis to give declarative semantics to a Role-based Trust-management language according to the principles of RT_0 [70], and called here RT_0^W: the statements of RT_0^W are "soft", *i.e.*, have a related c-semiring value. A similar improvement can be accomplished also for RT_1 [70], *i.e.*, RT_0 extended with parametrized roles. Similar variations for RTML (Role-based Trust Management Language) family languages were defined and implemented by using different formal tools. There, an initial comparison (and integration) between rule-based trust management (RTML) and reputation-based trust systems has been performed and a preliminary (ad hoc) implementation RTML weighted presented in [28] for GRID systems. However, having a uniform semantics approach, as the weighted datalog we are advocating here, to model these languages could be very useful to provide a common understanding as well as a basis for systematic comparison and uniform implementation.

As it has been shown since trust is not necessarily crisp, soft Datalog could be used to give formal semantics to this languages for soft credentials. We have shown here an approach for RTML. However, it can be further extended to other credential based languages, whose semantics could be given in terms of soft Datalog. We thus provide a formal semantics for such languages that could also bring to a uniform implementation approach, as well as to a comparison among these languages w.r.t. to others by using existing results on Datalog and soft constraints. Our systematic approach to give weights to facts and rules, contributes also towards bridging the gap between "rule-based" trust management (*i.e.*, hard security mechanisms) and "reputation based" trust management [58] (*i.e.*, soft security mechanisms).

As previously mentioned, access authorization usually needs another reasoning service: abduction [66, 76]. Loosely speaking, abduction is deduction in reverse: given a policy and a request for access to (*e.g.*) NS services, it consists in finding the credentials/events that would grant access, *i.e.*, a (possibly minimal) set of facts that added to the policy would make the request a logical consequence. The intuition behind an interactive access control system is the following: *i)* initially a client (NS in our case) submits a set of credentials and a service (another NS) request then, *ii)* the server checks whether the request is granted by the access policy according to the client's set of credentials. If the check fails, *iii)* by using abductive reasoning the server finds a (minimal) solution set of (disclosable) missing credentials that unlocks the desired resource and *iv)* returns them to the client, so that *v)* he can provide them in the second round.

This is a single negotiation step. It is useful to establish if the NS needs further information to grant the access to the services. Clearly, this approach eases cooperation among NSs and it is the basis for further cooperation.

We manage trust-based access authorization with weighted credentials, i.e. credentials where the associated level of trust can be used to enrich the authorization process. The proposed framework can be used to improve logical reasoning processes in autonomic networks of nodes [66], since these services (i.e. deduction and abduction) help the self-managing characteristics of distributed computing resources: a node sometimes needs to independently produce the missing information (through an abduction process).

This is an initial work focussing on access control policy negotiation in the presence of trust values. The initial results are promising and we do plan to extend this to the trust models defined in the Chapter 6, where the two NS may present two different trust models (not based on the same credential language that in our case simplifies the treatment).

7.4 Application to the Terrorist Alert Scenario

Referring to the Terrorist Alert scenario described in Chapter 2 and in Deliverable D6.2, in this section we show how we apply our approach for dealing with security aspects. In particular we show how the S×C×T paradigm and the Access control negotiation mechanisms can be applied for guaranteeing security requirement in the scenario of the terrorist alert.

7.4.1 Security-by-Contract-with-Trust

Let us assume that the CONNECTor, once synthesized, is deployed in a decentralized way on both the NSs. Let us also assume that each NSs doe not trust the other, *i.e.*, the Policemen do not trust civilian guards and vice versa.

According to the scenario, Policemen ask for establishing a communication with guards in a particular area of the stadium in order to send them a photo of the terrorist they are looking for.

The security requirement of the CONNECTed system is the following one:

The photo can only be received by authorized devices.

For sake of simplicity, we describe our approach from the point of you of a policeman. Hence, for the police "authorized" means:

- they have a certain level of trust;
- they have to send an ACK;
- they expose a certificate for proving who and where they are.

The conjunction of these three requirements is the security policy requested by the police to the CONNECT infrastructure.

We assume that the CONNECTor is the one described in Figure 2.6. The CONNECTor is deployed part on the device of the policeman and part on the device of a guard. Both of them receive also its contract that describes its behaviour. Both NSs are provided of the S×C×T mechanism for enforcing policies and monitoring contract. The CONNECTor is assumed to be safe and trust by construction. By the way, since it is deployed and run on a possible malicious platform, both the NSs run the S×C×T for controlling that nobody modifies the CONNECTor making it unsafe.

According to the level of trust of the guards, the CONNECTor is directly executed and we switch in the MC scenario, or, the contract of the CONNECTor is compared with the security policy set on the policeman's device. Referring to our model if some violation occurs, the NS that detects it, sends a negative feedback to the CONNECT infrastructure, *i.e.*, to the trust Enabler. It is in charge to manage it and, in this case, it penalizes the NS by decreasing its level of trust.

7.4.2 Access Control Policy Negotiation with Trust

Let us suppose that also the guards, before receiving a message from someone, ask for certain assurances. The CONNECTor is in charge to guarantee that these requirements are satisfied. So before establishing a communication, the two NSs negotiate their access control requirements through the CONNECTor.

Indeed, let us suppose that the CONNECTor has sent the `eReq` message to the civilian guards. According to Figure 2.6, this message is the translation of the `selectArea` message sent to the Police for reaching only the civilian guards located in a certain area of the stadium.

When the guards receive this message, they want to be sure that they are speaking with the Police. The CONNECTor does not establish the communication yet but requires that Police sends as parameter of the `selectArea` message also a certificate of its identity.

If the certificate is accepted form the civilian guards, then the communication can be established. Thus, (part of) the CONNECTor is deployed on each NSs and the communication is established. According to this access control negotiation, the trust module of each component establishes the level of trust of the deployed CONNECTor. According to this, the S×C×T configuration is the scenario MC or EPMC.

7.5 Conclusions and Further Research Directions

In this second year of the project we have extended the analysis of our threat models for dealing with different deployment strategies of a CONNECTor. Furthermore, we proposed an integrated version of the Security-by-Contract-with-Trust paradigm in which we have integrated a trust model that is in charge of managing feedbacks obtained as result of a CONNECTor contract monitoring and enforcement policy. We also shown how and when this approach is able to guarantee security requirements in the proposed

Terrorist Alert case study. As future directions, we aim to extend our work for covering as far as possible all the threat models presented here. We also would like to extend our enforcement strategy for dealing with cryptography.

Moreover, we have taken an additional research direction covering access control policy negotiation. We presented a trust negotiation framework for quantitative notions of trust based on soft-constraints, which will be the basis for subsequent work on automated trust negotiation among NSs.

8 Conclusions and Future Work

This deliverable reports advances achieved in Y2 within WP5. This is a broad scope workpackage addressing non-functional properties of CONNECTed systems at synthesis time and at runtime.

We coined the term CONNECTability to refer within a unique context concerns for dependability, performance, security and trust. A formal meta-model of CONNECTability properties and of their associated metrics has been specified, so that metrics for quantitative properties of interest can be automatically instantiated by means of a model-driven editor. In the next year we will work towards: $i)$ further refining this meta-model in some parts, such as the EventType definition to define a formal way to specify application-dependent observable behavior by using event operators shared with the algebra defined in WP2. This will be the ground on which the integration among the functional and non-functional aspects will be built. We also need to include the Trust meta-model which is now under development; $ii)$ specifying M2M and M2C transformations that translate the properties models towards the specific analysis methodologies and the intrumentation performed by the monitoring infrastructure.

We have presented the Dependability&Performance analysis Enabler which cooperates with the Synthesis Enabler to verify whether the prescribed properties would be satisfied by the CONNECTor under synthesis. We have extensively described the two analysis engines currently used, one based on Möbius and the other based on Prism. The enhancements/extensions planned for the third year mainly consist in: $i)$ progressing in the implementation of the Enhancer module of the the Dependability&Performance Enabler, by taking into account in the analysis basic fault-tolerance mechanisms/patterns, among a set available to react to selected failure modes experienced by the CONNECTed system. This is a very important feedback to the Synthesis Enabler towards the synthesis of a dependable CONNECTor; $ii)$ progressing in completing the loop between the Dependability&Performance Enabler and the Synthesis Enabler so that Synthesis can profitably embed in the synthesis of the CONNECTor the indications coming from the analysis; $iii)$ progressing in the interactions between the Dependability&Performance Enabler and the Monitor Enabler, to refine the analysis along time using real data/events observed at run-time.

We have introduced incremental verification, which allows for refining the analysis after changes to the system, without having to redo it all from scratch. Currently, this technique only allows the changes in probability values. In Y3, we will investigate the cases where system structure can evolve, i.e., some transitions can be removed from the model, and new transitions can be added.

We also showed how, at runtime, the Dependability&Performance analysis Enabler can interact with the Monitoring Enabler to check that the assumptions on which the analysis is based remain valid. During Y3, we want also to improve communication and analysis of monitored data. The integration with the above mentioned metrics meta-model to express monitoring rule will provide a more powerful monitoring infrastructure. Another research issue will be the enrichment of the architecture with a module dedicated to model to model and model to code conversion to provide an heterogeneous infrastructure able to communicate with different rules engines.

We have defined a formal trust meta-model which can generically describe any given trust model, so that we can ensure trust interoperability among heterogeneous Networked Systems through the CONNECT Trust Enabler. Given the respective trust models provided by two NSs, we are able to infer composite trust models for them to be able to interact in a trusted way.

We have then presented a refined version of the Security-by-Contract-with-Trust, based on an accurate study of the security threat models, and have discussed how it verifies that the security contract and the required trust levels are satisfied, or otherwise how it enforces them at runtime. We have also introduced an approach to negotiate credential-based trust access levels. In the third year we will work on extending the SxCxT system by covering as far as possible all the threat models presented here. We also would like to extend our enforcement strategy for dealing also with cryptography.

Finally, the illustrated research has been instantiated in several prototype tools, which are also part of Deliverable D5.2. Namely they include:

- CPMM eCore & Editor for CONNECT properties;
- the DePer Analysis prototype, which instantiates the Dependability&Performance architecture;
- PRISM CONNECT Bundle, which is a prototype of the incremental verification technique;
- the SxCxT infrastructure;

- the GLIMPSE run-time monitoring infrastructure, which can interoperate with DePer.

In the associated Appendix-Prototypes document, we provide their list, along with essential information and the URL from which they can be downloaded.

Beyond the specific improvements planned for each approach, the keywords for Y3 activities will be integration and convergence, as we intend to further push the interaction of WP5 Enablers and the other CONNECT Enablers.

Bibliography

[1] Activemq: A complete message broker. http://activemq.apache.org.

[2] Drools fusion: Complex event processor. http://www.jboss.org/drools/drools-fusion.html.

[3] Servicemix: an open source esb. http://servicemix.apache.org/home.html.

[4] ReSIST: Resilience for Survivability in IST. Deliverable D33: Resilience-explicit computing. Technical report, 2008.

[5] A. Abdul-Rahman and S. Hailes. A distributed trust model. In *NSPW: New Security Paradigms Workshop*, pages 48–60, New York, USA, 1997. ACM Press.

[6] S. Ahamed, M. Monjur, and M. Islam. CCTB: Context correlation for trust bootstrapping in pervasive environment. In *2008 IET 4th International Conference on Intelligent Environments*, pages 1–8, 2008.

[7] S. Amundsen and F. Eliassen. Combined Resource and Context Model for QoS-Aware Mobile Middleware. In *Proceedings of 19th International Conference on Architecture of Computing Systems (ARCS 2006)*, pages 84–98. 2006.

[8] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[9] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying continuous time Markov chains. In R. Alur and T. Henzinger, editors, *Proc. 8th International Conference on Computer Aided Verification (CAV'96)*, volume 1102 of *LNCS*, pages 269–276. Springer, 1996.

[10] C. Baier, J.-P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In J. Baeten and S. Mauw, editors, *Proc. 10th International Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *LNCS*, pages 146–161. Springer, 1999.

[11] L. Baresi, C. Ghezzi, and E. Di Nitto. Toward open-world software: issues and challenges. *Computer*, 39(10), 2006.

[12] A. L. Baroni, C. Calero, M. Piattini, and O. B. E. Abreu. A Formal Definition for ObjectRelational Database Metrics. In *Proceedings of the 7th International Conference on Enterprise Information System*, 2005.

[13] S. Ben Mokhtar, N. Georgantas, and V. Issarny. COCOA: COnversation-based service COmposition in pervAsive computing environments with QoS support. *J. Syst. Softw.*, 80:1941–1955, December 2007.

[14] A. Bertolino, A. Calabrò, F. Di Giandomenico, M. Martinucci, and P. Masci. Automated refinement of dependability analysis through monitoring in dynamically connected systems. In *Proc. IEEE International Symposium on Autonomous Decentralized Systems*, Tokyo, Japan, March 2011, to appear.

[15] A. Bertolino, A. Calabrò, F. Lonetti, and A. Sabetta. Glimpse: A generic and flexible monitoring infrastructure. Technical report, ISTI-CNR, 2010. 2010-TR-024.

[16] A. Bertolino, F. Di Giandomenico, A. Di Marco, P. M. Masci, and A. Sabetta. Qos metrics in dynamic, evolving and heterogeneous connected systems. In *8th International Workshop On Dynamic Analysis (WODA 2010)*, Trento, Italy, 2010.

[17] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proc. FSTTCS'95*, volume 1026 of *LNCS*. Springer, 1995.

[18] S. Bistarelli. *Semirings for Soft Constraint Solving and Programming*, volume 2962 of *LNCS*. Springer, 2004.

[19] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *J. ACM*, 44(2):201–236, 1997.

[20] P. Bonatti, C. Duma, D. Olmedilla, and N. Shahmehri. An integration of reputation-based and policy-based trust management. In *Semantic Web Policy Workshop*, 2005.

[21] U. Buy and G. Singal. Toward efficient algorithms for generating compact petri nets from labeled transition systems. In *COMPSAC '02: Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment*, pages 717–722, Washington, DC, USA, 2002. IEEE Computer Society.

[22] R. Calinescu and M. Kwiatkowska. Using quantitative analysis to implement autonomic it systems. In *Proc. 31st International Conference on Software Engineering*, pages 100–110, May 2009.

[23] J. Carmona, J. Cortadella, and M. Kishinevsky. Genet: A tool for the synthesis and mining of petri nets. In *ACSD '09*, pages 181–185, Washington, DC, USA, 2009. IEEE Computer Society.

[24] A. Castrucci, F. Martinelli, P. Mori, and F. Roperti. Enhancing java me security support with resource usage monitoring. In *ICICS*, pages 256–266, 2008.

[25] S. Chakraborty and I. Ray. Trustbac: integrating trust relationships into the rbac model for access control in open systems. In *SACMAT '06: Proc. of Access control models and technologies*, pages 49–58. ACM, 2006.

[26] F. Ciesinski, C. Baier, M. Größer, and J. Klein. Reduction techniques for model checking Markov decision processes. In *Proc. QEST'08*, pages 45–54. IEEE CS Press, 2008.

[27] G. Clark, T. Courtney, D. Daly, D. D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. G. Webster. The Möbius modeling tool. In *9th Int. Workshop on Petri Nets and Performance Models*, pages 241–250, Aachen, Germany, September 2001. IEEE Computer Society Press.

[28] M. Colombo, F. Martinelli, P. Mori, M. Petrocchi, and A. Vaccarelli. Fine grained access control with trust and reputation management for globus. In *OTM Conferences (2)*, pages 1505–1515, 2007.

[29] CONNECT Consortium. Deliverable 2.1 – Capturing functional and non-functional connector behaviours, 2010.

[30] CONNECT Consortium. Deliverable 3.1 – Modeling of Application- and Middleware-layer Interaction Protocols, 2010.

[31] CONNECT Consortium. Deliverable 4.1 – Establishing basis for learning algorithms, 2010.

[32] CONNECT Consortium. Deliverable 5.1 – Conceptual Models for Assessment & Assurance of Dependability, Security and Privacy in the Eternal CONNECTed World, 2010.

[33] CONNECT Consortium. Deliverable 1.2 – Intermediate CONNECT Architecture, 2011.

[34] CONNECT Consortium. Deliverable 4.2 – Further development of learning techniques, 2011.

[35] CONNECT Consortium. Deliverable 6.1 – Experiment scenarios, prototypes and report Iteration 1, 2011.

[36] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving petri nets from finite transition systems. *IEEE Trans. Comput.*, 47(8):859–882, 1998.

[37] G. Costa, N. Dragoni, V. Issarny, A. Lazouski, F. Martinelli, F. Massacci, I. Matteucci, and R. Saadi. Extending security-by-contract with quantitative trust on mobile devices. *Journal of Wireless Mobile Networks, Ubiquitous Computing and Dependable Applications (JOWUA)*, 1(4):75–91, December 2010 2011. ISSN (print): 2093-5374, ISSN (on-line): 2093-5382.

[38] G. Costa, N. Dragoni, A. Lazouski, F. Martinelli, F. Massacci, and I. Matteucci. Extending security-by-contract with quantitative trust on mobile devices. In *In Proceeding of CISIS 2010, The Fourth International Conference on Complex, Intelligent and Software Intensive Systems*, pages 872–877, Krakow, Poland, 15-18 February. IEEE Computer Society.

[39] G. Costa, F. Martinelli, P. Mori, C. Schaefer, and T. Walter. Runtime monitoring for next generation java me platform. *Computers & Security*, July 2009.

[40] G. Costa and I. Matteucci. Enforcing private policy via security-by-contract. *Identity and Privacy Management. Special issue of the journal UPGRADE*, 2010.

[41] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4), 1995.

[42] G. Cugola and A. Margara. TESLA: a formally defined event specification language. In *Proceedings of 4th ACM International Conference On Distributed Event-Based Systems (DEBS)*, pages 50–61, 2010.

[43] L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1997.

[44] G. Despotou, D. S. Kolovos, R. F. Paige, F. A. Polack, and T. P. Kelly. Towards a meta-model for dependability cases. In *OMG 1st Software Assurance Workshop, Washington, USA*, March 2007.

[45] F. Di Giandomenico, M. Kwiatkowska, M. Martinucci, P. Masci, and H. Qu. Dependability analysis and verification for connected systems. In T. Margaria and B. Steffen, editors, *Proc. ISOLA 2010 - Leveraging Applications of Formal Methods, Verification, and Validation*, volume 6416 of *LNCS*, pages 263 – 277. Springer, 2010.

[46] Eclipse Modeling Project. http://www.eclipse.org/modeling/.

[47] A. Ehrenfeucht and G. Rozenberg. Partial (set) 2-structures. part i: basic notions and the representation problem. *Acta Inf.*, 27(4):315–342, 1990.

[48] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed nist standard for role-based access control. *ACM Transactions on Information and System Security*, 4(3):224–274, 2001.

[49] C. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligences*, 19(1):17–37, 1982.

[50] S. Frolund and J. Koistinen. Quality-of-Service Specification in Distributed Object Systems. *Distributed Systems Engineering Journal*, 5:179–202, 1998.

[51] K. Fullam, T. Klos, G. Muller, J. Sabater, A. Schlosser, Z. Topol, K. Barber, J. Rosenschein, L. Vercouter, and M. Voss. A specification of the Agent Reputation and Trust (ART) testbed. In *Conference on Autonomous agents and multiagent systems*, pages 512–518. ACM, 2005.

[52] T. Grandison and M. Sloman. A survey of trust in internet applications. *Communications Surveys & Tutorials, IEEE*, 3(4):2–16, 2009.

[53] R. Gulati and J. B. Dugan. A modular approach for analyzing static and dynamic fault trees. In *Annual Reliability and Maintainability Symposium*, pages 57–63. IEEE Computer Society Press, 1997.

[54] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[55] M. Haque and S. Ahamed. An omnipresent formal trust model (FTM) for pervasive computing environment. In *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International*, volume 1, 2007.

[56] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. TACAS'06*, volume 3920 of *LNCS*. Springer, 2006.

[57] M. Huhn and A. Zechner. Analysing dependability case arguments using quality models. In *Proceedings of 28th International Conference on Computer Safety, Reliability, and Security (SAFECOMP 2009)*, pages 118–131. 2009.

[58] A. Jøsang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. *Decis. Support Syst.*, 43(2):618–644, 2007.

[59] A. Jøsang and S. Pope. Semantic constraints for trust transitivity. In *APCCM: 2nd Asia-Pacific conference on Conceptual modelling*, pages 59–68, Newcastle, New South Wales, Australia, 2005. Australian Computer Society, Inc.

[60] A. Jsang and R. Ismail. The beta reputation system. In *Proceedings of the 15th Bled Electronic Commerce Conference*, pages 17–19, 2002.

[61] S. Kaffille and G. Wirtz. Engineering Autonomous Trust-Management Requirements for Software Agents: Requirements and Concepts. *Innovations and Advances in Computer Sciences and Engineering*, pages 483–489, 2010.

[62] H. Kautz, B. Selman, and M. Shah. Referral Web: combining social networks and collaborative filtering. *Communications of the ACM*, 40(3):63–65, 1997.

[63] R. Keller. Formal verification of parallel programs. *Commun. ACM*, 19(7):371–384, 1976.

[64] T. P. Kelly. Arguing Safety - A Systematic Approach to Managing Safety Cases, PhD Thesis, 1998.

[65] Y. Kim and K. Doh. Trust Type based Semantic Web Services Assessment and Selection. *Proceedings of ICACT, IEEE Computer*, pages 2048–2053, 2008.

[66] H. Koshutanski and F. Massacci. A negotiation scheme for access rights establishment in autonomic communication. *J. Network Syst. Manage.*, 15(1):117–136, 2007.

[67] J. Laprie. Dependable computing and fault tolerance: concepts and terminology. In *Fault-Tolerant Computing, 1995, ' Highlights from Twenty-Five Years'., Twenty-Fifth International Symposium on*, pages 2+, 1995.

[68] Lars Vogel. Eclipse Modeling Framework Tutorial. http://www.vogella.de/articles/EclipseEMF/article.html, January 2011.

[69] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, 2(1):1–15, 1994.

[70] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust-management framework. In *SP '02: Proc. of Security and Privacy*, page 114. IEEE Computer Society, 2002.

[71] N. B. Mabrouk, N. Georgantas, and V. Issarny. A semantic end-to-end QoS model for dynamic service oriented environments. In *Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems*, PESOS '09, pages 34–41, 2009.

[72] J. Magee and J. Kramer. *Concurrency: state models & Java programs*. John Wiley & Sons, New York, NY, USA, 2006.

[73] M. Mansouri-Samani and M. Sloman. Gem - a generalised event monitoring language for distributed systems. *IEE/IOP/BCS Distributed Systems Engineering Journal*, 4, 1997.

[74] S. Marsh. *Formalising Trust as a Computational Concept*. PhD thesis, University of Stirling, Scotland, 1994.

[75] P. Masci, M. Martinucci, and F. Di Giandomenico. Towards automated dependability analysis of dynamically connected systems. In *Proc. IEEE International Symposium on Autonomous Decentralized Systems*, Tokyo, Japan, March 2011, to appear.

[76] T. Menzies. Applications of abduction: knowledge-level modeling. *Int. J. Hum.-Comput. Stud.*, 45(3):305–335, 1996.

[77] M. Monperrus, J.-M. Jézéquel, B. Baudry, J. Champeau, and B. Hoeltzener. Model-driven generative development of measurement software. *Software and Systems Modeling (SoSyM)*, tba, 2010.

[78] A. Movaghar and J. F. Meyer. Performability modeling with stochastic activity networks. In *1984 Real-Time Systems Symposium*, pages 215–224, Austin, TX, December 1984. IEEE Computer Society Press.

[79] L. Mui, M. Mohtashemi, C. Ang, P. Szolovits, and A. Halberstadt. Ratings in distributed systems: A bayesian approach. In *Proceedings of the Workshop on Information Technologies and Systems (WITS)*, pages 1–7. Citeseer, 2001.

[80] P. Nurmi. A bayesian framework for online reputation systems. In *Telecommunications, 2006. AICT-ICIW '06. International Conference on Internet and Web Applications and Services/Advanced International Conference on*, pages 121–121, Feb. 2006.

[81] OASIS. Quality Model for Web Services (WSQM). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsqm#overview, September 2005.

[82] OMG. UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE). http://www.omg.org/omgmarte/Specification.htm/.

[83] OMG. UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms Specification, Version 1.1. http://www.omg.org/spec/QFTP/1.1/, April 2008.

[84] S. Paradesi, P. Doshi, and S. Swaika. Integrating Behavioral Trust in Web Service Compositions. In *Proceedings of the 2009 IEEE International Conference on Web Services*, pages 453–460. IEEE Computer Society, 2009.

[85] A. Pataricza and F. Gyr. Towards unified dependability modeling and analysis. In *Proceedings of ARCS Workshops*, pages 113–122, 2004.

[86] A. Pnueli. The temporal logic of programs. In *Proc. 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Computer Society Press, 1977.

[87] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, 1994.

[88] D. Quercia, S. Hailes, and L. Capra. TRULLO-local trust bootstrapping for ubiquitous devices. *Proc. of IEEE Mobiquitous*, 2007.

[89] H. Raffelt, B. Steffen, and T. Berg. Learnlib: a library for automata learning and experimentation. In *FMICS '05: the 10th intl. workshop on Formal methods for industrial critical systems*, pages 62–71, New York, NY, USA, 2005. ACM.

[90] A. Rahman and S. Hailes. Supporting trust in virtual communities. *IEEE Hawaii International Conference on System Sciences*, page 6007, 2000.

[91] R. ReiBing. Towards a model for object-oriented design measurement. In *Proceedings of ECOOP Workshop on Quantative Approaches in Object-Oriented Software Engineering*, pages 71–84, 2001.

[92] R. Saadi, J. M. Pierson, and L. Brunie. Establishing trust beliefs based on a uniform disposition to trust. In *ACM SAC: Trust, Reputation, Evidence and other Collaboration Know-how track*. ACM Press, 2010.

[93] W. H. Sanders and J. F. Meyer. Stochastic Activity Networks: formal definitions and concepts. pages 315–343, 2002.

[94] R. Spalazzese and P. Inverardi. Mediating connector patterns for components interoperability. In *Proceedings of the European Conference on Software Architecture (ECSA2010)*, 2010.

[95] G. Suryanarayana, J. Erenkrantz, S. Hendrickson, and R. Taylor. PACE: an architectural style for trust management in decentralized applications. In *Software Architecture, 2004. WICSA 2004.*, pages 221–230. IEEE, 2004.

[96] G. Suryanarayana and R. Taylor. SIFT: A Simulation Framework for Analyzing Decentralized Reputation-based Trust Models. *Technical Report UCI-ISR-07-5*, 2007.

[97] G. F. Tondello and F. Siqueira. The QoS-MO ontology for semantic QoS modeling. In *Proceedings of the ACM symposium on Applied computing*, SAC '08, pages 2336–2340, 2008.

[98] L. Vercouter, S. Casare, J. Sichman, and A. Brandao. An experience on reputation models interoperability based on a functional ontology. In *Proceedings of 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, 2007.

[99] W3C. OWL-S: Semantic Markup for Web Services. http://www.w3.org/Submission/OWL-S/, November 2004.

[100] G. Wiederhold. Mediators in the architecture of future information systems. *Computer*, 25(3):38–49, 2002.

[101] R. Zhou, K. Hwang, and M. Cai. Gossiptrust for fast reputation aggregation in peer-to-peer networks. *IEEE Transactions on Knowledge and Data Engineering*, pages 1282–1295, 2008.

[102] P. R. Zimmermann. *The official PGP user's guide*. MIT Press, Cambridge, MA, USA, 1995.