Consolidated dependability framework

Antonia Bertolino, Antonello Calabro, Silvano Chiaradonna, Gabriele Costa, Felicita Di Giandomenico, Antinisca Di Marco, Mario Fusani, Valerie Issarny, Massimiliano Itria, Francesca Lonetti, et al.

CONNECT

Emergent Connectors for

Eternal Software Intensive Networked Systems

ICT FET IP Project

Deliverable D5.3

Consolidated dependability framework

SEVENTH FRAMEWORK PROGRAMME

http://www.connect-forever.eu

Inria · DOCOMO Euro-Labs · Amalencic · 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	:	D5.3
Title of Deliverable	:	Consolidated dependability framework
Nature of Deliverable	:	Report, Prototype
Dissemination Level	:	PU
Internal Version Number	:	1.0
Contractual Delivery Date	:	31 January 2012
Actual Delivery Date	:	16 February 2012
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, Valerie Issarny, Massimiliano Itria, Francesca Lonetti, Marta Kwiatkowska, Eda Marchetti, Fabio Martinelli, Paolo Masci, Ilaria Matteucci, Charles Morisset, Nicola Nostro, Hongyang Qu, Rachid Saadi, Anna Vaccarelli, Luca Wiegand
Reviewer(s)	:	Sofia Cassel, Guglielmo De Angelis, Antinisca Di Marco, Valerie Issarny

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, which we refer to by the one inclusive term "*CONNECT*ability", must be fulfilled. In Deliverable D5.1 we conceived the conceptual models at the foundation of* CONNECT*ability. In D5.2 we then presented a first version of the approaches and of their respective enablers that we developed for assuring* CONNECT*ability both at synthesis time and at run-time. In this deliverables, we present the advancements and contributions achieved in the third year, which include:*

- *a refinement of the* CONNECT *Property Meta-Model, with a preliminary implementation of a Model-to-Code translator;*
- *an enhanced implementation of the Dependability&Performance analysis Enabler, supporting stochastic verification and state-based analysis, that is enriched with mechanisms for providing feedback to the Synthesis enabler based on monitor's run-time observations;*
- *a fully running version of the Security Enabler, following the Security-by-Contract-with-Trust methodology, for the monitoring and enforcement of* CONNECT *related security policies;*
- *a complete (XML) definition of the Trust Model Description Language, an editor and the corresponding implementation of supporting tools to be integrated into the Trust Management Enabler.*

Examples of the properties on which the above contributions are illustrated are excerpted from the Terrorist Alert and from the GMES CONNECT *scenarios that are developed in Work Package 6. In the accompanying Prototype appendix D5.3P, references to the developed software tools and Enablers implementations are provided.*

Keyword List

CONNECT*ability,* CONNECT*ed System,* CONNECT*or, Dependability, Enabler, Monitoring, Networked System, Non-functional Properties, Performance, Property Meta-model, 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	All
0.4	Revised complete draft for internal review	All
0.5	Minor corrections	A. Bertolino
0.6	Revision after internal review by Issarny	All
1.0	Final version	A. Bertolino

Document Review

Date	Version	Reviewer	Comment
25/01/2012	V 0.5	S. Cassel - UU	Complete review with several specific comments
27/01/2012	V 0.5	A. Di Marco - Univaq	Review throughout with several specific comments, except Chapter 2
31/01/2012	V 0.5	G. De Angelis - CNR	Detailed review of Chapter 2
06/02/2011	V 0.6	V. Issarny - Inria	Make individual contributions more “Connected”, plus many detailed comments

Table of Contents

List of Figures

List of Tables

1 CONNECTability framework: Status and Perspectives

This deliverable reports about progress made in the third year of CONNECT within Workpackage 5 (WP5) on "Dependability Assurance". WP5 encompasses dependability and performance analysis and verification, as well as security, privacy and trust management within the CONNECTed world. We recall from Deliverable D5.2 [28] that we introduced the new ad hoc term CONNECTability to refer altogether to all non-functional attributes of concern for CONNECT. Precisely, CONNECTability refers to the ability of CONNECT enablers to provide the CONNECTed system with the intended properties of Dependability, Performance, Security and Trust in justifiable way.

As planned in the project Description of Work, Y3 has been devoted to refining and consolidating the CONNECTability Enablers previously introduced in prototype form in Deliverable D5.2 [28]. In this chapter we provide a short summary of Y3 highlights and main results about CONNECTability, which will be then detailed in the next chapters. In particular, we first recall briefly the WP aim and tasks (Section 1.1) and the prototype enablers developed in Deliverable D5.2 (Section 1.3). We then overview and answer comments received at the second year review (Section 1.4). Hence we hint at refinements and main results produced in the current reporting period (Section 1.5), and give a roadmap of this deliverable (Section 1.6).

1.1 The Role of Workpackage WP5

For completeness, we summarize here WP5 tasks and objectives, as already reported in earlier WP5 deliverables.

The CONNECT project aims at overcoming the interoperability barriers among heterogenous communication technologies by synthesizing suitable ad hoc CONNECTors between Networked Systems (NSs) that 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. In such a vision, the role of WP5 is to ensure that the obtained CONNECTed system can ensure to function under the required non-functional properties.

To achieve this, 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 aims at developing 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 aims at adapting and extending existing techniques for security-by-contract checking and enforcement.

Task 5.4. Distributed trust management: this aims at developing a unifying theory for trust and a corresponding reputation scheme.

1.2 Summary of D5.1: Conceptual Background

As the objectives of this WP are quite broad, in the first year an intense study and conceptual modelling activity has been carried out, for combining the differing backgrounds, terminologies and methods behind each task. The Y1 achievements reported in Deliverable D5.1 [26] included:

- A conceptual framework for expressing CONNECT-relevant metrics for the properties of interest [18]: 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 Summary of D5.2: The CONNECTability Enablers

In Deliverable D5.2 [28] we already introduced a set of prototypes developed within each of the above listed tasks.

CPMM Concerning Task 5.1, we have defined the CONNECT property meta-model (CPMM) that supports a model-driven approach to the specification of CONNECTability properties. CPMM is used as the exchange language among CONNECT Enablers to communicate and manage non-functional properties: thus properties defined according to such meta-model can be used, for example, as input for the dependability&performance Enabler to verify specified CONNECTability properties, or as instrumentation for the monitoring Enabler to generate suitable probes to monitor the specified properties on the CONNECTors, and so on.

In D5.2 [28] we introduced an advanced version of CPMM with examples of how specific CONNECTability properties of interest could be instantiated from the defined meta-model, and also provided a prototype version of the CPMM eCore model equipped with a dedicated editor (as an Eclipse Plugin) for deriving from CPMM instances of property models.

DEPER Concerning Task 5.2, we defined the architecture of the Dependability&Performance (DEPER) Enabler which cooperates before deployment with the Synthesis Enabler to verify whether the dependability and performance requirements requested by the NSs can be satisfied by a CONNECTor being synthesised, and at run-time with the Monitoring Enabler to check that the assumptions on which the analysis is based remain valid. DEPER supports both stochastic model checking and state-based stochastic evaluation approaches, using two analysis engines based on Möbius and on Prism, respectively. Preliminary prototypes for the two approaches have been developed and applied to the "Terrorist Alert" scenario (this is developed in WP6, please see Deliverable D6.2 [29]).

We have also introduced incremental verification, which allows for refining the analysis after changes to the system, without having to redo it all from scratch. In Y2 version, this technique only allowed the changes in probability values associated with transitions.

SxCxT With regard to Task 5.3, we presented the Security-by-Contract-with-Trust (SxCxT) infrastructure, which refined the existing security-by-contract approach through a detailed study of the CONNECT-specific security threat models, and showed how it can verify that the security contract and the required trust levels are satisfied, or otherwise how it can enforce them at run-time. We have also introduced an approach to negotiate credential-based trust access levels.

Trust Management Concerning Task 5.4, this has been partially covered by the SxCxT approach above outlined. In addition, we have introduced a generic trust meta-model as a basis to express and to compose a wide range of trust models, so that heterogeneous trust management systems belonging to different NSs can interoperate transparently. To this aim, novel mediation algorithms have been developed to overcome the heterogeneity between the trust metrics, relations and operations associated with the composed trust models. We illustrated the proposed trust meta-model under development on the Terrorist Alert scenario, by mediating and composing two heterogeneous trust models (i.e., Guard and Police).

1.4 Y2 Review Comments

Reviewers were satisfied with the progress reported at Y2 review. In particular they appreciated our work in defining a property meta-model for specifying CONNECT relevant non-functional properties and metrics. They also acknowledged the proposed architecture for the DEPER Enabler and its integration with the GLIMPSE monitor for run-time verification of dependability and performance properties, which was demonstrated at the review.

For future work, they recommended that the overall project should "look at the relation between connector adaption and connector synthesis". This issue does not concern specifically WP5 activity, but, as explicitly added in the review report, it is reported about WP5 as "it deals with monitoring the non-functional elements of the environment such as performance that drive adaptation".

In the reporting period we have worked actively at integrating CONNECTability concerns within the overall CONNECT architecture, so that not only CONNECTor synthesis can be adapted in synergy with run-time monitoring, as indicated in the review, but also other enablers, such as learning or security, can dynamically interact with the monitoring infrastructure and adapt to run-time observed behaviour of the CONNECTed system. The overall refined CONNECT architecture depicting the integration of CONNECTability enablers is reported in Deliverable D1.3 [30], whereas the detail of the workflow concerning each specific enabler is described in the following Chapters 3, 4 and 5. More notably, as described extensively in Chapter 3, the Updater module within DEPER interacts with the GLIMPSE monitor to get at run-time updated parameter estimations; then we have embedded within DEPER (precisely into the Enhancer module for Möbius-based analysis and into the Repairer module for Prism-based analysis) algorithms for adapting the assessment at run-time. The work on the complete integration between the synthesis enabler, DEPER and GLIMPSE to support dynamic adaptation based on run-time behaviour has also started and is still ongoing; it will reported in the next deliverable, however a preliminary description is provided in [15].

1.5 Y3 Results and Achieved Improvements

During the reported year we have made several improvements on all CONNECTability enablers.

CPMM For what concerns the CONNECT Property Meta-Model (CPMM) already defined during the second year of the CONNECT project and presented in D5.2, several improvements have been embedded into the meta-model to better specify properties and complex events. Specifically, the main CPMM improvement is about the EventType specification (as detailed in the following Section 2.2). While in the previous version of the meta-model this specification was simply defined by means of a label or string, now we provide a formal specification for the complex events and design several event composition operators by taking into account those of two existing event specification languages that are GEM [54] and Drools Fusion [1]. The update also takes into account feedback achieved by disseminating the CPMM to focused events [16, 40]. Moreover, in this year a first version of the Model-To-Code, Model2Code in the following, transformation has been defined that translates models conforming to CPMM (or part of them) to Drools rules, which are used to configure the GLIMPSE monitoring infrastructure. The aim of such transformation is to allow for the dynamic configuration of GLIMPSE whenever a new property to be monitored is specified and introduced in CONNECT. This transformation has been implemented using the Acceleo technology.

DEPER We have augmented several components of the DEPER enabler architecture. The enhancements concern the functioning of DEPER both at pre-deployment and at run-tim. In particular, on the one

side we support the pre-deployment adaptation of CONNECTor design to improve dependability and performance requirements, through the analysis of variants of the CONNECTed system model reinforced with dependability mechanisms patterns, to overcome deficiencies of the CONNECTor specification (Enhancer module). We also proposed an approach to construct a new CONNECTor that differs from the previous CONNECTor on the values of adjustable parameters. On the other side, we also support the run-time adaptation of the off-line analysis performed at pre-deployment time to cope with changes in, or inaccurate estimates of, model parameters, through processing of incremental accumulation of real observations obtained from the Monitor enabler (Updater module and Repair module). We have also extended the on-line verification technique presented in D5.2 to use symbolic implementations, based on binary decision diagrams (BDDs) and extensions such as multi-terminal BDDs (MTBDDs), by proposing a novel hybrid adaption of the Tarjan algorithm that combines symbolic (BDD) and explicit-state data structures. The results of Y3 improvements to DEPER have been published in [14, 56, 39, 52].

SxCxT We have partly implemented the SxCxT framework by focusing on security aspects, through code instrumentation in particular, as we further detail in Chapter 4. By refining and adapting the previous prototype (which was tailored for mobile code only), we now provide a first implementation of the Security Enabler, which can instrument the concrete CONNECTor in such a way that the communications between two NSs always respect the individual NS policies; we have illustrated this implementation on demonstrative examples from the GMES scenario. Relevant results have been published in [13, 35, 33].

Trust Management Finally, concerning the Trust Manager, we have enhanced the TMDL (Trust Model Description Language) language and developed a complete XML representation and provided a corresponding enabler supporting the composition of any given trust management systems according to given mapping rules. We have also developed a TMDL editor that guides developers to create a valid and correct TMDL description which can serve to automatically generate the JAVA code of the corresponding trust management system. These results have been published in [66].

1.6 This Deliverable

The rest of this document includes the following chapters. Chapter 2 is devoted to the CONNECT Property Meta-Model: we report in detail the performed refinements and develop examples with reference to the Terrorist Alert scenario. Chapter 3 reports the refinements to the Dependability&Performance Analysis Enabler, considering both a stochastic model-based analysis and stochastic model-checking, incorporating incremental verifications. Chapter 4 describes the implementation of the security enforcer, and Chapter 5 describes the TMDL implementation and related tools. Each chapter illustrates, in self-contained way, the achieved results on examples taken from the CONNECT scenarios developed in WP6. In particular, Chapters 2 and 3 refer to the Terrorist Alert scenario introduced in D6.2 [29], which had also been used in D5.2, whereas Chapters 4 and 5 refer to the GMES scenario preliminarily released in D6.3 [32]. It was our intention to have a general integrated demonstration of the CONNECTability framework on a same scenario, this has been however deferred to the next deliverable, as the targetted scenario, i.e. GMES, is still undergoing further development at the time this document is written. Finally, conclusions and discussion of future work are included in Chapter 6.

2 The CONNECT Property Meta-Model

This chapter reports on the current status of the CONNECT Property Meta-Model (CPMM) defined during the second year of the CONNECT project and presented in D5.2 [28], and on the Model-To-Code (Model2Code) transformation that translates models conforming to CPMM to Drools rules [1], which are used to configure the GLIMPSE monitoring infrastructure (GLIMPSE has been presented in Deliverable D4.2 [27]). The aim of such transformation is to allow for the dynamic configuration of the input to GLIMPSE whenever a new property to be monitored is specified and introduced in CONNECT.

More specifically, during this third year, several improvements have been embedded into the meta-model to better specify properties and complex events. Moreover, a first version of the Model2Code transformation has been implemented using Acceleo [5], to automatically convert the CONNECT metrics and properties models into a concrete monitoring setup. The output of this transformation is represented by Drools Fusion rules processed by the complex event processor component of GLIMPSE.

The chapter proceeds as follows: Section 2.1 briefly recalls the main concepts of CPMM (a full description is in D5.2 [28]); then Section 2.2 reports on the CPMM improvements made in the third year of the project, that mainly refer to *Property* (Section 2.2.1) and *EventType* (Section 2.2.2) specifications. Section 2.3 reports on the Acceleo Model2Code transformation. Section 2.4 reports the coverage property for the Terrorist Alert scenario, the corresponding CPMM models and the Drools Fusion rules generated by the defined transformation for the *coverage* property. Finally, Section 2.5 overviews main related work and Section 2.6 concludes the chapter.

2.1 CONNECT Property Meta-Model Overview

We recall that the CPMM defines elements and types to specify prescriptive (required) and descriptive (owned) quantitative and qualitative properties that CONNECT actors may expose or must provide, respectively. The properties specified from this language might refer to the CONNECTed or the Networked System; they can serve as reference model requirements for the CONNECTor synthesis or as events to be observed by the monitoring Enabler during CONNECTor execution. Besides, the specified properties can also describe the characteristics owned by the Enablers.

The key concepts of this meta-model are: Property, MetricsTemplate, Metrics, EventSet, and EventType. We separate the property definition from the application domain and its specific ontology. The ontology is linked to the Property meta-model via the EventType entity that models a generic event type where the terms of the application-domain ontology will be used. Figure 2.1 sketches an overview of the CPMM as it has been improved and refined in this third year.

The editor associated to CPMM contains the information of the defined models and allows to create new model instances of the Property, Metrics, MetricsTemplate, EventType and EventSet meta-models.

CPMM has been devised by taking into account the following features:

- **comprehensiveness**: CPMM allows for the specification of CONNECT properties that span over dependability, performance, security and trust [1]. Indeed, it is an instrument to specify existing as well as novel types of properties and metrics, providing the experts with a tool able to extend the set of specified properties and metrics with new emerging ones, if necessary.

- **flexibility**: we distinguish in CPMM the Metrics and MetricsTemplate concepts. 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) (e.g., the latency can be modeled as the difference of two time-stamps or as a duration, and both definitions are always valid whatever the target system is); this permits the expert to define once the generic MetricsTemplate and then the expert and not expert to use it several times for all future aims.

[1] However, some CPMM parts need to be refined to address trust, specifically the ActionBaseExpression introduced to model trust at the moment is not completely defined.

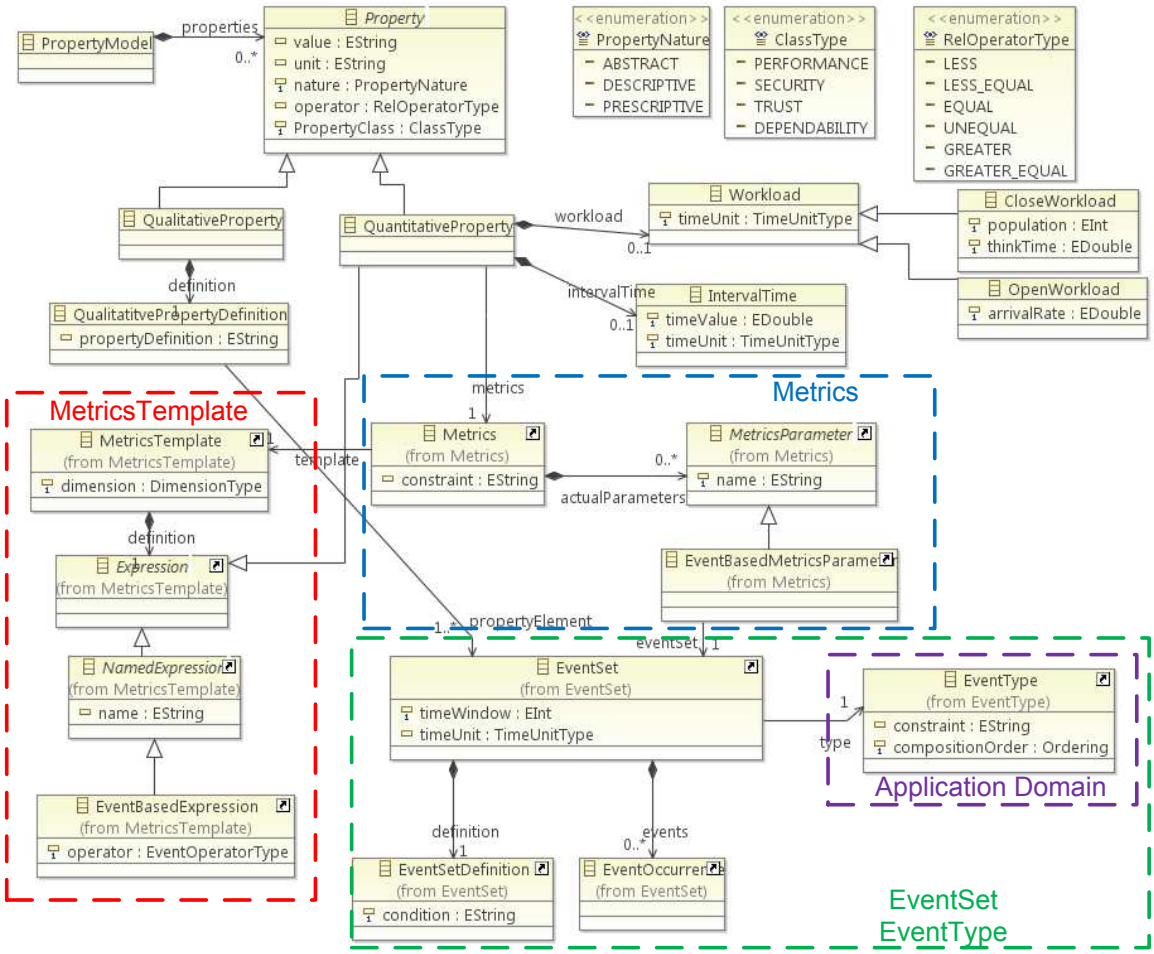


Figure 2.1: Key Concepts of the CONNECT Property Meta-Model

- **support for automated Model2Code transformations**: on top of the CPMM meta-model we defined automated procedures (in form of Model2Code transformations) that, from the CPMM models, instrument the GLIMPSE monitor for run-time verification of CONNECT properties.

In the proposed model-driven approach to monitoring, the properties to be monitored (either qualitative or quantitative) are specified according to a meta-model which is independent from the domain of application. Using this approach, and leveraging an underlying generic monitoring infrastructure, we can thus separate the problem of defining properties and metrics of interest in a dedicated domain-specific language, from the problem of converting these specifications into a concrete monitoring setup, which is done automatically in our approach. Model-driven specification of non-functional properties is not a new idea, on the contrary (as we report later in the Section 2.5) it is now a well established principle, which has been developed in several contexts. The contribution of our work in CONNECT stays in: i) the effort (ambition) to offer a comprehensive meta-model for non-functional properties that spans over dependability, performance, security and trust attributes and is machine-processable: existing meta-models generally address only a subset of the above properties or do not support transformational approaches; ii) the interconnection between the above meta-model and a modular event-based monitoring infrastructure.

2.2 CONNECT Property Meta-Model Improvements

The improvements on the CPMM in this year are about the *Property* and mainly about the *EventType* definitions, as detailed below [2].

2.2.1 Property specification

As described in D5.2 [28], a *Property* in CPMM is a *NamedElement* having two required/mandatory attributes and three optional ones. The required attributes are: *nature* and *PropertyClass*. The *nature* attribute refers to the nature of the property that can be *ABSTRACT*, *DESCRIPTIVE*, or *PRESCRIPTIVE*, the *PropertyClass* can have the following values: *PERFORMANCE*, *SECURITY*, *TRUST*, and *DEPENDABILITY*. An *ABSTRACT* property indicates a generic property that 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. 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*. A note with an OCL constraint (see upper note at the right-hand side of Figure 2.2) has been added to set that 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 and the *unit* attribute indicates its unit of measure whereas the *operator* attribute models a relational operator.

A property can be qualitative (*QualitativeProperty*) or quantitative (*QuantitativeProperty*). A *QualitativeProperty* models properties about the event occurrences of an *EventSet* that are observed and cannot be measured. They in general refer to the behavioral description of the system (e.g., deadlock freeness or liveness). Quantitative properties (*DESCRIPTIVE* or *PRESCRIPTIVE*) are measurable and have associated *Metrics*. A *QuantitativeProperty* can have a *Workload* and/or an *IntervalTime*. The former is mandatory for *PERFORMANCE* properties, while the latter is mandatory for *DEPENDABILITY* ones. The *Workload*, which in the previous version of the CPMM was an attribute (with a string value) of the *QuantitativeProperty*, in the current CPMM version is an element that can be open or close and has the *timeUnit* attribute specified according to one of the time units listed in the *TimeUnitType* enumeration.

The *IntervalTime*, which also was an attribute (with a string value) of the *QuantitativeProperty*, in the revised CPMM is a new element that has the *timeUnit* and the *timeValue* attributes, needed to specify the interval of time it represents. Figure 2.2 reports the portion of the current meta-model defining a *Property*.

The most important improvement in the current CPMM version with respect to that presented in D5.2 [28] is about the *EventTypeModel* and is represented by the current *EventTypeSpecification*. While in the previous version of the meta-model this specification was simply defined by means of a label or string, now we provide a formal specification for the complex event definition, as we detail in the following.

2.2.2 EventType specification

The *EventTypeModel* of the CPMM is composed by zero or more *EventType* elements, each one modeling the type of an event. As described in D5.2 [28], 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 one or more parameters, one constraint and is composed by one or more *ComplexEvent*. We provide a formal specification for the complex event definition.

As presented in Figure 2.3, the *ComplexEvent* of the *EventType* model is a combination of primitive and other composite events, combined by means of the operators defined in *OperatorType*. The required *compositionOrder* attribute represents the order of the events in the composition and can take one of the values listed in the *Ordering* enumeration. The *OperatorType* of *EventType* model can be one of the operators described in Table 2.1, where e1 and e2 represent the current and correlated events respectively and e3 is a generic event involved in some operators.

We detail in the following all the operators:

[2] Italic words represent CPMM elements.

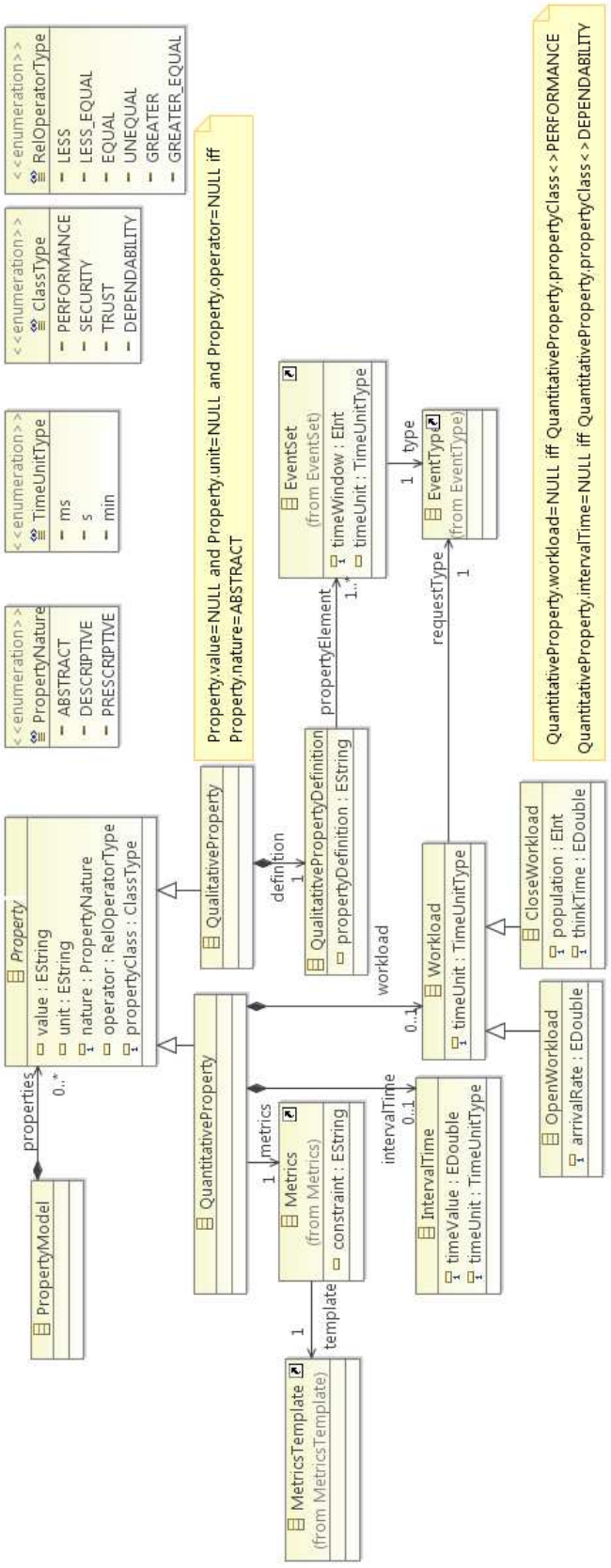


Figure 2.2: Property

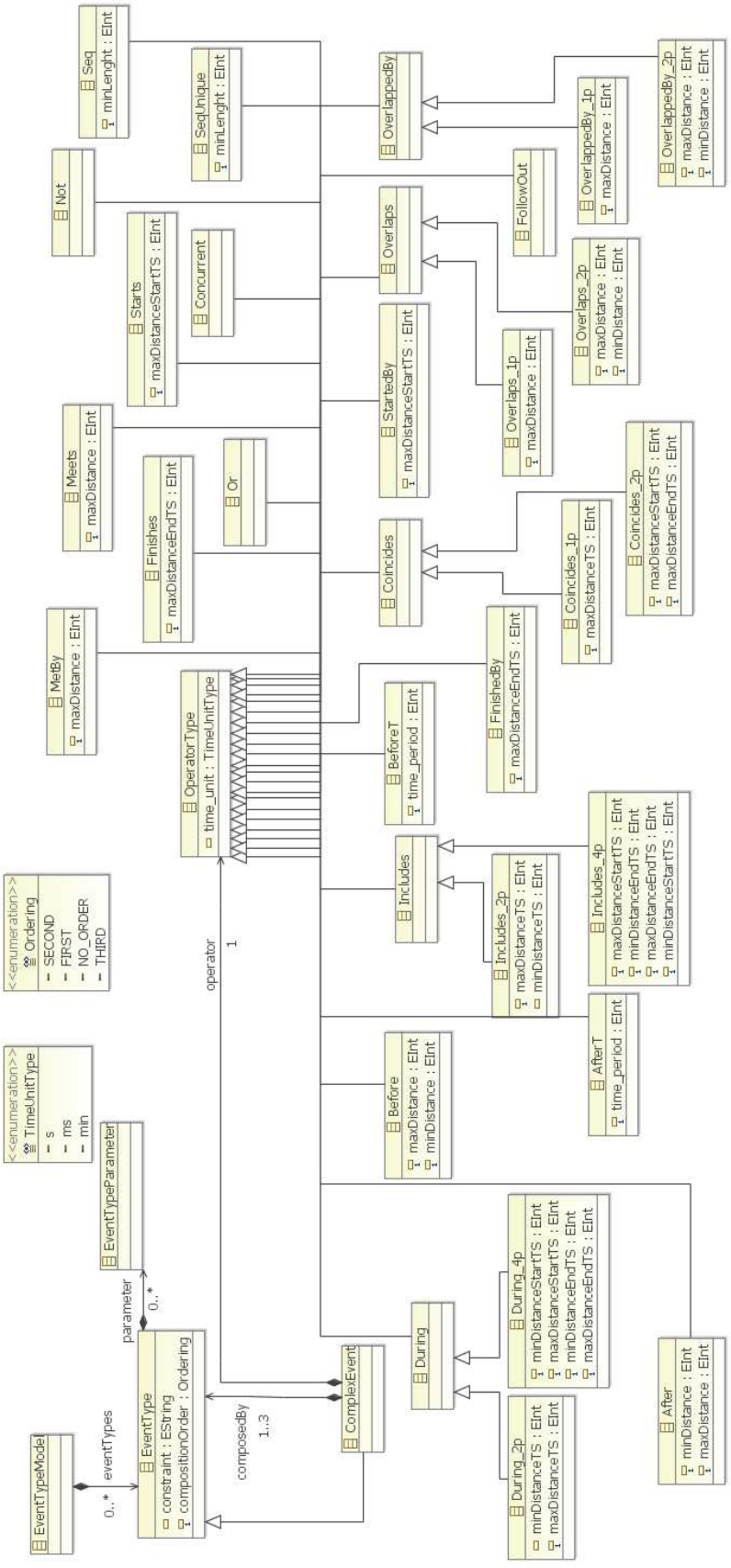


Figure 2.3: EventType

After The *After* operator involves two events and occurs when the current event happens after the correlated event. This operator uses two parameters to quantify the temporal distance between the time when the correlated event finishes and the current event starts: the former indicates the minimum distance while the latter indicates the maximum one. These parameters are called *minDistance* and *maxDistance*.

AfterT The *AfterT* operator involves one event and a time-period, it occurs when the event happens after the specified time-period. A parameter is defined corresponding to the time-period.

Before The *Before* operator involves two events and occurs when the current event happens before the correlated event. This operator uses two parameters to quantify the temporal distance between the time when the current event finishes and the correlated event starts, the former indicates the minimum distance while the latter indicates the maximum one. These parameters are called *minDistance* and *maxDistance*.

BeforeT The *BeforeT* operator involves one event and a time-period, it occurs when the event is followed by a specified time-period. A parameter is defined corresponding to the time-period.

Coincides The *Coincides* operator involves two events and occurs when both happen at the same time, specifically the two events have the same start and end timestamps. This operator accepts one or two parameters, if only one parameter is defined, it represents the maximum distance between the corresponding timestamps while if two parameters are defined, the former represents the maximum distance between the start timestamps while the latter represents the maximum distance between the end timestamps. In the CPMM there are two versions of this operator: the former is implemented by the *Coincides_1p* operator and represents the behavior when only one parameter is defined (*maxDistanceTS*); the latter corresponds to *Coincides_2p* operator that represents the behavior using two parameters, these parameters are called *maxDistanceStartTS* and *maxDistanceEndTS*.

Concurrent The *Concurrent* operator involves two events and occurs when both events happen irrespective of their order.

During The *During* operator involves two events and occurs when the current event happens during the correlated event: specifically the current event starts after the correlated event and finishes before it. This operator accepts one, two or four parameters: if only one parameter is defined, it represents the maximum distance between the start timestamps of the two events and the maximum distance between the end timestamps; if two parameters are defined, the former represents the minimum distance between the timestamps while the latter represents the maximum distance between the timestamps; if four parameters are defined, the first two values represent the minimum and the maximum distance between the start timestamps while the other two values represents the minimum and the maximum distance between the end timestamps. In the CPMM there are two versions of this operator: the former is implemented by the *During_2p* operator that represents the behavior when one parameter (*maxDistanceTS*) or two parameters (*maxDistanceTS* and *minDistanceTS*) are defined; the latter is implemented by the *During_4p* operator that represents the behavior when four parameters (*minDistanceStartTS*, *maxDistanceStartTS*, *minDistanceEndTS* and *maxDistanceEndTS*) are defined.

FinishedBy The *FinishedBy* operator involves two events and occurs when the current event starts before the correlated event but both events end at the same time. This operator accepts one parameter (*maxDistanceEndTS*) that indicates the maximum distance between the end timestamps.

Finishes The *Finishes* operator involves two events and occurs when the current event starts after the correlated event but both events end at the same time. This operator accepts one parameter, called *maxDistanceEndTS*, that indicates the maximum distance between the end timestamps.

FollowOut The *FollowOut* operator involves three events and occurs when the first event is followed by the second event and without the occurrence of the third event.

Includes The *Includes* operator involves two events and occurs when the correlated event happens during the current event, specifically the correlated event starts after the current event and finishes before it. It is the symmetrical opposite of the During operator. This operator accepts one, two or four parameters: if only one parameter is defined, it represents the maximum distance between the start timestamps of the two events and the maximum distance between the end timestamps; if two parameters are defined, the former represents the minimum distance between the timestamps while the latter represents the maximum distance between the timestamps; if four parameters are defined, the first two values represent the minimum and the maximum distance between the start timestamps while the other two values represent the minimum and the maximum distance between the end timestamps. In CPMM there are two versions of this operator: the former is implemented by the *Includes_2p* operator that represents the behavior when one parameter (*maxDistanceTS*) or two parameters (*maxDistanceTS* and *minDistanceTS*) are defined; the latter is implemented by the *Includes_4p* operator that represents the behavior when four parameters (*minDistanceStartTS*, *maxDistanceStartTS*, *minDistanceEndTS* and *maxDistanceEndTS*) are defined.

Meets The *Meets* operator involves two events and occurs when the current event finishes at the same time when the correlated event starts. This operator accepts one parameter, called *maxDistance*, that indicates the maximum distance between the end timestamp of the current event and the start timestamp of the correlated event.

MetBy The *MetBy* operator involves two events and occurs when the current event starts at the same time when the correlated event finishes. This operator accepts one parameter, called *maxDistance*, that indicates the maximum distance between the end timestamp of the correlated event and the start timestamp of the current event.

Not The *Not* operator involves one event and occurs when the specified event doesn't happen.

Or The *Or* operator involves two events and occurs when one of the two events happens.

OverlappedBy The *OverlappedBy* operator involves two events and occurs when the correlated event happens before the current event and finishes before the current event but after the current event starts. This operator accepts one or two parameters, if only one parameter is defined, it represents the maximum distance between the start timestamp of the current event and the end timestamp of the correlated event. If two parameters are defined, the first one represents the minimum distance while the second one represents the maximum distance between the start timestamp of the current event and the end timestamp of the correlated event. In the CPMM there are two versions of this operator: the former is implemented by the *OverlappedBy_1p* operator that represents the behavior when only one parameter (*maxDistance*) is defined, the latter is implemented by the *OverlappedBy_2p* operator that represents the behavior when two parameters (*minDistance* and *maxDistance*) are defined.

Overlaps The *Overlaps* operator involves two events and occurs when the current event start before the correlated event and finishes before it but after that the correlated event starts. This operator accepts one or two parameters: if only one parameter is defined it represents the maximum distance between the start timestamp of the correlated event and the end timestamp of the current event. If two parameters are defined, the former represents the minimum distance while the latter represents the maximum distance between the start timestamp of the correlated event and the end timestamp of the current event. In the CPMM there are two versions of this operator: the former is implemented by the *Overlaps_1p* operator and represents the behavior when only one parameter is defined (*maxDistance*); the latter corresponds to *Overlaps_2p* operator that represents the behavior using two parameters, these parameters are called *minDistance* and *maxDistance*.

Seq The *Seq* operator involves one event and occurs when there is a sequence of occurrences of it. This operator accepts one parameter (*minLenght*) that indicates the minimum length of the sequence.

SeqUnique The *SeqUnique* operator is similar to the *Seq* operator. The additional feature is that the sequence captured by *SeqUnique* doesn't contain duplicate occurrences of the events.

StartedBy The *StartedBy* operator involves two events and occurs when both events start at the same time and the correlated event finishes before the current one. This operator accepts one parameter (*maxDistanceStartTS*) that indicates the maximum distance between the start timestamps of the events.

Starts The *Starts* operator involves two events and occurs when both events start at the same time and the current event finishes before the correlated event. This operator accepts one parameter (*maxDistanceStartTS*) that indicates the maximum distance between the start timestamps of the events.

The *EventType* model defined in the current version of CPMM combines features of two existing event specification languages that are GEM [54] and Drools Fusion [1] (see Section 2.5) and in addition presents new features not included in the considered languages. In particular, we have defined operators that allow for modeling a temporal relationship (as those of Drools Fusion) and operators that allow for combining simple or complex events (as those of GEM), in addition we have identified situations of interest not covered by the operators of GEM and Drools Fusion that have been formalized through new operators. In Table 2.2 we present a mapping between CPMM *EventType* model operators and the corresponding ones in GEM and Drools Fusion.

Table 2.1: Event Composition Operators

Operator	Parameters	Parameter meaning
After (Before) e1 after (before) e2	*minDistance* *maxDistance*	min time distance between e2 (e1) finishing and e1 (e2) starting max time distance between e2 (e1) finishing and e1 (e2) starting
AfterT (BeforeT)	*time_period*	time period before (after) that the event occurs
Coincides_1p	*maxDistanceTS*	max distance between start and end timestamps (e1 and e2 have same timestamps)
Coincides_2p	*maxDistanceStartTS* *maxDistanceEndTS*	max distance between e1 and e2 start timestamps max distance between e1 and e2 end timestamps
Concurrent e1, e2 irrespective of their order	- -	- -
During_2p e1 during e2	*maxDistanceTS* *minDistanceTS*	max distance between e1 and e2 timestamps min distance between e1 and e2 timestamps
During_4p e1 during e2	*maxDistanceStartTS* *minDistanceStartTS* *maxDistanceEndTS* *minDistanceEndTS*	max distance between e1 and e2 start timestamps min distance between e1 and e2 start timestamps max distance between e1 and e2 end timestamps min distance between e1 and e2 end timestamps
Finishes (FinishedBy) e1 starts after (before) e2 e1, e2 end at the same time	*maxDistanceEndTS*	max distance between e1 and e2 end timestamps
FollowOut e1 followed by e2 without e3	- -	- -
Includes_2p e2 during e1	*maxDistanceTS* *minDistanceTS*	max distance between e1 and e2 timestamps min distance between e1 and e2 timestamps
Includes_4p e2 during e1	*maxDistanceStartTS* *minDistanceStartTS* *maxDistanceEndTS* *minDistanceEndTS*	max distance between e1 and e2 start timestamps min distance between e1 and e2 start timestamps max distance between e1 and e2 end timestamps min distance between e1 and e2 end timestamps
Meets (MetBy) e1 finishes (starts) when e2 starts (finishes)	*maxDistance*	max distance between e1 (e2) end and e2 (e1) start timestamps
Not e1 doesn't happen	- -	- -
Or e1 or e2 happens	- -	- -
Overlaps_1p (OverlappedBy_1p) e1 (e2) finishes after e2 (e1) starts	*maxDistance*	max distance between e2 (e1) start and e1 (e2) end timestamps
Overlaps_2p (OverlappedBy_2p) e1 (e2) finishes after e2 (e1) starts	*maxDistance* *minDistance*	max distance between e2 (e1) start and e1 (e2) end timestamps min distance between e2 (e1) start and e1 (e2) end timestamps
Seq sequence of e1 occurrences	*minLenght*	min length of the sequence
SeqUnique sequence of e1 occurrences no duplicate occurrences	*minLenght*	min length of the sequence
Starts (StartedBy) e1 (e2) finishes before e2 (e1) e1, e2 start at the same time	*maxDistanceStartTS*	max distance between e1 and e2 start timestamps

Table 2.2: Mapping with GEM and Drools operators

CPMM operator	GEM operator	Drools operator
A After B $(minDistance = x)$ $(maxDistance = y)$	(B+t); A $x \leqslant t \leqslant y$	A after B[x,y]
A AfterT 10	-	-
A Before B $(minDistance = x)$ $(maxDistance = y)$	(A+t); B $x \leqslant t \leqslant y$	A before B[x,y]
A BeforeT 10	A + 10	-
A Coincides_1p B $(maxDistanceTS = x)$	$(((A_s + t); B_s)I((B_s + t); A_s)))\&(((A_e + t); B_e))I ((B_e + t); A_e))t \leqslant x$	A coincides B[x]
A Coincides_2p B $(maxDistanceStartTS = x)$, $(maxDistanceEndTS = y)$	$(((A_s + t_1); B_s)I((B_s + t_1); A_s)))\&(((A_f + t_2); B_f))I ((B_f + t_2); A_f))t_1 \leqslant x; t_2 \leqslant y$	A coincides B[x,y]
A Concurrent B	$A\&B$	-
A During_2p B $(maxDistanceTS = x)$	$((B_s + t); A_s)\&((A_e + t); B_e)t \leqslant x$	A during B[x]
A During_2p B $(maxDistanceTS = x)$ $(minDistanceTS = y)$	$((B_s + t); A_s)\&((A_e + t); B_e)x \leqslant t \leqslant y$	A during B[x,y]
A During_4p B $(minDistanceStartTS = x)$ $(maxDistanceEndTS = y)$ $(minDistanceEndTS = u)$ $(maxDistanceStartTS = z)$	$((B_s + t_1); A_s)\&((A_e + t_2); B_e)x \leqslant t_1 \leqslant y; u \leqslant t_2 \leqslant z$	A during B[x,y,u,z]
A FinishedBy B $(maxDistanceEndTS = x)$	-	A finishedby B[x]
A Finishes B $(maxDistanceEndTS = x)$	$(B_s; A_s)\&(((A_e + t); B_e)I ((B_e + t); A_e))t \leqslant x$	A finishes B[x]
A FollowOut B,C	$\{A; B\}!C$	-
A Includes_2p B $(minDistanceTS = x)$	$B_e)t \leqslant x$	A includes B[x]
A Includes_2p B $(maxDistanceTS = x)$ $(minDistanceTS = y)$	$B_e)x \leqslant t \leqslant y$	A includes B[x,y]
A Includes_4p B $(minDistanceStartTS = x)$ $(minDistanceEndTS = y)$ $(maxDistanceEndTS = u)$ $(maxDistanceStartTS = z)$	$B_e)x \leqslant t_1 \leqslant y; u \leqslant t_2 \leqslant z$	A includes B[x,y,u,z]
A Meets B $(maxDistance = x)$	-	A meets B[x]
A MetBy B $(maxDistance = x)$	-	A metby B[x]
Not A	-	-
A Or B	AIB	-
A OverlappedBy_1p B $(maxDistance = x)$	-	A overlappedby B[x]
A OverlappedBy_2p B $(minDistance = x)$ $(maxDistance = y)$	-	A overlappedby B[x,y]

A Overlaps_1p B $(maxDistance = x)$	$(A_s; ((B_s + t); A_e)); B_s$ $t \leqslant x$	A overlaps B[x]
A Overlaps_2p B $(minDistance = x)$ $(maxDistance = y)$	$(A_s; ((B_s + t); A_e)); B_s$ $x \leqslant t \leqslant y$	A overlaps B[x,y]
SeqA$(min\_lenght = n)$	$A; A; A; A; \cdots$ (n times)	A after A after A after A $\cdots$ (n times)
SeqUniqueA$(min\_lenght = n)$	$A_1; A_2; A_3; \cdots A_n$ [2]	A_1 after A_2 after $A_3 \cdots$ after A_n
A StartedBy B $(maxDistanceStartTS = x)$	-	A startedby B[x]
A Starts B $(maxDistanceStartTS = x)$	$(((A_s + t); B_s) I ((B_s + t);$ $A_s)) \& (A_s; B_s) t \leqslant x$	A starts B[x]

2.3 Property-driven Monitoring Configuration

We have described so far how CPMM defines CONNECTability properties. However, there is a gap to be filled between the definition of properties of interest and their concrete usage within the CONNECT architecture. In particular, we refer to their usage within the GLIMPSE monitoring infrastructure in order to be able to evaluate and keep under control any such property at runtime. We need concretely to instruct GLIMPSE about what raw data (events) to collect and how to infer whether or not a desired property is fulfilled. Unfortunately, not only this task is time-consuming, but without recurring to a formalized model-driven approach, it would also require a substantial human effort and specialized expertise if the high-level description of the CONNECT properties to observe have to be translated into lower-level monitor configuration directives. Consequently, the outcome of such effort is very hard to generalize and to reuse, and, as a matter of fact, the resulting monitor configuration typically is only relevant in the specific situation at hand. This process would need to be iterated each time the properties to be monitored change.

To address such issues, in CONNECT we provide a model-based approach to automatically convert CPMM metrics and properties specifications into a concrete monitoring setup. Indeed CPMM supports the definition of quantitative and qualitative properties in a machine-processable way allowing for the dynamic monitors configuration. Precisely, the editor provided along with CPMM allows the software developer to specify a new property as a model that is conforming to the CPMM meta-model. If this model represents a property to be monitored, it can be used to instruct the GLIMPSE infrastructure, according to the approach sketched in Figure 2.4. As shown, the GLIMPSE manager component takes in input such property model and activates an external component (named in the figure Model2Code Transformer), which performs the code generation according to a specific complex event processing language that is embedded into GLIMPSE. The output of this transformation is represented by a specific rule that the manager component passes to the complex event processor component of GLIMPSE.

The current running implementation of GLIMPSE infrastructure uses the Drools Fusion complex event processor [1] and the code generator used for transforming models into code is Acceleo [5]. We developed a first version of an automated Model2Code Transformer component according to the Drools Fusion rule specification language. Indeed, the advantage of adopting a model-driven approach is that it allows the monitor to use any complex event processing engine as long as a Model2Code Transformer transforms the property model into the rule specification language of that processing engine.

In the next sections we show in detail first the Acceleo transformer component (Section 2.3.1), then a coverage property model required in the system for the CONNECT scenario, specified using the CPMM meta-model (in Section 2.4), and finally in Section 2.4.1 we provide the obtained rules for Drools Fusion generated for monitoring the considered property.

[1] A_s and B_s represent the start timestamps of the events A and B respectively, similarly A_e and B_e represent the end timestamps of A and B.

[2] $A_1; A_2; A_3; \cdots A_n$ are events with the same type but different parameters values.

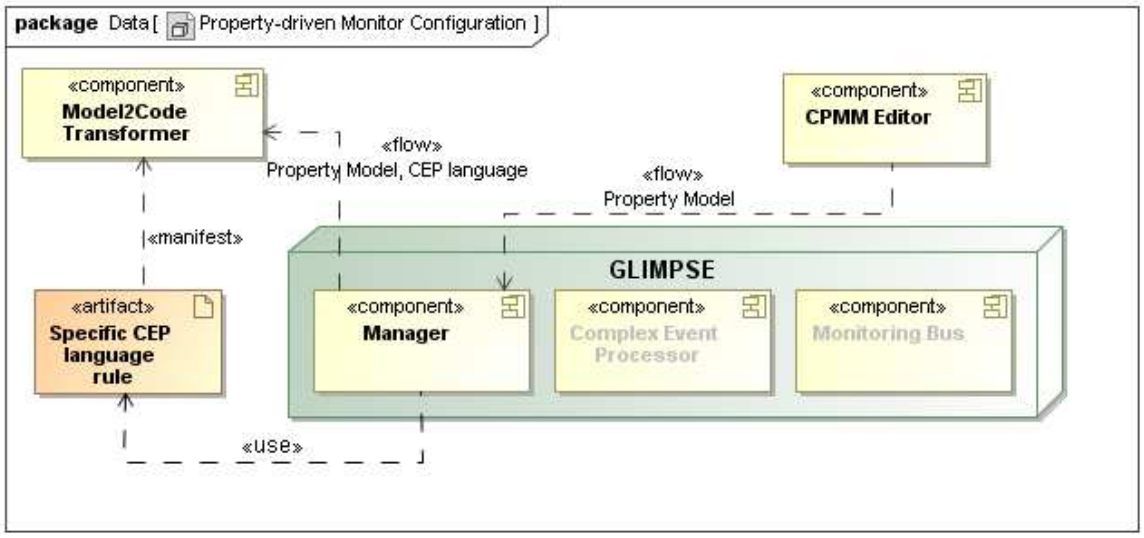


Figure 2.4: Property-Driven GLIMPSE Configuration

2.3.1 Model2Code transformer description

The Model2Code Transformer component depicted in Figure 2.4 takes as inputs the CPMM ecore models (specifically Core.ecore, EventSet.ecore, EventType.ecore, Metrics.ecore, MetricsTemplate.ecore, Property.ecore) and the model (compliant to CPMM) of the property to be monitored, and produces Drools Fusion code, specifically one or more Drools rules, that are the input to the GLIMPSE complex event processor.

We show here some parts (Listing 2.1) of the Model2Code Transformer code. Specifically, this code derives the Drools rules presented in Listing 2.2. After specifying the input meta-model and the imported services (line 2-3) we define a main template (line 4-13) where if the input model is about a quantitative property, a specific template (line 20-48) processing the associated metrics is called. In this template the *Metrics* model is navigated for identifying the associated *MetricsTemplate* model, then the *MetricsTemplate* parameters and the associated *EventBasedExpression*s. These *EventBasedExpression*s represent expressions based on events and their name represents the observable, simple or complex, event/behavior the operator applies to. If the *EventOperatorType* is CARDINALITY, this operator has to be applied to the whole set of event occurrences of the observed *EventType*, then a specific template (named *processCount*) addressing counting of events occurrences (line 33) is called. This template (line 50-72) takes as input an *EventSet* and generates a Drools rule counting the event occurrences associated to an *EventType* in the time window dimension specified in the *EventSet*. For instance, if the model inputs are those of coverage property presented in Section 2.4, this template generates two Drools rules (see Listing 2.2) counting the number of *eAler_eAck* complex event occurrences (line 1-21 of Listing 2.2) and that of *deviceRegistration* simple event occurrences (line 24-42 of Listing 2.2) respectively. The *processCount* template calls on the inside the *processEventType* template. This template (line 74-102) navigates the *EventType* model and maps all the *EventType* operators presented in Table 2.1 into the corresponding Drools Fusion operators as specified in Table 2.2.

As an example, we show the transformation code for the *Before* (line 84-91) and *Seq* (line 92-99) operators that are used for defining the *eAler_eAck* event model presented in Figure 2.11. Note that, for addressing the *Seq* operator we need to call a Java service named *Print* (line 96) by defining an Acceleo query (line 140-141) in the template. Finally, we navigate the *MetricsTemplates* model (line 108-138) for identifying the *Dimension* indicating the type of the value defined by the *MetricsTemplate* and all the defined mathematical expressions (they can be nested), and for each of them we generate a Drools rule addressing the corresponding *MathOperatorType*. As an example of the Model2Code transformer application, we will present in Section 2.4.1 the Drools code generated for the coverage property models presented in Section 2.4.

```
1  [comment encoding = UTF-8 /]
2  [module generate('cpmm/model/Core.ecore','cpmm/model/EventType.ecore','cpmm/model/EventSet.
       ecore','cpmm/model/Metrics.ecore','cpmm/model/MetricsTemplate.ecore','cpmm/model/Property
       .ecore')]
3  [import cpmm::acceleo::utilities::utility]
4  [template public generateElement(p : Property)]
5  [comment @main/]
6  [file (p.name, false, 'Cp1252')]
7   [if p.oclIsTypeOf(QuantitativeProperty)]
8    [processMainQuantitativeProperty(p)/]
9   [elseif (p.oclIsTypeOf(QualitativeProperty))]
10   [processQualitativeProperty(p)/]
11  [/if]
12 [/file]
13 [/template]
14
15 [template public processMainQuantitativeProperty(p : Property)]
16 [processMetrics(p.oclAsType(QuantitativeProperty).metrics)/]
17 [/template]
18
19
20 [template public processMetrics(m : Metrics)]
21 [comment ————omissis————————/]
22 [if (m.metricsTemplate.definition.oclIsTypeOf(MathExpression))]
23  [for (op : Expression | m.metricsTemplate.definition.oclAsType(MathExpression).operands)]
24   [if (op.oclIsTypeOf(MathExpression))]
25    [for (op1 : Expression | op.oclAsType(MathExpression).operands)]
26     [if (op1.oclIsKindOf(NamedExpression))]
27      [if (op1.oclAsType(NamedExpression).oclIsTypeOf(EventBasedExpression))]
28       [if  (op1.oclAsType(NamedExpression).oclAsType(EventBasedExpression).operator.toString
             () = 'CARDINALITY')]
29        [for (act : MetricsParameter | m.actualParameters)]
30         [if (act.oclIsTypeOf(EventBasedMetricsParameter))]
31          [for (tp: EStringToNamedExpressionObjectMap | m.metricsTemplate.TemplateParameters)]
32           [if ((tp.key = act.oclAsType(EventBasedMetricsParameter).name) and (tp.value=op1))]
33            [processCount(act.oclAsType(EventBasedMetricsParameter).eventSet)/]
34           [/if]
35          [/for]
36         [/if]
37        [/for]
38       [/if]
39      [/if]
40     [/if]
41    [/for]
42   [/if]
43  [/for]
44 [/if]
45 [comment ————omissis————————/]
46 [processMetricsTemplate (m.metricsTemplate)/]
47 [comment ————omissis————————/]
48 [/template]
49
50 [template public processCount(es : EventSet)]
51
52 declare Total_[es.type.name/]captured
53     @total: int
54 end
55
56 rule "Number of [es.type.name/] incomingEvents"
57 no-loop
58 salience 999
59 dialect "java"
60
61 when
62  $total[es.type.name/]: Number();
63   from accumulate(
64    $event_[es.type.name/]:[processEventType(es.type)/]over window:time( [if not(es.timeWindow
       .oclIsUndefined())]
```

```
65  [es.timeWindow/][/if][if not(es.timeUnit.oclIsUndefined())][es.timeUnit/][/if] ) from entry-
        point "DEFAULT", count($event_[es.type.name/]))
66 then
67   Total_[es.type.name/]captured count[es.type.name/] = new Total_[es.type.name/]captured();
68   count[es.type.name/].setTotal($total[es.type.name/])
69   insert(count[es.type.name/]);
70   System.out.println( "Number of Incoming events: " + $total[es.type.name/]);
71 end
72 [/template]
73
74 [template public processEventType(et : EventType)]
75 [if (et.oclIsTypeOf(ComplexEvent))]
76  [processComplexEvent(et.oclAsType(ComplexEvent))/]
77  [else]
78  [processSimpleEvent(et)/]
79  [/if]
80 [/template]
81
82 [template public processComplexEvent(ce : ComplexEvent)]
83 [comment ---------omissis-----------------/]
84 [elseif (ce.operator.oclIsTypeOf(Before))]
85 [for (op : EventType | ce.composedBy)]
86   [if (not (op.oclIsTypeOf(ComplexEvent)) and (op.compositionOrder.toString() = 'FIRST'))]
87    [op.name/][' '/][processBefore(ce.operator.oclAsType(Before))/]
88   [elseif (op.compositionOrder.toString() = 'SECOND') and (not (op.oclIsTypeOf(ComplexEvent))
         )] $[op.name/]event [') '/]
89     [elseif (op.oclIsTypeOf(ComplexEvent))][processComplexEvent(op.oclAsType(ComplexEvent))
           /][') '/]
90   [/if]
91  [/for]
92 [elseif (ce.operator.oclIsTypeOf(Seq))]
93 $event_[ce.name/]:
94 [for (op : EventType | ce.composedBy)]
95  [if (not (op.oclIsTypeOf(ComplexEvent)))]
96    [op.name/][' '/] this [ce.operator.oclAsType(Seq).eClass().Print(ce.operator.oclAsType(Seq
         ).eClass(), ce.operator.oclAsType(Seq).minLenght-1, op.name.toString())/][') '/]
97   [else] [processComplexEvent(op.oclAsType(ComplexEvent))/]
98  [/if]
99 [/for]
100 [comment ---------omissis-----------------/]
101 [/if]
102 [/template]
103
104 [template public processBefore (a : Before)]
105  this before ['['/] [a.minDistance/],[a.maxDistance/] [']'/]
106 [/template]
107
108 [template public processMetricsTemplate (mt: MetricsTemplate)]
109 [comment ---------omissis-----------------/]
110 [if (mt.oclAsType(MetricsTemplate).dimension.toString() = 'PERCENTAGE')]
111  [processExpression(mt.definition)/]
112 [/if]
113 [comment ---------omissis-----------------/]
114 [/template]
115
116 [template public processExpression(e : Expression)]
117 [if (e.oclIsTypeOf(MathExpression))]
118  [processMathExpression(e.oclAsType(MathExpression))/]
119 [elseif (e.oclIsTypeOf(Constant))]
120  [processConstantExpression(e.oclAsType(Constant))/]
121 [elseif (e.oclIsTypeOf(QuantitativeProperty))]
122  [processQuantitativeProperty(e.oclAsType(QuantitativeProperty))/]
123 [elseif (e.oclIsKindOf(NamedExpression))]
124  [processNamedExpression(e.oclAsType(NamedExpression))/]
125 [/if]
126 [/template]
127
128 [template public processMathExpression(e : MathExpression)]
129 [if (e.operator.toString()='DIVISION')]
```

```
130   [processDivision(e.operator)/]
131  [elseif (e.operator.toString()='AVG')]
132   [processAVG(e.operator)/]
133  [comment ----------omissis----------/]
134  [/if]
135  [for (op : Expression | e.operands)]
136   [processExpression(op)/]
137  [/for]
138  [/template]
139  [comment ----------omissis----------/]
140  [query public Print(seq : EClass, length : Integer, opname : String) : String
141    = invoke('cpmm.acceleo.utilities.utility', 'Print(org.eclipse.emf.ecore.EClass, int, java.
         lang.String)', Sequence{arg0, arg1, arg2}) /]
```

Listing 2.1: Acceleo code for Model2Code Transformer

Acceleo

For the sake of completeness, we recall that the Model2Code Transformer component has been implemented using the Acceleo code generator IDE [5]. Acceleo is an Eclipse based product, included in the Eclipse release since Eclipse 3.6 Helios. It is a reference implementation of the OMG MOF (Meta-Object Facility) Model to Text Language (MTL) standard. It supports the automated code generation from UML and EMF and provides the developer with simple syntax, efficient code generation, advanced tool features such as quick outline, navigation links to the declaration of model elements, template elements and variables, quick fixes, refactoring, syntax highlighting and so on.

Acceleo uses a template to generate code (or text) from a model. In the generated template we can find the heading section that defines which meta-model the template will apply for and the generated file; then the template is divided into a certain number of scripts. A script is the elementary unit of a template and is applied on a model element to produce some text. Every script will be evaluated on a given element type. There are static areas, they will be included as they are defined in the generated file and dynamic areas that correspond to the expression evaluation on the current object.

Acceleo supports the user code blocks specification in some areas of the generated file. A common way to do this is by defining a Java service (standard Java code) that is executed by accessing to it from any Acceleo template or query.

2.4 *Coverage* Property in the Terrorist Alert Scenario: An Example

In this section, we first recall briefly the Terrorist Alert Scenario which is one of the demonstration examples developed in WP6 (see D6.2 [29]). Then, we show how to use the current CPMM for modeling the *coverage* property for the Terrorist Alert scenario.

Terrorist Alert scenario The CONNECT Terrorist Alert scenario developed in D6.2 [29] depicts the critical situation that during a show in a stadium, the control center spots one suspect terrorist moving around. The alarm is immediately sent to the Policemen, equipped with ad hoc handheld devices which are connected to the Police control center to receive commands and documents, 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. The 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 some civilian guards, on service around the stadium, an alert message in which one picture of the suspect is distributed. The guards control center sends an *EmergencyAlert* message to all guards of the patrolling groups; the message reports the alert details. On correct receipt of the alert, each guard's device automatically sends an ack to the control center.

***Coverage* Property for the Terrorist Alert Scenario** 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*

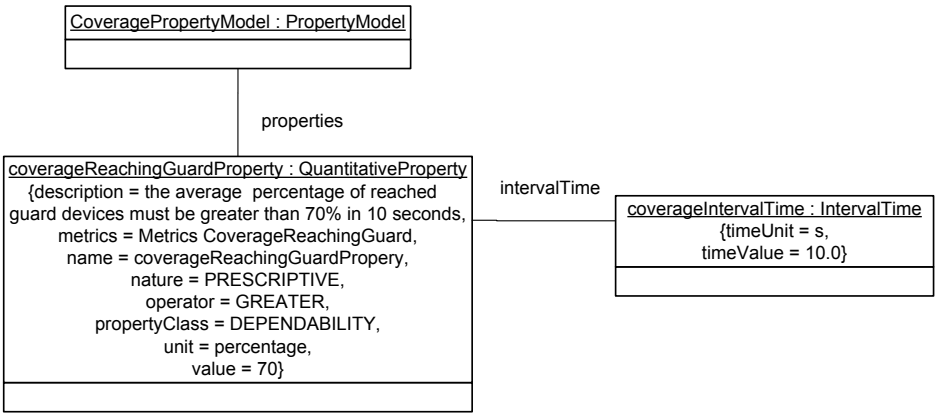


Figure 2.5: Coverage Property for the Terrorist Alert Scenario

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 model for this property is shown in Figure 2.5. This property is a *PRESCRIPTIVE DEPENDABILITY* property (i.e. a dependability requirement) specifying that the *CoverageReachingGuard* metrics must be *GREATER* than *70%* after an *IntervalTime* of *10 seconds* starting from the EmergencyAlert event occurrence.

The *CoverageReachingGuard* metrics in Figure 2.7 actualizes the corresponding *Average Coverage Metrics Template* (see Figure 2.6) by linking to the *TemplateParameter*s the corresponding *EventSet*s.

The average coverage represents a *PERCENTAGE* measure defined as average of the division among the *CARDINALITY* of two sets of instances of two types of events, named x and y. Finally, the template exposes two *templateParameters*, e linked to x, and e_3 linked to y (see Figure 2.6). We recall that the template is generic and can be used in other scenarios, this shows the flexibility of the proposed meta-model.

Finally, Figures 2.8 and 2.9 report the model for the e and e_3 event sets, respectively. e_3 *EventSet* 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., events having the same *IDg* value).

The *deviceRegistration EventType* presented in Figure 2.10, 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 is the formal parameter of the interface operation.

The e *EventSet* in Figure 2.8 refers to *eAlert_eAck EventType* without introducing additional conditions.

The *eAlert_eAck EventType*, shown in Figure 2.11, is a complex event representing the *EmergencyAlert* with its related *eAck* from the guards. The *Constraint* attribute defines the related condition that imposes that all the *eAck.IDe* must be equal to the *emergencyAlert.ID*. *eAlert_eAck* has a *Before* operator with *maxDistance* parameter equal to 10. This operator is applied to *eAlert* simple *EventType* and to *Seq_eAck* complex *EventType* representing respectively the former and the latter events to which the *Before* operator is applied. The *eAlert EventType* has the *eAlertID* parameter representing the *EmergencyAlert* ID the sequence refers to. The *Seq_eAck* is another *ComplexEvent* type with *Seq* operator (see Table 2.1). It is composed by *eAck EventType*, with two parameters: *IDg* that is the ID of the reached guard and *IDe* that is the *EmergencyAlert* ID the *eAck* responds to. In this case the event *compositionOrder* is *NO_ORDER*. The *eAlert_eAck EventType* has two parameters: the *EmergencyAlert* ID (namely IDe) the sequence refers to, and the list of guards messages acknowledging the alert (namely, IDgList).

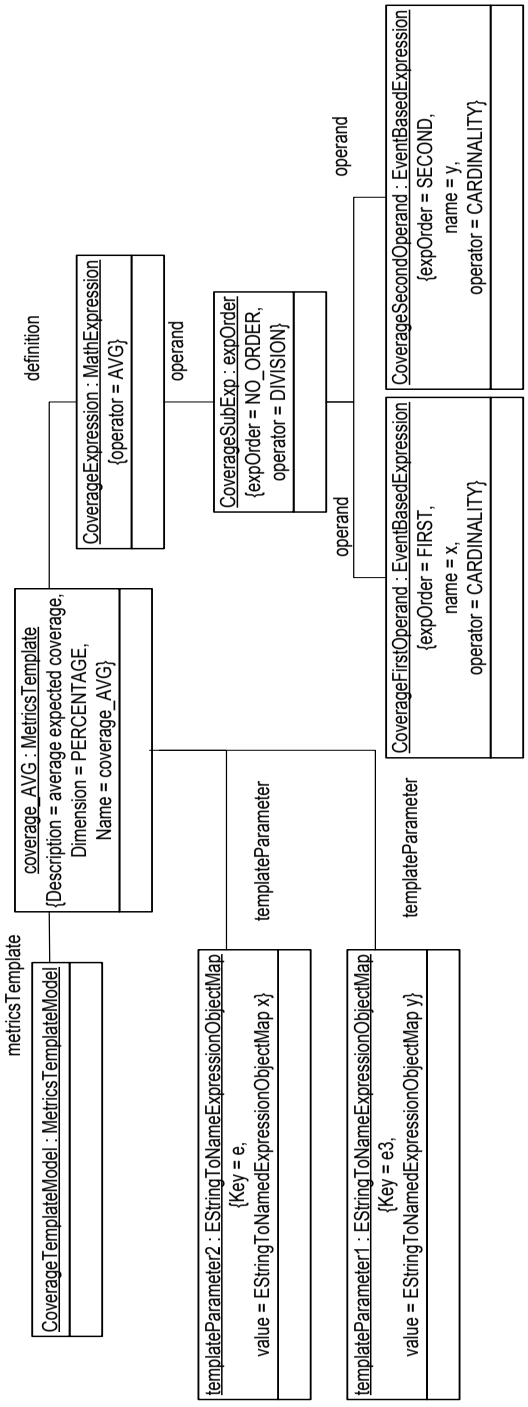


Figure 2.6: Average Coverage Metrics Template

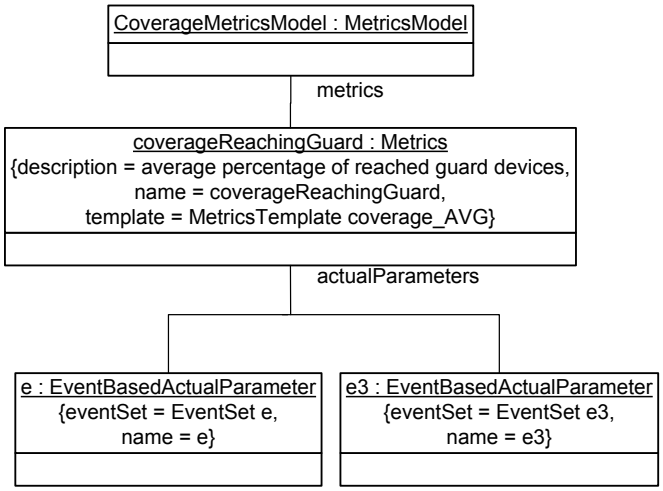


Figure 2.7: Average Coverage Metrics for the Terrorist Alert Scenario

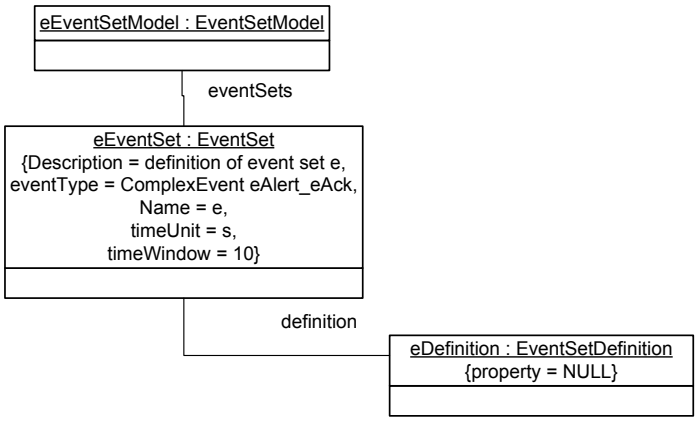


Figure 2.8: e EventSet

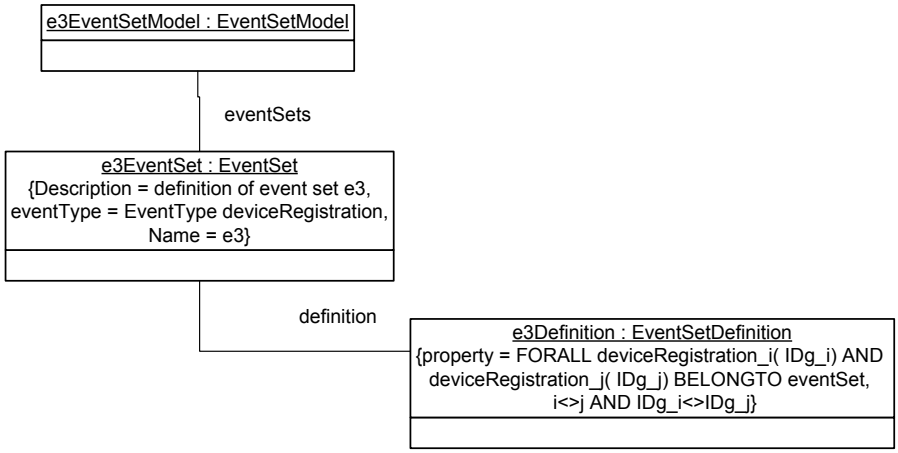


Figure 2.9: e_3 Actual Parameter in the CoverageReachingGuards Metrics Model

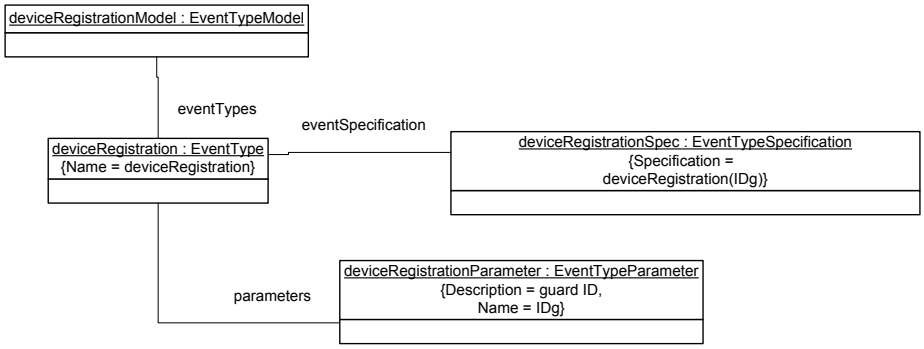


Figure 2.10: Guard Device Registration

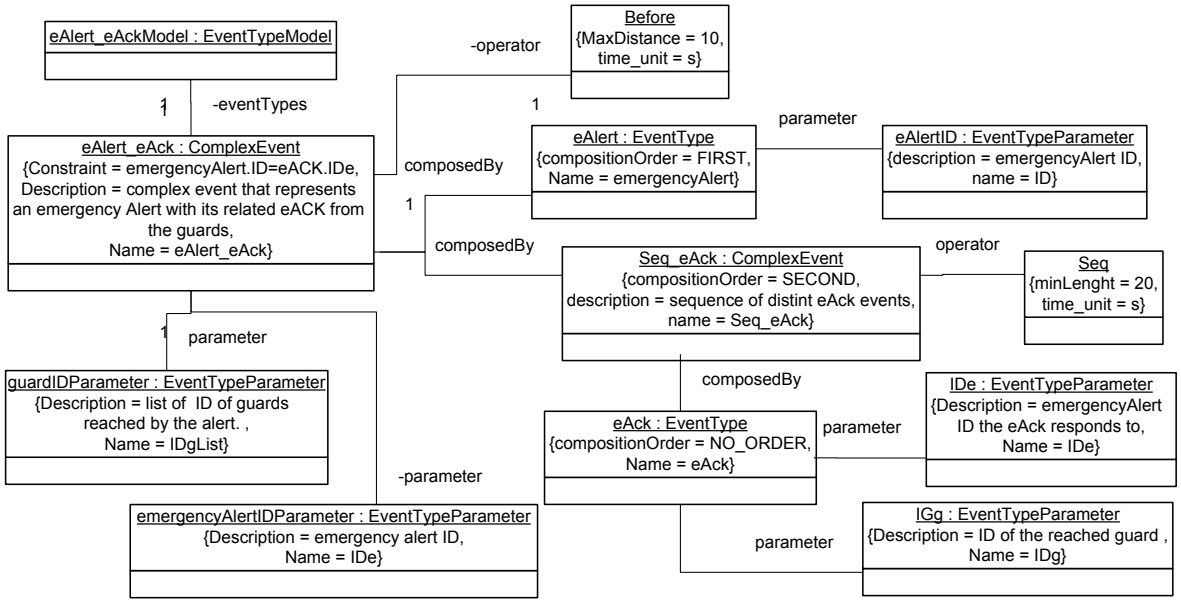


Figure 2.11: Sequence of Ack for an Alert

2.4.1 Drools rule specification for the Coverage Property

The Listing 2.2 shows the Drools code used to monitor the coverage property defined above for the Terrorist Alert Scenario from D6.2 [29]. This code has been generated by the Model2Code Transformer described in Section 2.3.1 specifying as inputs the models of Figures 2.5, 2.6, 2.7, 2.8, 2.9, 2.10, 2.11. Specifically, five rules are generated. The first rule (line 1-21) counts the number of *eAler_eAck* events that happen in a time window of 10 seconds and saves this information into a new generated event (called *counteAlert_eAck*). The second rule (24-42) is similar to the first one: it generates the *countdeviceRegistration* event containing the number of *deviceRegistration* events. Note that as described in Section 2.3.1 both rules are generated by the same piece of the Model2Code Transformer code (see Listing 2.1) for the two different event types (*eAler_eAck* and *deviceRegistration*). The third rule (lines 45-62) captures the *counteAlert_eAck* and *countdeviceRegistration* events and computes a precise coverage measure, its value is then returned in the new generated event (named *percentage*, see line 59). Similarly, the fourth rule (65-83) captures the *percentage* events and computes the average coverage measure. Finally the last rule (86-95) checks that this measure is greater than the specified value in the coverage property model.

```
1  declare Total_eAlert_eAckcaptured
2   @total: int
3  end
4
5  rule "Number of eAlert_eAck incomingEvents"
6  no-loop
7  salience 999
8  dialect "java"
9  when
10  $totaleAlert_eAck: Number();
11   from accumulate(
12  $event_eAlert_eAck: emergencyAlert( this before
13  $event_seq_eAck:
14  eAck( this after eAck after eAck after eAck after eAck after eAck after eAck after eAck
        after eAck after eAck after eAck after eAck after eAck after eAck after eAck after eAck
        after eAck after eAck after eAck after eAck )
15  )over window:time(10s) from entry-point "DEFAULT", count($event_eAlert_eAck))
16 then
17  Total_eAlert_eAckcaptured counteAlert_eAck = new Total_eAlert_eAckcaptured();
18  counteAlert_eAck.setTotal($totaleAlert_eAck)
19  insert(counteAlert_eAck);
20  System.out.println( "Number of Incoming events: " + $totaleAlert_eAck);
21 end
22
23
24 declare Total_deviceRegistrationcaptured
25  @total: int
26 end
27
28 rule "Number of deviceRegistration incomingEvents"
29 no-loop
30 salience 999
31 dialect "java"
32 when
33  $totaldeviceRegistration: Number();
34   from accumulate(
35  $event_deviceRegistration: deviceRegistration
36  from entry-point "DEFAULT", count($event_deviceRegistration))
37 then
38  Total_deviceRegistrationcaptured countdeviceRegistration = new
        Total_deviceRegistrationcaptured();
39  countdeviceRegistration.setTotal($totaldeviceRegistration)
40  insert(countdeviceRegistration);
41  System.out.println( "Number of Incoming events: " + $totaldeviceRegistration);
42 end
43
44
45 declare CoveragePercentage
46  @Percentage: float
```

```
47 end
48
49 rule "Incoming coveragePercentage"
50  no-loop
51  salience 999
52  dialect "java"
53 when
54  $value: Float();
55   $eA: Total_eAlert_eAckcaptured() and  $dR: Total_deviceRegistrationcaptured();
56   from entry-point "DEFAULT"
57 then
58   value = Math.round( eA.total / dR.total )
59   CoveragePercentage percentage = new coveragePercentage();
60   percentage.setPercentage($value)
61   insert(percentage);
62 end
63
64
65 declare AVGCoveragePercentage
66  @Percentage: float
67 end
68
69 rule "AVGcoveragePercentage"
70  no-loop
71  salience 999
72  dialect "java"
73 when
74  $cp : CoveragePercentage()
75   $avg : Float()
76    from accumulate( CoveragePercentageItem ($value : value )
77    from entry-point "DEFAULT"
78    $avg : average( $value) )
79 then
80   AVGCoveragePercentage avgpercentage = new  AVGCoveragePercentage();
81   avgpercentage.setPercentage($avg)
82   insert(avgpercentage);
83 end
84
85
86 rule "checkSatisfiedProperty"
87  no-loop
88  salience 999
89  dialect "java"
90 when
91   $cov : AVGCoveragePercentage( Percentage >  70 )
92   from entry-point "DEFAULT"
93 then
94   System.out.println( "Satisfied Coverage Property");
95 end
```

Listing 2.2: Drools Code generated by Model2Code Transformer for the *coverage* property

2.5 Related Work

Defining expressive complex event specification languages has been an active research topic for years [54, 24, 37]. Among these languages, GEM [54] is a generalized and interpreted event monitoring language. It is rule-based (similar to other event-condition-action approaches) and provides a detection algorithm that can cope with communication delay. Snoop [24] follows an event-condition-action approach supporting temporal and composite events specification but it is especially developed for active databases. A more recent formally defined specification language is TESLA [37] that has a simple syntax and a semantics based on a first order temporal logic. The main focus of these works is the definition of a complex-event specification language, whereas our framework provides a more high-level and more specialized meta-model to define monitoring goals (functional properties and metrics definitions), which are then automatically translated into complex-event specifications. Some existing open-source event processing engines

are Drools Fusion [1] and Esper [2]. They can fully be embedded in existing Java architectures and provide efficient rule processing mechanisms. The authors of [37] also provide an efficient event detection algorithm by translating TESLA rules into automata.

The operators proposed into CPMM have been designed by taking into account the event composition operators addressed in GEM [54] and Drools Fusion [1]. Specifically, GEM allows for specifying high-level events in terms of combinations of lower-level events. It assumes an event as an happening of interest, which occurs instantaneously at a specific time. The event operators defined in GEM allow for defining a composite event in terms of a combination of primitive events and other composite events. In Drools Fusion the events are special entities that represent an important change of state in the application domain; they have several characteristics, like being usually immutable, having strong temporal constraint and relationships. The set of operators used in Drools Fusion are temporal operators and allow for modeling and reasoning over temporal relationship between events. The *EventType* model defined in the current version of CPMM combines features of both languages and in addition presents new features not included in the considered languages as we showed in Section 2.2.2.

Another related research area, addressing QoS modeling, focuses on ontologies that allow for the definition of QoS with rich semantic information [53]. In particular, in [53] 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) [62] 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 [53], the proposed QoS model concentrates on QoS knowledge representation rather than a language to specify QoS, thus any appropriate QoS specification language can be used on top of this model. Differently from this approach, CPMM allows for the specification of non functional properties. In the future the integration of the two approaches can be investigated and provided.

The proposed meta-model is used to instruct the GLIMPSE monitor about non-functional properties to be checked at run-time. Other monitoring frameworks exist, that address mostly the monitoring of performance in the context of system management [11, 57, 3]. Among them, Nagios [11] offers a monitoring infrastructure to support the management of IT systems spanning network, OS, applications; Ganglia [57] is especially dedicated for high-performance computing and is used in large clusters, focusing on scalability through a layered architecture whereas the Java Enterprise System Monitoring Framework [3] deals with web-based and service-driven networks solutions.

Finally, the work in [65] has several similarities with our approach, concerning the conceptual modeling of non-functional properties. However it is more specifically focused on measurement refinement, whereas our work targets a more comprehensive scope for modeling and transformation. In future work we plan to look closely at this model to possibly incorporate some of its refinements.

2.6 Conclusions and Future Work

We proposed in this chapter the latest advances to the CONNECT Property Meta-Model, as the basis of the CONNECT model-driven infrastructure for run-time monitoring. The monitoring configuration is automatically executed by parsing the models of the properties of interest. To allow for automatization, such models must conform to a suitable meta-model. In this chapter we presented: *i)* the Y3 improvements we embedded into CPMM to better specify properties and complex events; and, *ii)* the model-driven monitor configuration, combining CPMM and GLIMPSE. As a proof of concept, we showed the application of the model-driven infrastructure for run-time monitoring to the Terrorist Alert CONNECT scenario.

There are various directions for future work. First, some meta-model parts need to be refined to address trust. In particular, *i)* the QualitativePropertyDefinition meta-class should be refined by defining a suitable language (meta-model) that allows for the specification of complex properties; this work is partially done in Chapter 5 of this deliverable and will be integrated in near future; *ii)* the whole definition of the ActionBaseExpression introduced to model trust at the moment is not completely defined. Since, the meta-model only allows for the specification of the transition (or action)-based properties, we plan to extend the meta-model with state-based properties that are specific properties expressed on the application internal state. Moreover, we plan to evaluate the use of the powertype concept [43] in CPMM by trying to

achieve a tradeoff between the clearer modeling aspects introduced by it and the easy understanding of CPMM by not-expert users.

We must complete the implementation of the Model2Code transformations to automatically derive the Drools rules needed to configure GLIMPSE. For example, we could address the reliability and performance issues of the proposed monitoring framework when a lot of data are generated, providing a comparison of GLIMPSE with similar existing approaches. A more complete testing and validation of the devised Model2Code transformations should be carried out.

Finally, as a longer term goal beyond the lifetime of CONNECT, we aim at a broader research roadmap, which we refer to as a *Property-Driven Software Engineering Approach*. In line with previous work in [65], this roadmap foresees an enhanced model-driven software engineering approach where models of non-functional properties, such as the ones we specify in CPMM, become first-class entities. The specified properties should be seamlessly monitored and enforced, throughout the lifetime of engineered systems. The approach we have devised in CONNECT should then become part of such more general approach by providing automated procedures (in form of ModelToModel or ModelToText transformations) that, from such CPMM property models, produce suitable inputs for the analysis and enforcement tools.

3 Dependability and Performance in CONNECT

Dependability and performance are CONNECTability attributes of major concern to many networked systems willing to communicate. It is therefore important to be able to assess in CONNECT whether a synthesized CONNECTor meets required levels of satisfaction for these attributes. To this purpose, a probabilistic model-based approach is adopted. In general terms, stochastic model-based 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, to execution time and to transmission time); (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. 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. As already motivated in D5.1 [26], in CONNECT we consider as evaluation techniques both state-based stochastic methods and stochastic model checking methods. We experimented both 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 the second year, the 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 preliminary architecture of a Dependability&Performance Analysis Enabler, called DEPER, that supports both state-based stochastic methods evaluation and stochastic model checking techniques. DEPER interacts with other Enablers in the CONNECT architecture (as better detailed in the next section) to get inputs for its evaluation activity and then applies automated steps to implement the overall evaluation process. Preliminary prototypes for the two approaches, based on Möbius and PRISM assessment tools respectively, have been also developed (a first release for the former has been included in the Appendix-Prototypes). The complementarity of the two approaches has been also shown through the analysis of the CONNECT "Terrorist Alert" scenario from D6.2 [29], used throughout Deliverable D5.2 [28] for demonstration of the WP5 developed approaches.

The activity of this third year concentrated on consolidating and extending the DEPER Enabler and related prototype implementations, as described in the next sections of this chapter.

3.1 DEPER Enabler Overview

The DEPER Enabler provides support to the definition of a CONNECTor that allows NSs to interact with a given level of dependability and performance properties, if such non-functional requirements have been expressed by the involved NSs.

The architecture of DEPER, already defined during the second year, is shown in Figure 3.1. We recall that DEPER has been conceived as a general framework for automated assessment of dependability and performance metrics, which can accommodate a number of different model-based stochastic methods (each one implemented through an appropriate Analysis Engine). To this purpose, two modules have been defined implementing: i) an initial *Selector* functionality, in charge of selecting one (or more than one) Analysis Engine(s) in accordance to some defined criteria, and ii) a final *Aggregator* functionality, in charge of determining the analysis results to be provided in output to the *Synthesis* Enabler, in case more Analysis Engines have been activated on a CONNECTed system specification. Both Selector and Aggregator modules have not been further developed during this third year, so they are not dealt with in this deliverable (more details about them are in D5.2 [28]).

Currently, DEPER accommodates both the stochastic state-based and the stochastic model-checking approaches. In the following, we will describe separately the progress performed during the third year in both approaches (see next two sections).
To better understand how DEPER works, we first overview how it is positioned in the CONNECT architecture by briefly recalling its relationships with the other Enablers.

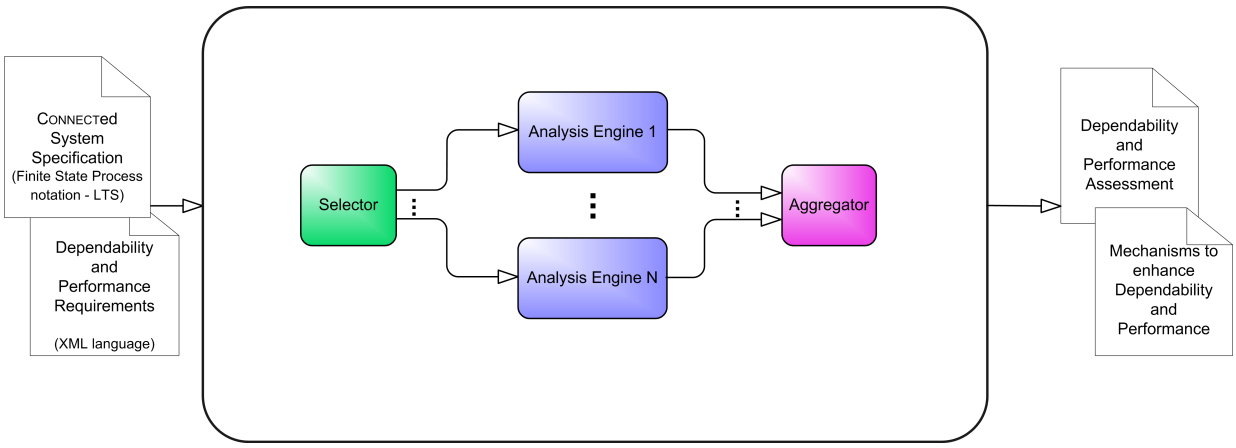


Figure 3.1: Architecture of the Dependability&Performance Analysis (DEPER) Enabler

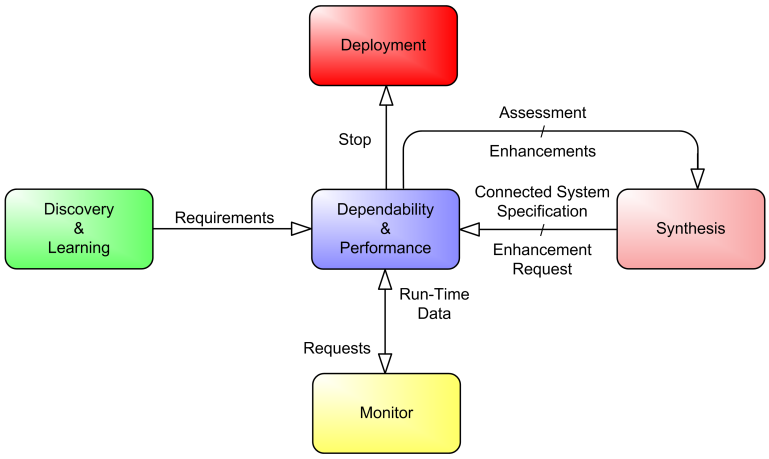


Figure 3.2: Input-Output Relations between DEPER and the Other Enablers

3.1.1 DEPER interaction with other Enablers in the CONNECT architecture

In Deliverable D5.2 [28], the input-output relations of DEPER with other Enablers of the CONNECT architecture have been already discussed. During this third year, we have revised and consolidated them, also considering the refinement of the overall CONNECT architecture as detailed in D1.3 [30]. Such input-output relations are shown in Figure 3.2. Although full description of DEPER interfaces with the other Enablers are in deliverable D1.3 [30], we briefly recall them in the following for the sake of completeness.

Interaction with *Discovery* and *Learning.* These Enablers gather information on the NSs. Specifically, Discovery discovers mutually interested NSs, and retrieves information on the specification of their interfaces. Learning then completes the specification through a learning procedure (e.g., via model-based testing) when NSs do not provide a sufficient description of their behavior. DEPER receives from them the NSs behaviour specifications and information about the dependability and performance requirements related to the NSs.

Interaction with *Synthesis.* This Enabler performs the dynamic synthesis of mediating CONNECTors to enable interoperation among NSs willing to interact. DEPER receives from this Enabler: i) the specification of the CONNECTed system (expressed in Labelled Transition Systems), annotated with non-functional information necessary to build the dependability/performance model of the CONNECTed system; ii) the request to attempt enhancement of the CONNECTor specification, by adopting mechanisms to contrast depend-

ability or performance deficiencies. In turn, DEPER sends to this Enabler the results of the dependability and performance assessment and, in case an enhancement request has been issued by Synthesis, the mechanism to embed in the CONNECTor specification to satisfy the requirements, if any.

Interaction with *Monitor.* This Enabler becomes operational when the CONNECTor is deployed and continuously monitors the deployed CONNECTor to update the functional and non-functional specification of the CONNECTor with run-time data. DEPER sends to this Enabler requests to monitor specific events of interest to dependability and performance assessment and receives from it observed values each time those events are observed during the CONNECTor executions along time.

Interaction with *Deployment.* The Deployment Enabler is in charge to deploy a synthesized CONNECTor. It takes the concrete LTS models, as provided by the Synthesis Enabler, and constructs a CONNECTor. The created CONNECTor, together with an informative description, is then added to a repository of already built CONNECTors managed by the Discovery Enabler, to be possibly reused to address future interoperability requests. When the dependability and performance requirements are no longer satisfied by a deployed CONNECTor, due to evolution of the NSs or change in the environment, DEPER alerts the Deployment Enabler, which will take an appropriate action to cope with this situation (typically, it should stop the CONNECTor's execution).

In the following, the interface specification between Synthesis and DEPER Enablers is shown as an example.

```
public interface DePerAnalysis{
/**
* Send a dependability and performance analysis request on the specified Connector.
* @param The FSP model representing the LTSs of NSs and Connector
* @param Deper address
*/
public void sendFSP(String fspSpecification, String deperAddress);

/**
* Send the Non-Functional specifications of the Connector.
* @param The XML model of the Non-Functional specifications
* @param Deper address
*/
public void sendNonFunctionalSpec(String nonFunctionalSpecification, String deperAddress);

/**
* Send the Metrics of the specified Connector.
* Expressions that describe how to obtain a quantitative assessment of the properties of
* interest.  They are expressed in terms of transitions and states of the LTS specification
* of the Networked Systems.
* @param The XML model of the Metrics of the Connector
* @param Deper address
*/
public void sendMetrics(String metrics, String deperAddress);

/**
* Send the guarantess of the specified Connector.
* Guarantees are boolean expressions that are required to be satisfied on the metrics.
* @param The XML model of the Guarantees of the Connector
* @param Deper address
*/
public void sendGuaranteesSpec(String guarantees, String deperAddress);
```

```
/**
* Send an enhancement request of the specified Connector that does not meet the dependability
and performance requirements.
* @param Connector ID
* @param Deper address
*/
public void enhance(String connectorId, String deperAddress);
}
```

3.2 Stochastic Model-based Analysis Engine

The Stochastic Model-based Analysis Engine of DEPER, already outlined during the second year of the project, is logically split into five main functional modules (see Figure 3.3): Builder, Analyser, Evaluator, Enhancer and Updater. The role of each module is briefly presented in the following. In the next Section we concentrate on the description of the modules which underwent major re-definition and extensions during this third year (Evaluator, Enhancer and Updater).

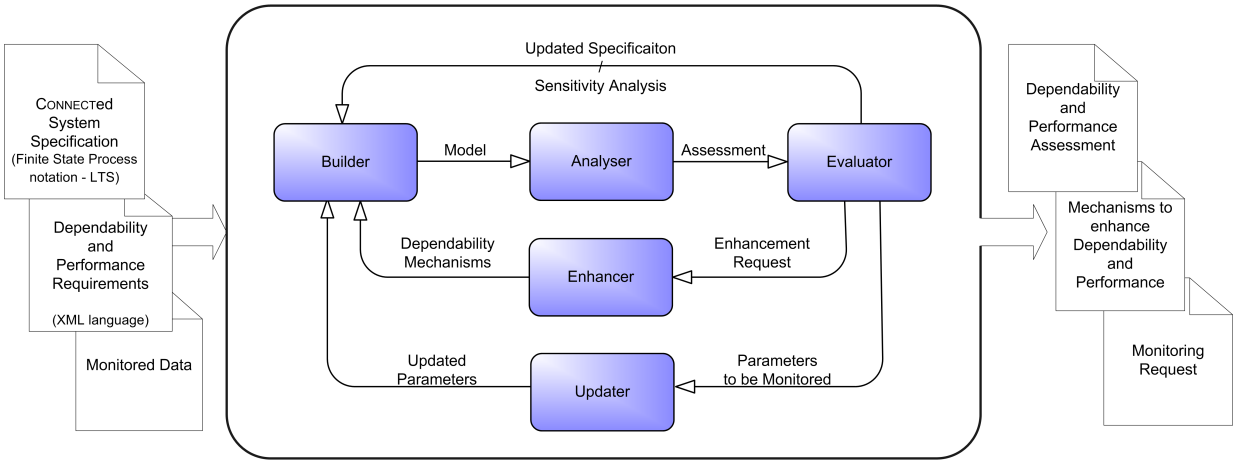


Figure 3.3: Architecture of the Stochastic Model-based Analysis Engine of DEPER

Builder

The Builder module takes in input the specification of the CONNECTed system from Synthesis. This specification is given with LTSs annotated with non-functional information necessary to build the dependability and performance model of the CONNECTed system. Annotations include, for each labelled transition, the following fields: *time to complete*, *firing probability*, and *failure probability*.
The module produces in output a dependability and performance model of the CONNECTed system suitable to assess the given dependability and performance requirements. Such model 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 properties required by the Networked Systems from Discovery/Learning. These requirements are expressed as *metrics* and *guarantees*. Metrics are arithmetic expressions that describe how to obtain a quantitative assessment of the properties of interest of the CONNECTed system. They are expressed in terms of transitions and states of the LTS specification of the Networked

Systems. Currently we are using the XML notation, but we plan to fully adopt the CPMM meta-model presented in Chapter 2 to describe the metrics, and in the last year we will develop a translator from CPMM to the format required by Möbius. Guarantees are boolean expressions that are required to be satisfied on the metrics. The module extends the received model with reward functions suitable to quantitative assessment of the metrics of interest. Then, it makes use of a solver engine to produce a quantitative assessment of the dependability and performance metrics.

Evaluator

The Evaluator module reports to Synthesis if the CONNECTed system satisfies the dependability and performance requirements provided by Discovery/Learning. If the requirements are satisfied, Evaluator reports to Synthesis that the CONNECTor can be successfully deployed, and reports to Updater the aspects that must be observed for the CONNECTor that is going to be deployed, e.g., transition durations, and probability of transitions failure. Instead, in the case of requirements mismatch Evaluator sends a warning message to Synthesis, and may receive back a request to evaluate if enhancements can be applied to improve the dependability or performance level of the CONNECTed system.

Enhancer

The Enhancer module is activated by 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 with indications about how to select the dependability mechanism to try and to which elements of the original model the mechanism has to be applied. Then, it performs the following actions: (i) selects the dependability mechanisms that can be employed, following the indications by Evaluator; (ii) triggers a new analysis round by instructing the Builder module on the application of the selected dependability mechanism in the CONNECTed system model.

Updater

The Updater module interacts with the Monitor Enabler to refine the accuracy of model parameters through run-time observations, obtained through a continuous monitoring activity. Therefore, it provides adaptation of the pre-deployment analysis to cope with changes in, or inaccurate estimates of, model parameters.

3.3 Detailed Description of DEPER Modules Developed During the Third Year

As already anticipated, during the third year the activity mainly concentrated on the Enhancer and Updater modules, and on extending the Evaluator module to properly manage and trigger the functionalities of the previous two.

3.3.1 Detailed description of Evaluator

The role of Evaluator, already introduced above, consists in checking the analysis results on dependability and performance metrics with the requested guarantees . Upon revealed mismatch, Evaluator may receive a request from the Synthesis Enabler 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, 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 to contrast failure modes affecting performance and/or dependability metrics (e.g., a message retransmission technique).

3. Apply a combination of the previously mentioned enhancements.

During this third year, we have concretely defined and developed enhancements through dependability mechanisms, which are mainly in charge to the Enhancer module. However, Evaluator prepares the activation of Enhancer, so significant new effort has been devoted to this module. Specifically, towards Enhancer, Evaluator is in charge of:

i) providing indications helpful to select a dependability mechanisms among the (possibly many) available. A precise identification of the most suitable mechanism could be actually possible by exploiting the results of a sensitivity analysis which would help understanding which are the model parameters (namely, the time to complete a transition and its failure probability) mostly impacting on the metrics evaluation, so as to include primarily dependability mechanisms capable of limiting their effects. Although feasible in an advanced version of the DEPER Enabler, in the current version we are working on a much simpler criterion, but still reasonably sound: if the metric under evaluation and not satisfying the required value (that is, the guarantee) is a performance-related metric, then a dependability mechanism suitable to cope with malfunctions in the time domain is preferred, while a mechanism suitable to deal with value failures is chosen in case of dependability-related metrics. In a more structured vision, the dependability mechanism(s) suitable to improve on a given failure event are determined through an ontology of dependability mechanisms, such as that reported in [4]. The definition of such an ontology is posponed as future work.

ii) identifying the elements of the dependability or performance model to which the selected mechanism has to be applied. Again, a careful identification of the weak elements would come out from a sensitivity analysis, which is unfortunately costly to perform. A simpler strategy would be the following. First, identify the model elements whose malfunction would have, on average, higher impact than the others on the metric under assessment. Dependability and performance models of the CONNECTed system consist of the representation of communications between NS1 and the CONNECTor, and between the CONNECTor and NS2. These communications are characterized by a failure rate and by a time to complete. Roughly, it could be assumed that dependability-related metrics are mainly influenced by the failure probability (or rate), while the performance-related ones are mainly influenced by the time to complete the transmission. Therefore, the choice of the model elements is made among those representing communications with the highest failure probability in case of dependability-related metrics, and among those representing communications with the highest time to complete in case of performance-related metrics. This is actually the approach we have followed when implementing the prototype.

In view of the synergic cooperation with the monitoring infrastructure, this module also informs the Updater module, which is in relationship with the Monitor Enabler, about the model parameters for on-line observation. Further details are provided when describing the Updater module.

3.3.2 Detailed description of Enhancer

As already introduced, the Enhancer module is in charge of assessing whether the CONNECTor can be improved trough dependability mechanisms, when the guarantees are not satisfied and Synthesis makes a request to investigate on enhancements. Therefore, this module performs a form of pre-deployment adaptation to try to overcome deficiencies of the CONNECTor specification as revealed by the analysis.

To accomplish its task, Enhancer is equipped with models representing basic dependability mechanisms suitable to contrast two typical classes of failure modes that may happen during interactions: *timing failures*, in which networked systems send messages at time instants that do not match an agreed schedule, and *value failures*, in which networked systems send messages containing incorrect information items. We consider timing failures of type omission, i.e., late messages are always discarded, and value failures that cause a networked system to respond within the correct time interval but with an incorrect value.
It is triggered by Evaluator, that receives the enhancement request from Synthesis. Since Enhancer does

not have knowledge of the metric under analysis, it is instructed by the Evaluator module with indications about: a) how to select the dependability mechanism model to try and b) to which elements of the *basic* model (that is, the model originally developed without any dependability enhancement) the mechanism has to be applied. The policies adopted for a) and b) have been already sketched when describing Evaluator.
Then, it performs the following actions: (i) selects the dependability mechanism's model that can be employed; (ii) instructs the Builder module on the application of the selected dependability mechanism model in the *basic* CONNECTed system model, to all or to a subset of the model elements indicated by Evaluator, to be then solved anew by Analyser. At the end of this new analysis, Evaluator verifies whether the enhanced CONNECTor fulfills the dependability and performance requirements. If yes, Evaluator informs the Synthesis Enabler about the mechanism to add to the CONNECTor design and the DEPER support to the design of this CONNECTor is completed. Otherwise, Enhancer makes a further attempt with the next dependability mechanism's model (if available), according to some internal pre-defined policies about how to rank the available mechanisms models and about how to apply them to model elements provided by Evaluator, and a new cycle with Builder, Analyser and Evaluator is repeated. This loop ends either when a successful mechanism is found, or when all the mechanisms are exhausted.

This module has been significantly extended during this third year of CONNECT, with the definition of a library of dependability mechanism models, specified using the Stochastic Activity Networks (SAN) [67] formalism. The models have been developed according to three basic rules that allow to simplify the automated procedure for embedding the mechanism in the specification of the synthesised CONNECTor: (i) each model has an initial place, `s0`, whose tokens enable the first activity of the model; (ii) each model has a final place, `s1`, which contains tokens whenever the last activity of the model completes; (iii) the overall number of tokens in `s1` is always less or equal to the number of tokens in `s0`. With the above rules, the behaviour of the model can be seen as an *enhanced activity*, and can be directly used to replace any activity that moves tokens between two places in the specification of the CONNECTor (the basic semantics of an activity is always preserved).
In the following, a brief description of each of the five developed mechanisms is provided.

Retry Mechanism. The retry mechanism consists in re-sending messages that get corrupted or lost during communications, e.g., due to transient failures of communication links. This mechanism is widely adopted in communication protocols, such as TCP/IP [20] for enabling reliable communication over unreliable channels. A typical implementation of the retry mechanism uses time-outs and acknowledgements: after transmitting a message, the sender waits for a message of the receiver that acknowledges successful communication. If the acknowledgement is not received within a certain time interval (defined in accordance with the performance requirement), the sender assumes that the communication was not successful, and re-transmits the message.

The stochastic activity network shown in Figure 3.4 is the model representing this mechanism. On the sender side, the mechanism creates a message re-transmission policy for re-sending the message at most N times; on the receiver side, the mechanism creates a policy for avoiding duplicated reception of messages and for sending acknowledgements.

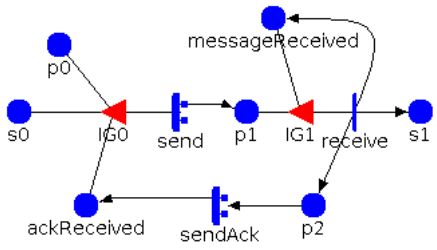


Figure 3.4: Retry Mechanism

In the model, all places initially contain zero tokens, except `p0`, which contains N tokens, where N is a model parameter representing the maximum number of re-transmissions. Activity `send` is enabled when the conjunction of the following conditions is true: `p0` and `s0` contain at least one token, and `ackReceived`

contains zero tokens. When activity send completes with success (case 0, with probability $pr0$), a token is removed from s0 and p0, and the marking of p1 is incremented by one. Activity receive is enabled when p1 contains at least one token and messageReceived contains zero tokens. When activity receive completes, a token is moved to s1, and the marking of p2 and messageReceived is incremented by one. A token in p2 enables activity sendAck, whose aim is to enable the receiving host notify the sender that the message has been successfully received. The sender stops re-transmitting the message as soon as it gets an acknowledgement that the message has been successfully received, or after N attempts.

Probing Mechanism. The probing mechanism exploits redundant paths and periodic *keep-alive* messages for enabling reliable communication in face of path failures. The basic idea is to continuously collect statistics on the characteristics of the communication channels, and to select the best channel on the basis of such statistics. This mechanism has been used for defining communication services with guaranteed delivery and performance levels, e.g., see Akamai's SureRoute [8] and reliable multi-cast protocols for peer-to-peer networks [72].

The stochastic activity network shown in Figure 3.5 is the model representing a probing mechanism that uses two redundant communication channels. The mechanism instruments the sender with a periodic channel probing functionality suitable to feed a monitoring system that collects statistics about the reliability level of the communication channels.

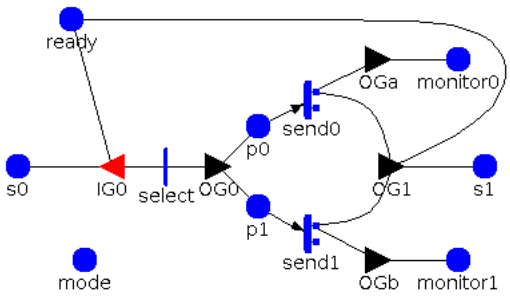


Figure 3.5: Probing Mechanism

In the model, place mode is a state variable that indicates the mode of operation of the mechanism, which can be either *probing mode* (mode contains zero tokens), i.e., the mechanism tests the characteristics of the communication channels through keep-alive messages, or *normal mode* (mode contains one token), i.e., the mechanism selects the best estimated channel for relaying messages. Initially, all places contain zero tokens, except ready, which contains one token.

When in normal mode, activity select is enabled when s0 contains at least one token, and ready contains one token. When select completes, one token is removed from s0 and ready, and send0 gets enabled if monitor0 has more tokens than monitor1 (send1 gets enabled in the other case). The number of tokens of monitor0 and monitor1 is proportional to the reliability level of the channels they are associated with. If send0 completes with success (case 0), then a token is added to s1 and ready. Similarly, when send1 completes with success, a token is moved to s1 and ready.

When in probing mode, the model behaves as follows: send0 and send1 have both the same rate $R0$ (while their case probabilities depend on the characteristics of the channels, which may vary over time). Activity select is enabled when ready contains one token; when select completes, ready contains zero tokens and activities send0 and send1 get enabled (by moving one token in both p0 and p1). When send0 completes with success (case 0), a token is added to monitor0. Similarly, when send1 completes, a token is added to monitor1. A token is moved to ready when both send0 and send1 complete.

Majority Voting Mechanism. Majority voting is a fault-tolerant mechanism that relies on a decentralised voting system for checking the consistency of data. Voters are software systems that constantly check each other's results, and has been widely used for developing resilient systems in the presence of faulty components. In a network, voting systems can be used to compare message replicas transmitted over different channels, see, for instance, the protocol proposed in [73] for time-critical applications in acoustic sensor networks.

The stochastic activity network shown in Figure 3.6 is the model representing a majority voting mechanism that uses three redundant communication channels. The mechanism replicates the message sent by the transmitting host over three channels. In this case, the mechanism is able to tolerate one faulty channel.

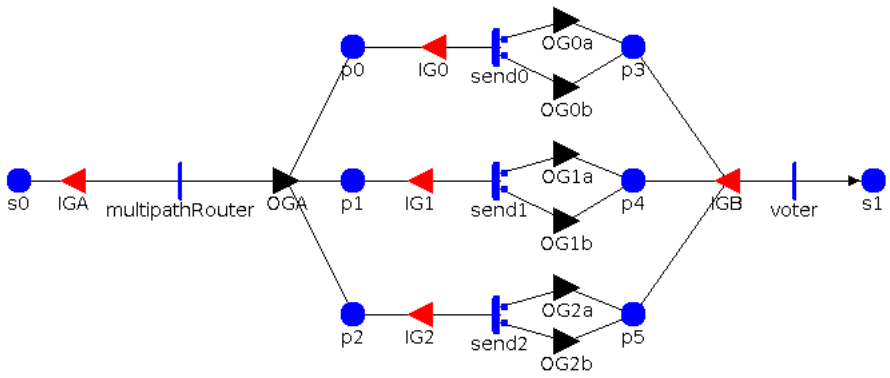


Figure 3.6: Majority Voting Mechanism

In the model, all places initially contain zero tokens. Activity `multipathRouter` gets enabled when `s0` contains a token. When such an activity completes, the token is removed from `s0`, and three send activities (`send0`, `send1`, `send2`) get enabled by moving tokens into places `p0`, `p1`, and `p2`. The number of tokens moved in such places encode the actual informative content of the message. When a send activity completes with success, the activity preserves the number of tokens (i.e., all tokens are moved forward into the next place). Activity `voter` gets enabled when the all sends complete (such activities will eventually change the marking of `p3`, `p4`, and `p5`). When activity `voter` completes, a token is moved into `s1` and all tokens in other places are removed.

Error Correction Mechanism. Error correction deals with the detection of errors and re-construction of the original, error-free data. A widely used approach for enabling hosts to automatically detect and correct errors in received messages is *forward error correction* (FEC). The mechanism requires the sender host to transmit a small amount of additional data along with the message. The mechanism has been used, for instance, in [68] for defining an overlay-based architecture for enhancing the quality of service of best-effort services over the Internet.

The requirement of additional data, to be transmitted by the sender along with the message, does not make this mechanism straightforwardly implementable in CONNECT, where a basic assumption is at the moment to make transparent to the involved NSs any means taken at CONNECTor level. However, we have developed a model for this mechanism and included it in the library for future usages, opening to a wider vision where NSs showing high criticality to dependability and performance requirements are internally equipped with measures to contrast communications malfunctions.

The stochastic activity network shown in Figure 3.7 is the model representing an error correction mechanism that uses two redundant communication channels. One channel is used to send the original message, and the other channel is used to send the error correction (EC) code. The receiver is instrumented with a filtering mechanism that checks and corrects messages.

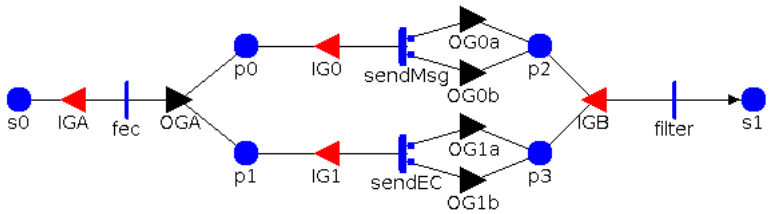


Figure 3.7: Error Correction Mechanism

Initially, all places contain zero tokens. When a token is moved into s0, activity fec gets enabled. When such an activity completes, a token is removed from s0, and activities sendMsg and sendEC get enabled by moving tokens into p0 and p1. The number of tokens moved in such places encode the actual informative content of the message. When sendMsg completes with success, all tokens in place p0 are moved into p2. Similarly, when sendEC completes, all tokens of p1 are moved into p3. Activity filter gets enabled when places p2 and p3 contain tokens. When activity filter completes, a token is moved into s1 and all tokens in other places are removed.

Security Mechanism. A typical way to enforce protection on a host is to decouple the host from the rest of the network. A host can, for instance, be protected from receiving unwanted traffic by creating a ring that selectively filters the incoming traffic. Similarly, the identity of a host can be protected by anonymising the host's messages through a set of intermediary hosts, denominated *proxies*. This mechanism has been used, for instance, in [51] and [9] for protecting hosts from denial-of-service attacks.

The stochastic activity network shown in Figure 3.8 is the model representing a security mechanism over a network with two intermediary hosts. The mechanism creates an anonymiser service that selects a channel with a certain probability, and forwards the message on such a channel. In the model, all places

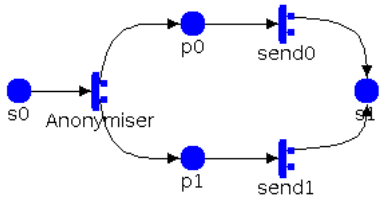


Figure 3.8: Security Mechanism

initially contain zero tokens. When a token is moved into s0, activity Anonymiser gets enabled. When such an activity completes, a token is moved from s0 either to p0 (with probability $pr0$), or to p1 (with probability $pr1$), and either send0 or send1 gets enabled. When a send activity completes, a token is moved in s1.

3.3.3 Detailed description of Updater

As already introduced, the Updater module interacts with the Monitor Enabler to refine the accuracy of model parameters through run-time observations, in order to adapt to changes that are revealed during the CONNECTed system executions. Necessity to revise the non-functional values used in the pre-deployment analysis at CONNECTor design time is mainly due to two possible causes: i) limited knowledge of the NSs characteristics acquired by Discovery/Learning Enablers; ii) evolution along time of the NSs, as naturally accounted for in the CONNECT context.
Updater receives inputs from both internally to DEPER (by the Evaluator module) and externally (by the Monitor Enabler).

For each CONNECTor ready to be deployed, the Updater module receives from the Evaluator module the model parameters to convey to the Monitor Enabler for run-time observations. The parameters received from Evaluator are obtained through a sensitivity analysis that aims to understand which elements of the CONNECTed system have highest impact on the dependability and performance measure.
From the Monitor Enabler, the Updater module receives a continuous flow of data of the parameters under monitoring relative to the different executions of the CONNECTor. Accumulated data are processed through statistical inference techniques. If, for a given parameter, the statistical inference indicates a discrepancy between the on-line observed behaviour and the estimated value used in the pre-deployment model, a new analysis is triggered by instructing the Builder module to update the CONNECTed system model. To improve on efficiency, the Updater module could receive indications not only on the parameters to be monitored, but also on a range of values for each of them, thus setting the variation interval within which the already performed analysis is subject to negligible modifications. Of course, the efficiency gained in avoiding repetitions of the analysis triggered by Updater has to be compared with the additional

effort necessary at pre-deployment time to assess the ranges for the parameters values via sensitivity analysis. In the current prototype implementation of DEPER, range values have been not accounted for and left as a future extension of the enabler.

Then, should the new values determined via inference techniques on on-line collected data differ from the originally considered values, a new analysis phase is triggered to understand whether the dependability and performance requirements are still satisfied. If the CONNECTed system does not meet anymore the stated requirements, the CONNECTor under usage is no more adequate and the intervention of Synthesis is requested. A cycle with Enhancement is then possibly started, as at pre-deployment time, upon request by Synthesis to investigate improvements through dependability mechanisms. Also, in this case of non satisfaction of requirements, the CONNECTor execution should be stopped by the Deployment Enabler and consequently the Monitor Enabler does not have to observe anymore its behaviour. Therefore, messages are sent by Updater to both Enablers to inform them that the CONNECTor is no more adequate and consequent actions need to be taken. If, instead, the new analysis reveals that the requirements are still fulfilled, Updater is informed of this positive outcome and the CONNECTor continues to be used and kept under observation by Monitor.

As reported in [70], methods of statistical inference applied to a collection of elements under investigation (called *population*), allow to estimate the characteristics of the entire population. In our case, the collection of values relative to each parameter under monitoring constitute a subset (called *sample*) of the population to which such techniques are applied.
Parameter estimation is the process by which it is possible to get information, from the observed sample, in order to assign a value (*point estimate*) to the parameter or a set of values (*interval estimate*). The sampling process represents a significant problem, because it is unknown which is the representative sample size (*n*). It seems intuitive that the precision of the estimates increases with *n*. On the other hand increasing *n* could lead to excessive increase of time and costs.

The methods of parameter estimation rarely produce a point estimate of the desired parameter which coincides with the actual value. Therefore, it is often preferred to find an interval estimate, called *confidence interval* Δ, which contains the real value of the parameter under analysis with a *confidence level* α.
Suppose we choose a value of $(1-\alpha)$ such that:

$$P(\overline{\theta} - \epsilon < \theta < \overline{\theta} + \epsilon) = (1-\alpha)$$

we say that the random interval $A(\theta) = (\overline{\theta} - \epsilon, \overline{\theta} + \epsilon)$ is a $100(1-\alpha)\%$ confidence interval for the parameter θ and $(1-\alpha)$ is called *confidence coefficient*.
Let $X_1, X_2, \ldots, X_n$ be a random sample, from a population with mean μ and variance σ^2, then $\overline{X}(n)$ and $S^2(n)$ are the *sample mean* and the *sample variance* respectively.

$$\overline{X}(n) = \frac{1}{(n)} \sum_{i=1}^{n} X_i$$

$$S^2(n) = \frac{1}{(n-1)} \sum_{i=1}^{n} (X_i - \overline{X}(n))^2$$

Under the hypothesis of independent and identically distributed observations and based on the central limit theorem for a value of $n > 30$ the random variable

$$Z = \frac{\overline{X}(n) - \mu}{\sqrt{S^2(n)/n}}$$

is standard normal.
If we want to obtain a $100(1-\alpha)\%$ confidence interval for the population mean μ, we have:

$$P(-z_{\alpha/2} < Z < z_{\alpha/2}) = 1 - \alpha$$

where the value of $z_{\alpha/2}$ is tabulated. Now

$$P\left(-z_{\alpha/2} < \frac{\overline{X}(n) - \mu}{\sqrt{S^2(n)/n}} < z_{\alpha/2}\right) = 1 - \alpha$$

$$P\left(\overline{X}(n) - z_{\alpha/2}\frac{S(n)}{\sqrt{n}} < \mu < \overline{X}(n) + z_{\alpha/2}\frac{S(n)}{\sqrt{n}}\right) = 1 - \alpha$$

Which means that the sample mean $\overline{X}(n)$ deviates from the actual value no more than $\Delta = z_{\alpha/2}\frac{S(n)}{\sqrt{n}}$

The size n of the random sample affects the confidence interval, therefore it is possible to determine the value of n based on the confidence interval:

$$n \geqslant \left\lceil \left(\frac{z_{\alpha/2}S(n)}{\Delta}\right)^2 \right\rceil \tag{3.1}$$

When the sample size is relatively small ($n < 30$), we can use the Student *t* distribution.
So the inequality to get the $100(1-\alpha)\%$ confidence interval of μ becomes:

$$\overline{X}(n) - t_{n-1;\alpha/2}\frac{\sigma}{\sqrt{n}} < \mu < \overline{X}(n) + t_{n-1;\alpha/2}\frac{\sigma}{\sqrt{n}}$$

which represents a Student *t* distribution with $n-1$ degrees of freedom and where the value of $t_{n-1;\alpha/2}$ can be read from a table.
To evaluate the sample size n we encounter two difficulties:

1. S^2 is not known in advance;
2. $t_{n-1;\alpha/2}$, which can be read from a table, depends on n.

These difficulties can be solved by the following two points:

1. using an assumed value of the variance, indicated by S^{*2}, from pilot investigations;
2. using an iterative algorithm, to evaluate n using from time to time the degrees of freedom obtained at the previous step. The stop condition of the algorithm is reached when the result of two successive steps is the same.

Following this approach and considering fixed values of the confidence interval and confidence level, we are able to define the sample size of monitored parameter values collected from the Monitor Enabler necessary to apply statistical inference . In order to assess the sample size, the iterative Algorithm 1 has been used.

Algorithm 1 *Sample size*

```
n := ∞
while false do
    n' = ((2·t_{n;α/2})/Δ)^2 · S^{*2};
    if n == n' then
        true
    else
        n := n'
    end if
end while
return n
```

3.3.4 Implementation of the stochastic model-based analysis engine

The prototype implementation of the Stochastic Model-based Analysis Engine has been realized by adopting the Stochastic Activity Networks (SANs) [67] formalism and the Möbius [25] software tool, that provides a comprehensive framework for model-based dependability and performance evaluation of systems. Synthetic description of the SAN formalism and of Möbius were included in the second year Deliverable D5.2 [28] and are not reported here.

Figure 3.9: Explanation of the Graphical Symbols

The following Figures 3.10, 3.11, 3.12, 3.15 and 3.16, illustrate the activity diagram of the modules Builder, Analyser, Evaluator, Enhancer and Updater in the current implementation of DEPER (the explanation of the graphical symbols used in the graphics is in Figure 3.9).

Builder module. A first implementation of the Builder module was already completed during the second year, but an enhanced and extended version was produced during this third year (the module's activity diagram is in the Figure 3.10).

The improvements mainly consist in applying a more efficient technique in building the composed SAN model, which allows to better cope with the problem of states explosion in presence of multiple system actors represented in the model (for example, the number of guards under a commander in the Terrorist Alert scenario carried on during the second year). According to this technique, first a *reduced connected system* model is built where only one single atomic SAN model for each type of involved networked systems is represented; then, this reduced model is expanded to account for the required multiplicity, resulting in the *basic* CONNECTed system model.

The performed extension consists in the enrichment of the *basic* model, built the first time Builder is activated on a given connected system to be analysed through DEPER, with template models representing a dependability mechanism, according to the instructions received from the Enhancer module. Such instructions consist in the identification of the template model to adopt (representing an available dependability mechanism), and which are the elements of the *basic* model that need to be replaced with the chosen dependability mechanism. The extension with a dependability mechanism (arrow labeled *Next time build* in Figure 3.10) is performed only in case a request of enhancement is issued by the Synthesis Enabler upon an unsatisfactory feedback provided by DEPER on the original CONNECTor's design.

Analyser module. The implementation of the Analyser module, whose activity diagram is shown in Figure 3.11, did not undergo major modifications during this third year, being already rather well consolidated. Upon receiving the SAN connected system model from the Builder module, Analyser automatically derives the reward functions 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. The SAN model and the reward functions are fed to the Möbius solver for getting the analysis results, which are then conveyed to the Evaluator module.

Evaluator module. Evaluator is somehow the *core* module of the overall analysis engine, being in charge of coordinating several activities with other modules inside DEPER and of the relationships with the Synthesis Enabler. Its implementation has been significantly revised and extended during this third year, with respect to the partial one already available from the previous year.

On the left side of the activity diagram shown in Figure 3.12, it is reported the sequence of activities performed by this module after receiving the results of the analysis performed on the *basic* model of the CONNECTed system under evaluation. Upon successful match of the analysis results with the requested guarantees, a positive feedback is conveyed to the Synthesis Enabler, and instructions about model parameters to be monitored on-line, in order to improve the accuracy of the analysis with field data, are produced and transmitted to the Updater module. In the version currently implemented, all the model parameters are selected for run-time monitoring, but, for the sake of efficiency, only the most impacting ones on the metric under evaluation should be in general considered. In case the guarantees are not satisfied, we consider that the Synthesis Enabler always asks for enhancement of the CONNECTor design through dependability mechanisms. In consequence of this, Evaluator selects the model elements which

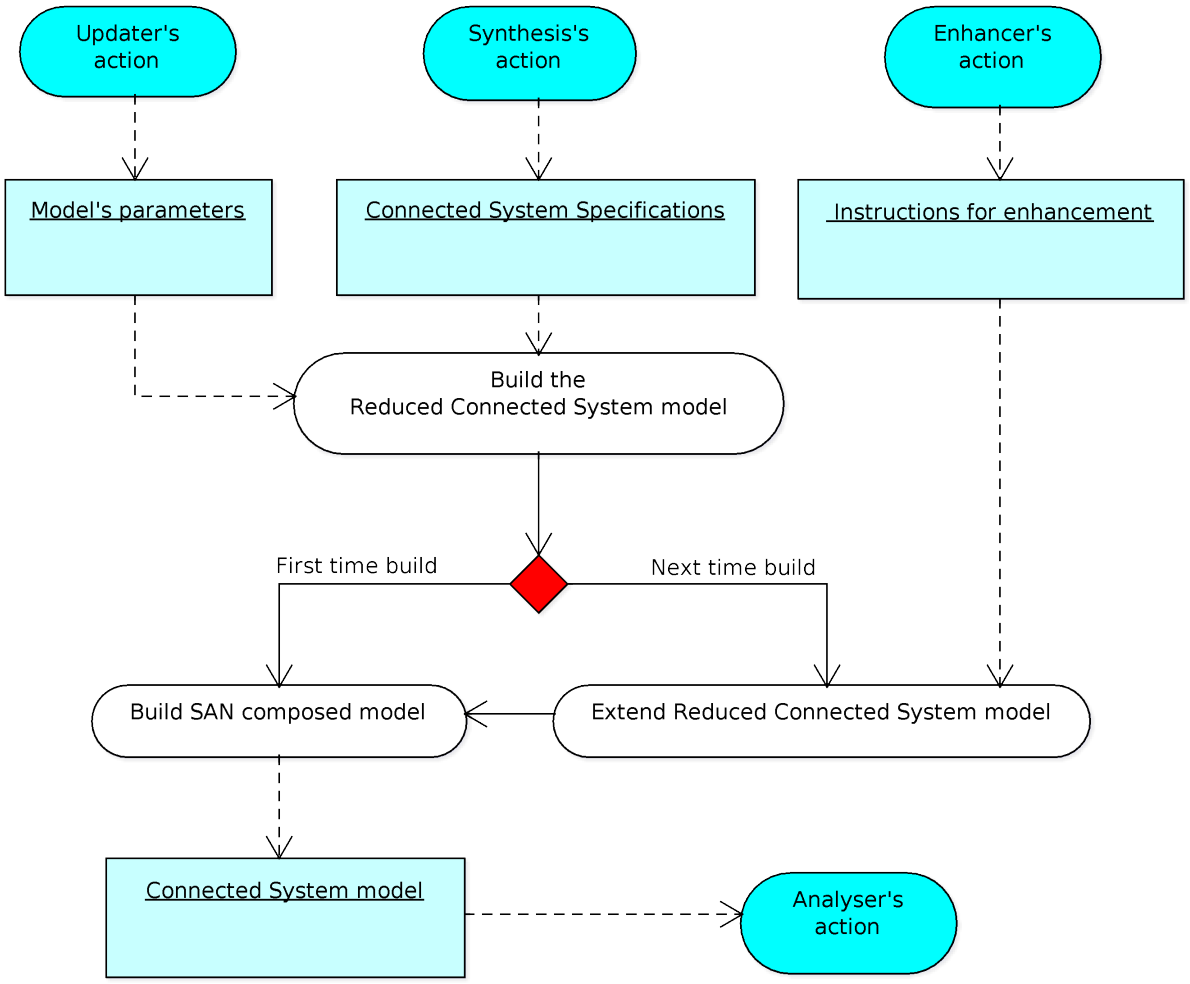


Figure 3.10: Activity Diagram of the Builder Module

need to be made more robust, and transmits them to the Enhancer module. This selection is currently made as follows. All the communication channels are selected but, depending on the kind of metric under assessment (whether dependability-related or performance-related), they are ranked according to the highest failure rate (in case of dependability-related) or the longest time to complete the transmission (in case of performance-related).

On the right side of the scheme in Figure 3.12, it is reported the sequence of actions performed by Evaluator upon receiving the results of successive evaluations of the connected system under analysis. It is distinguished the case where these successive evaluations are performed at pre-deployment time from the case where they occur after the deployment of the CONNECTor.

In the former case, the actions mainly consist in the loop started with the Enhancer module about the attempts for enhancing the CONNECTor. The activity diagram related to this branch is illustrated in Figure 3.13. The loop is interrupted either when Analyser provides results which satisfy the guarantees, or when all the dependability mechanisms available have been tried without success. Upon guarantees satisfaction, the Synthesis Enabler is informed about the dependability mechanism which led to success, in order to be embedded in the CONNECTor design before being deployed. Usual activities toward Updater are also performed. In addition, the Enhancer module needs to be informed that no further enhancement tries are necessary. If, instead, exhaustion of enhancement tries is reached without success, Synthesis is informed and no further analysis activities are performed on this CONNECTed system (we may think

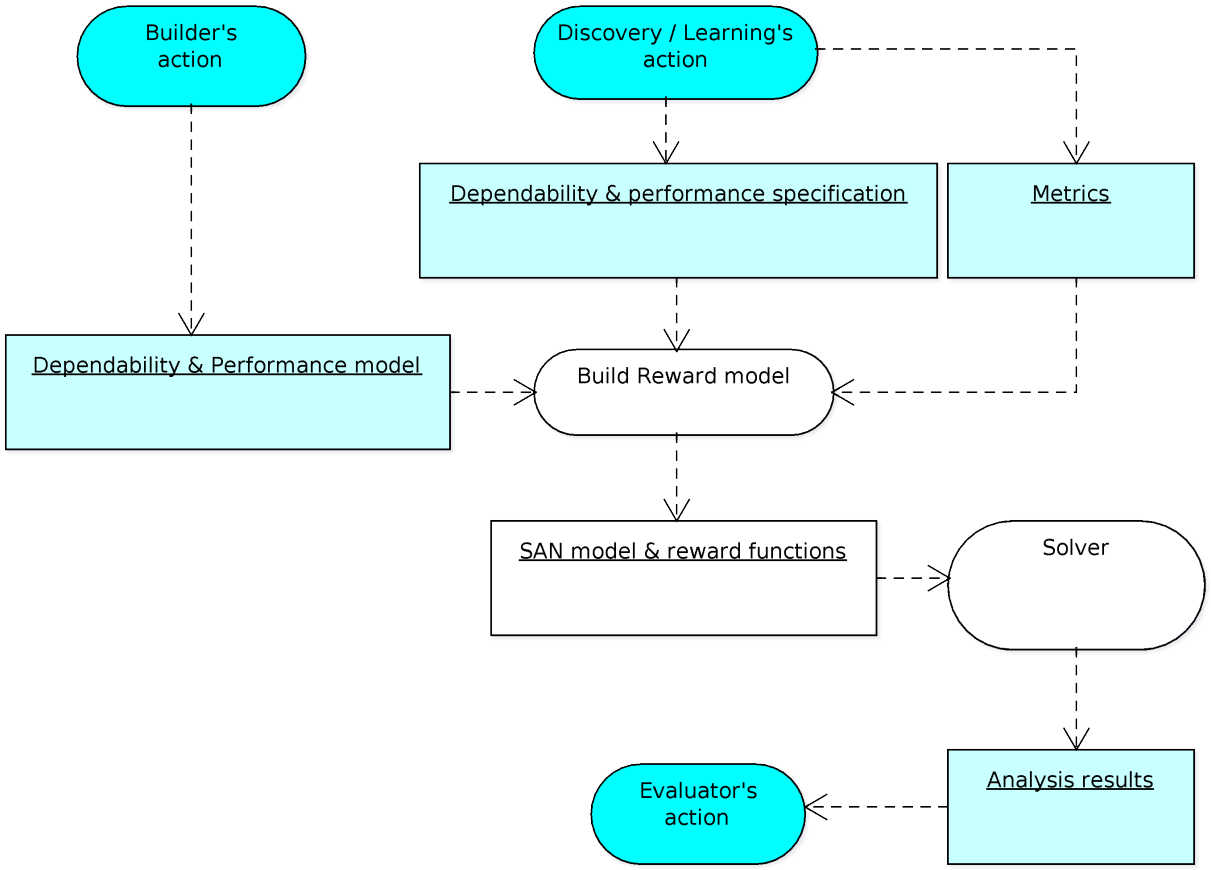


Figure 3.11: Activity Diagram of the Analyser Module

that at this point a new CONNECTor need to be synthesized, or some kind of negotiation with the involved networked systems is started about reducing the dependability and performance expectations).

In the latter case (illustrated by the activity diagram in Figure 3.14, actions also include communications to Updater about the results of the analysis performed with refined parameters values, so that Updater can in turn solicit actions towards Monitor and Deployment, should the new analysis reveal that the CONNECTor does not meet anymore the dependability and performance requirements. In this situation, a loop with Enhancement is also started, as a support to Synthesis in improving the CONNECTor's definition (which will however be considered a new CONNECTor after re-synthesis and deployment).

Enhancer module. Figure 3.15 illustrates the activities of the Enhancer module. This module, in charge of managing the dependability mechanisms' models developed for assessing their efficacy to cope with failures undermining the dependability and performance of the CONNECTor, has been completely implemented during this third year. The five dependability mechanisms, whose models have been already described in Section 3.2, have been implemented.

Enhancer is activated by Evaluator when enhancement of the CONNECTor under development is requested, upon deficiencies discovered through the analysis performed on its *basic* model. It remains active, by sequentially performing enhancement attempts among the set of available alternative dependability mechanisms' models, until an attempt results in satisfaction of the requested guarantees or all the attempts have been tried without guarantees satisfaction. The five available models are currently ranked almost randomly, with the retry mechanism as the first. A more suitable policy to rank them, in accordance with the kind of metrics under evaluation (whether dependability-like or performance-like) will be explored as future work. However, we implemented the possibility of selecting a subset of the model elements to be enhanced through the dependability mechanism and transmitted by Evaluator, so as to

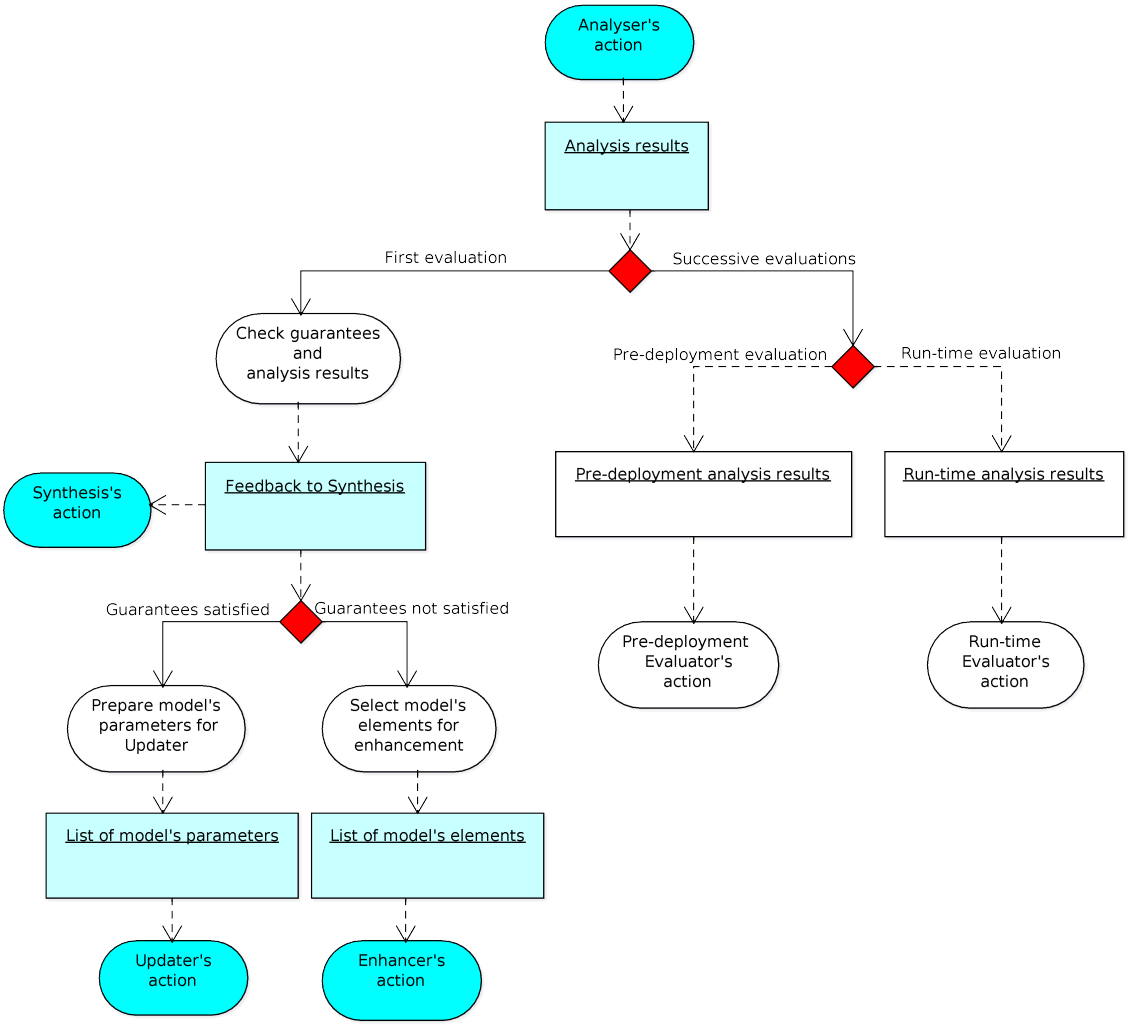


Figure 3.12: Activity Diagram of the Evaluator Module

have the possibility to gradually apply the mechanism to one, to a certain number of or to all the elements of the model. The selected dependability mechanism's model, as well as the indication concerning to which elements to apply the mechanism (*Instructions for enhancement* in the figure), are communicated to Builder, which properly enriches the *basic* SAN model and triggers a new analysis iteration. A final internal reset operation is performed at the end of the activities related to the CONNECTor under evaluation, to be ready to start a new enhancement campaign on the next CONNECTor requiring improvements. The *Enhancer warning* refers to a warning issued by Enhancer to Evaluator when the last enhancement try is issued: should this last try be not successful, Evaluator knows that no further attempt can be performed by Enhancer.

Updater module. The implementation of the Updater module has been developed during this third year. As described in Section 3.2, this module is in charge of updating the parameters of the SAN model based on real observed data coming from Monitor Enabler. Once the monitored data are received, the module updates the parameters of the model with new ones in order to allow a more refined analysis.

Figure 3.16 illustrates the activities of the Updater module. Updater is activated by the Evaluator module when the CONNECTor has been deployed. Evaluator sends to Updater a list of critical elements, that is the elements whose variations, due to initial uncertainty, could compromise the dependability and

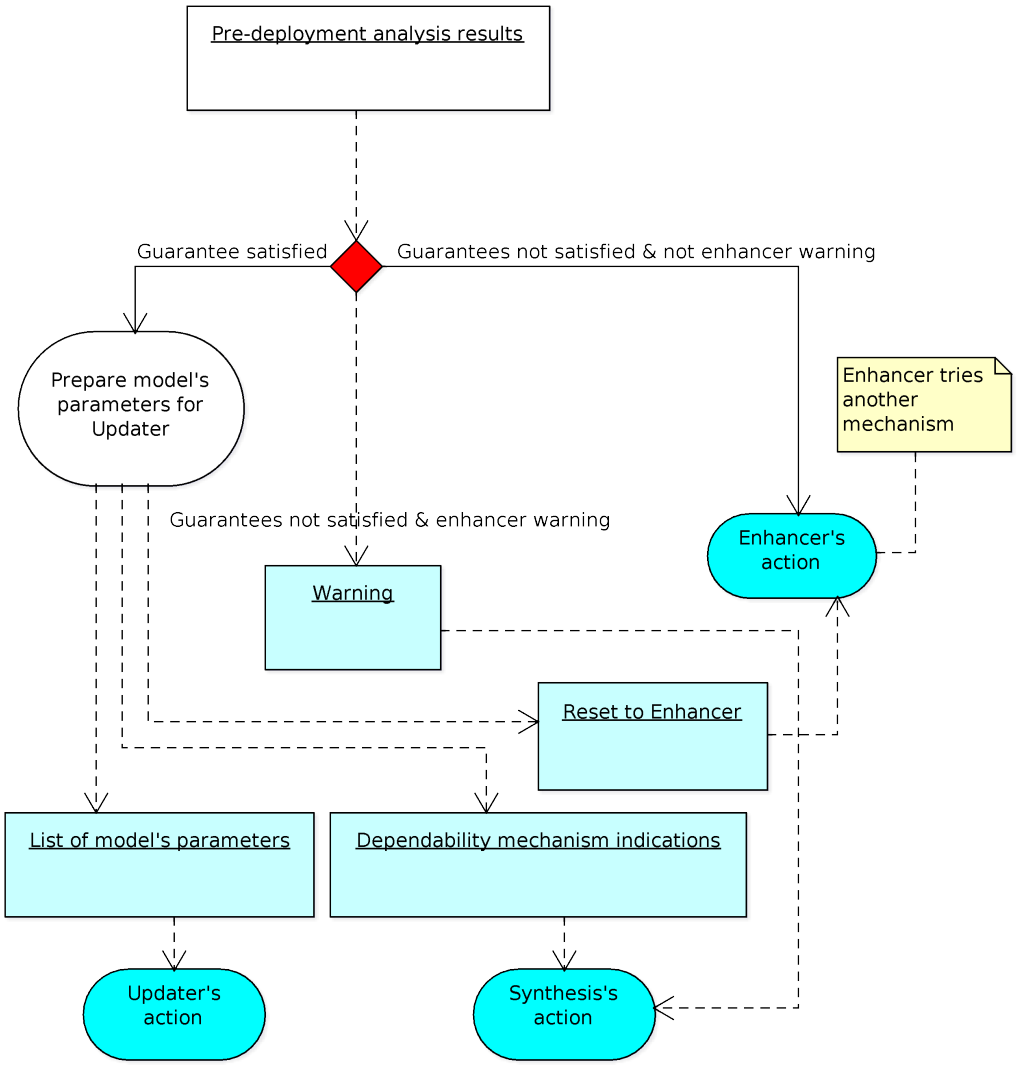


Figure 3.13: Activity Diagram of the Pre-Deployment Branch of the Evaluator Module

performance requirements, originally considered satisfied from the pre-deployment analysis. Updater sends a request to the Monitor Enabler in order to get actual values about the collection of the critical elements. It remains active and collects monitored data. When the statistical inference, applied on the collected data, reports that the sample is large enough to estimate the elements under investigation, the parameters of the model are eventually analysed.

If it is not found any discrepancy between the original and the monitored data, the Updater module keeps on collecting data from Monitor; this is useful because of the evolving and updating of the NSs. Otherwise, Updater sends the new values to the Builder module in order to update the model and a new analysis is performed, with Evaluator checking whether the dependability and performance require-

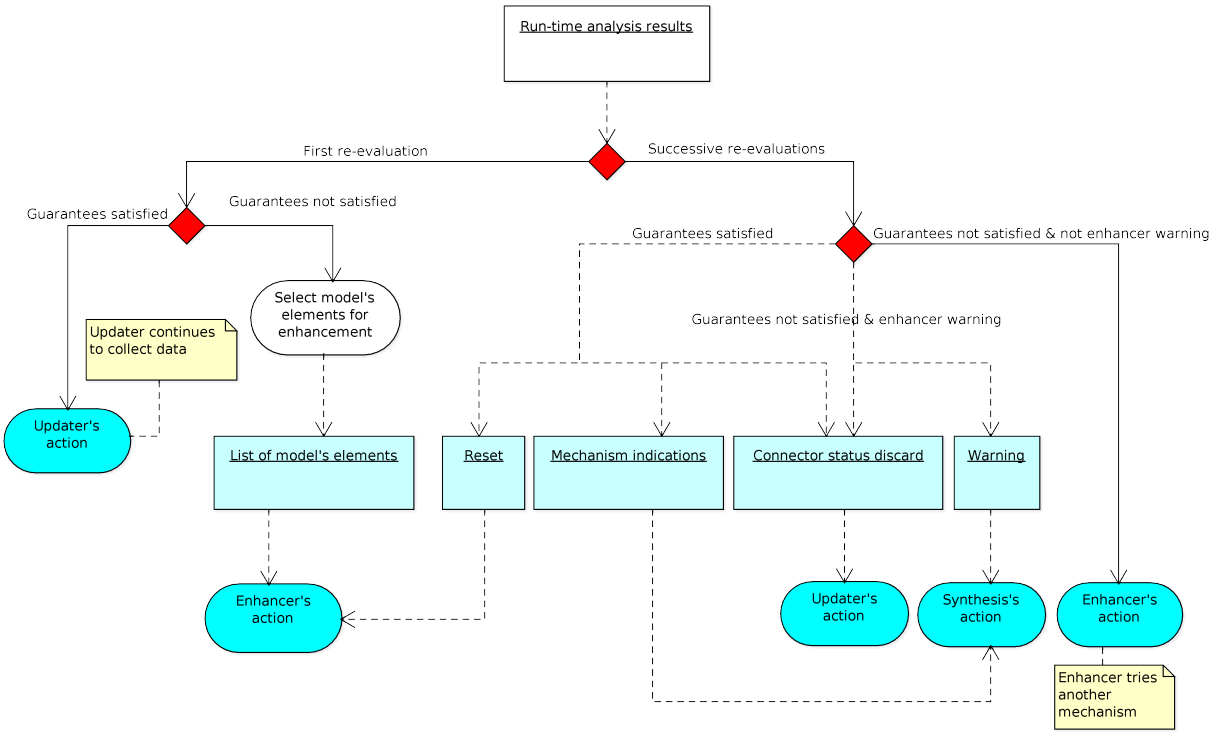


Figure 3.14: Activity Diagram of the Run-Time Branch of the Evaluator Module

ments are still satisfied or not (branch *Run-time evaluation* of the Evaluator's activity diagram in Figure 3.14). Upon detection of mismatch between the requirements and the assessed metrics, a CONNECTor enhancement cycle is attempted. At the same time, Evaluator informs Updater that the CONNECTor is no more adequate, so as to properly trigger actions towards Monitor (stop observing the CONNECTor's executions) and Deployment (stop the CONNECTor's executions). If, instead, the new analysis reveals that the requirements are still fulfilled, Updater is informed of this positive outcome and the CONNECTor continues to be used and kept under observation by Monitor.

3.4 Stochastic Model Checking Analysis Engine

The stochastic model checking engine has an internal structure very similar to that of the stochastic model-based engine described in Section 3.2. In Figure 3.17, the *Builder*, *Analyser*, *Evaluator* and *Updater* modules have the same functionality as the corresponding modules in the stochastic model-based engine. The *Builder* module produces a probabilistic model from the system specification; the *Analyser* computes quantitative assessment of the non-functional metrics; the *Evaluator* checks the quantitative assessment against the value specified in the non-functional properties and reports the results to the Synthesis enabler; and the *Updater* module receives the run-time data from the Monitor Enabler to update the model parameters. The *Repairer* module performs a similar function to the *Enhancer* module in the stochastic model-based engine: it is also used to adjust the model in case the synthesised CONNECTor does not satisfy the non-functional properties. However, we adopt a different assumptions from the *Enhancer*. Instead of iteratively attempting to satisfy the non-functional properties through the failure modes and retry mechanism, we focus on the situation where certain parameters in the CONNECTor can be adjusted. To enable usage at run-time, we aim to achieve fast performance of both analysis and repair. In the rest of this section, we give a short description of these modules, focusing on the difference between the two engines.

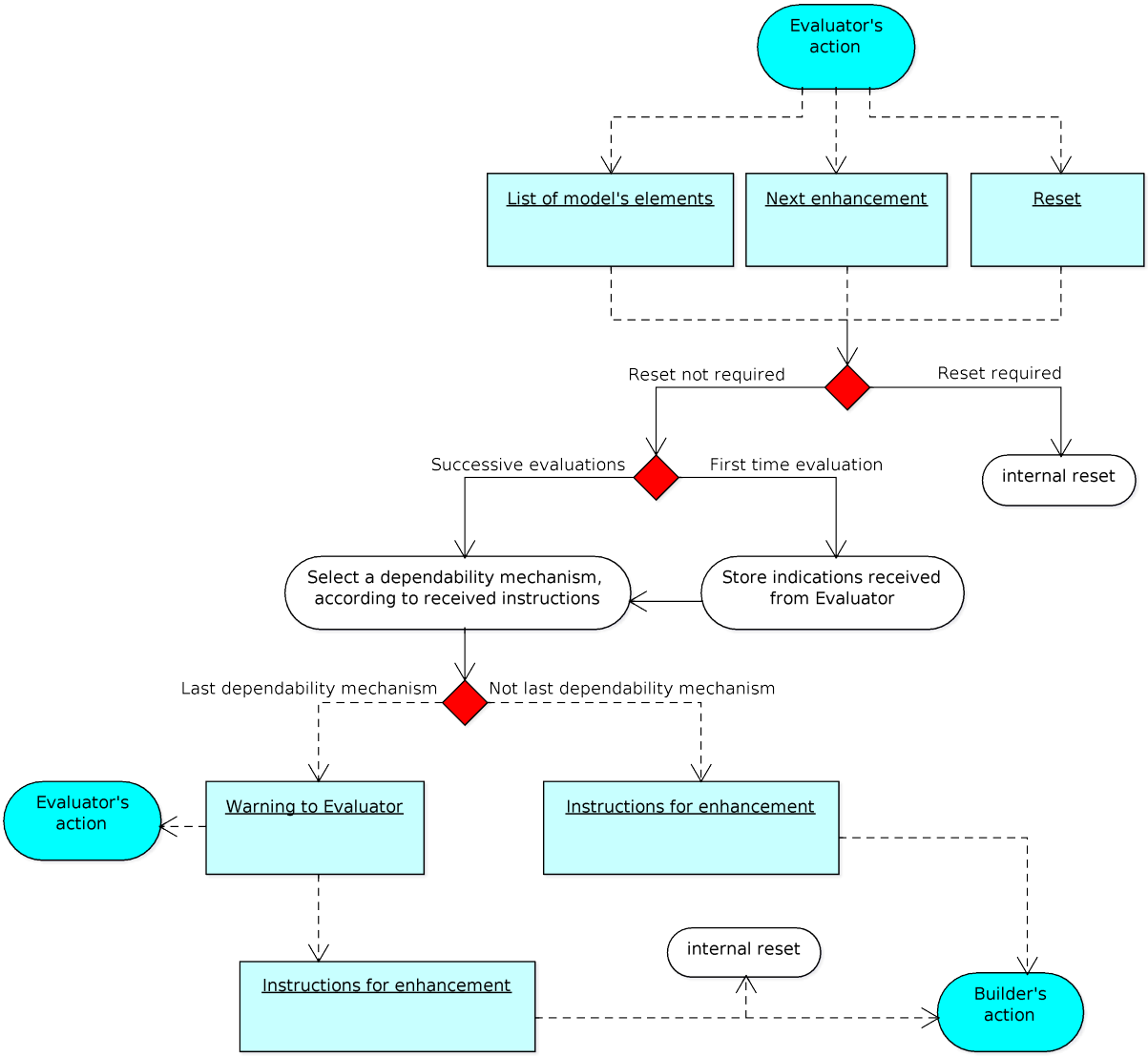


Figure 3.15: Activity Diagram of the Enhancer Module

Builder

The *Builder* module in the stochastic model-based engine translates the LTS specification of the CONNECTed system and non-functional information into a SAN model. As described in deliverable D5.2 [28], the stochastic model-checking engine is based on the well known probabilistic model checker PRISM [49]. Therefore, this module generates a PRISM model for the CONNECTed system being analysed.

Analyser

For an initial PRISM model of the CONNECTor, this module uses the offline verification technique in PRISM to compute quantitative values of the metrics specified in the required non-functional properties. Since offline verification can be slow, it may not be applicable for repeated use at run-time in CONNECT settings. Also, if only a small portion of the model has changed, such as through adaptation of the parameter values, it is not necessary to re-run the offline verification from scratch. In deliverable D5.2 we have developed an incremental verification approach which substantially improves on verification performance for models where only some probability values have changed. We thus propose to employ this incremental

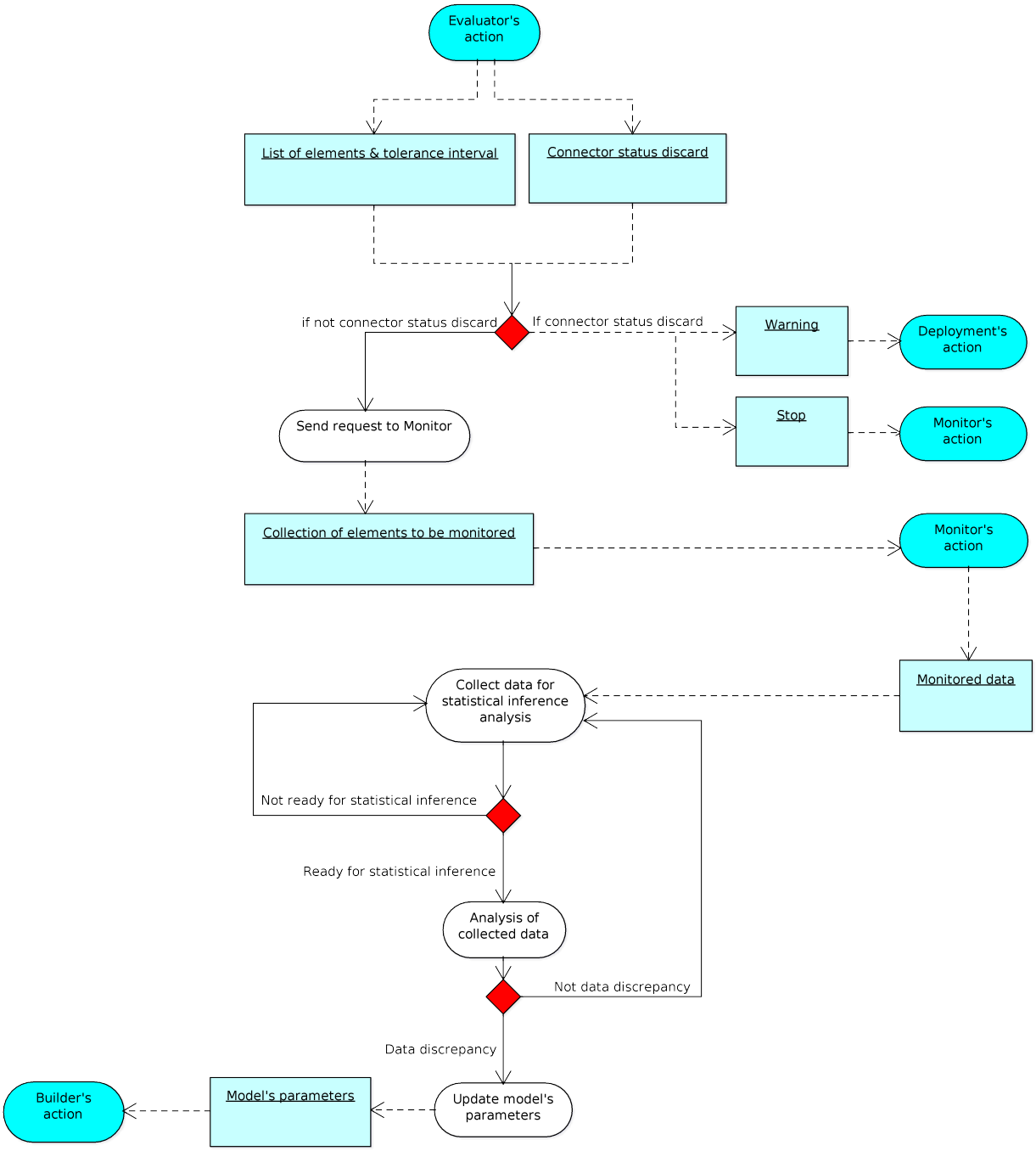


Figure 3.16: Activity Diagram of the Updater Module

verification technique for the follow-up computations, i.e., for CONNECTor models modified/evolved after requests from *Updater* and *Repairer* modules. The details will be given in Section 3.4.1.

Evaluator

This module notifies the Synthesis enabler. It compares the computed value of the non-functional metrics and those specified in the properties, and submits the comparison results to the Synthesis enabler. In case the non-functional properties are not satisfied, *Evaluator* asks *Repairer* to recommend a new CON-

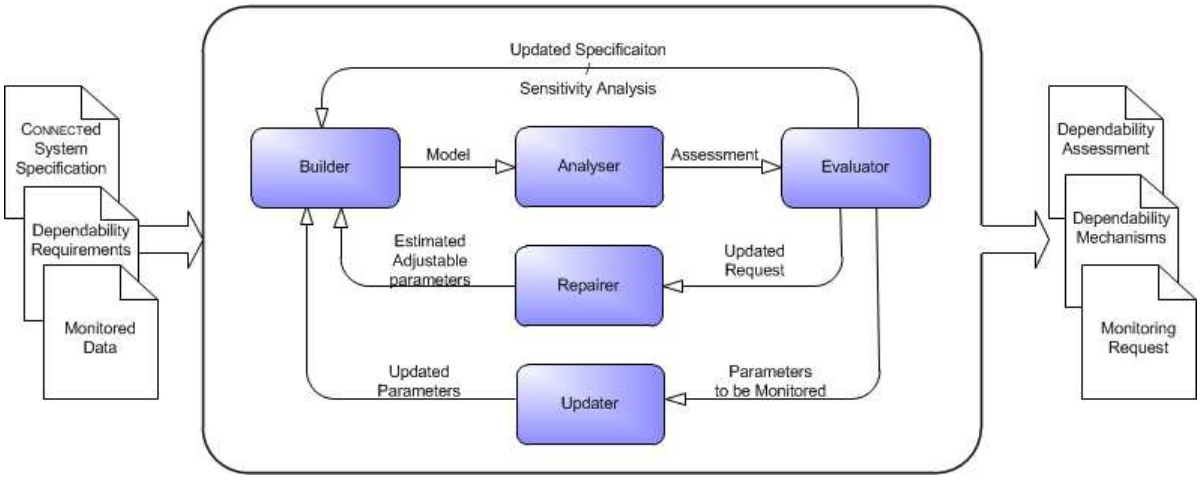


Figure 3.17: Architecture of the Stochastic Model-Checking Analysis Engine of DEPER

NECTor for synthesis. The dis-satisfaction can occur in the initial model, or due to the evolution of the model. In contrast to the method adopted in the stochastic state-based engine, here we use a different way to compute a new CONNECTor. We assume there are some parameters in the model that can be adjusted, such as the transmission rate of the channel. For example, when several communication channels are available to synthesise a CONNECTor, each of which has a different cost, which channel is used in synthesis depends not only on the cost, but also on the non-functional characteristics of the channel. We may have to choose a fast channel if the speed of the channel chosen by the synthesis enabler does not satisfy the non-functional requirement.

Updater

This module receives the data from the monitor enabler to check if some parameters take different values from the initial ones, as *Updater* in the stochastic model-based engine does.

Repairer

This module is activated by *Evaluator* when non-functional properties are not satisfied in a CONNECTed system. If there are adjustable parameters in the CONNECTor, then *Repairer* attempts to compute proper values for these parameters such that the new CONNECTor synthesised using the new values ensure that the non-functional properties satisfied. In order to effectively determine new parameter values for a new CONNECTor, we assume each adjustable parameter is defined in a bounded domain. *Repairer* adopts a variant of the Monte-Carlo sampling method to test a set of samples until it finds a *good* sample. Here a sample is a valuation of parameters and a *good* sample makes the non-functional properties satisfied in the CONNECTed system. We propose to use sampling to improve efficiency of the method, so that it can be used at run-time. This technique will be discussed in Section 3.4.2.

3.4.1 Symbolic incremental verification

In Deliverable D5.2, we have presented an effective incremental verification technique for online quantitative verification in order to provide results quickly when a CONNECTed system needs to be re-verified. In this technique, we proposed to perform offline probabilistic verification in a more efficient way: firstly, the probabilistic model, which is a DTMC, MDP, or CTMC, is partitioned into a set of strongly connected components (SCCs), which constitute a directed acyclic graph (DAG); secondly, each SCC is processed in a reversed topological order. We also gave a sufficient condition for detecting special SCCs, in each of which all states have probability zero or one. By this condition, the time-consuming precomputation is avoided. For the verification of a modified model, in which only some probability values are changed, it is not necessary to process each SCC from scratch. Instead, we analyse the DAG and identify the set

of SCCs that need to be re-processed. The experimental results shown in D5.2 demonstrated that the SCC-based incremental verification can be very fast.

A problem with the implementation, however, is that the explicit-state data structures used to store the state space and transition relation imposes a limit on the size of models that can be handled. A successful approach for alleviating this in the context of verification is to use *symbolic* implementations, based on binary decision diagrams (BDDs) [21] and extensions such as multi-terminal BDDs (MTBDDs).

A problem here, though, is that the Tarjan algorithm [69] for identifying SCCs is known to be poorly suited to symbolic implementation. Various SCC decomposition algorithms have been proposed, specifically for implementation with BDDs [19, 42]. Unfortunately, they do not explore SCCs in reverse topological order, and it is very slow to generate this order once the SCCs are stored as BDDs.

In this deliverable, we further improve the incremental verification method by adapting the Tarjan algorithm to the case where model information is stored using BDDs. This way, we are able to handle larger models than for the explicit-state implementation presented in Deliverable D5.2. We omit here low-level details of how BDDs can be used to represent and manipulate sets of states and transition relations (see e.g. [22]). Here, it suffices to know that some operations are efficient in this form and others are not. For example, some operations in the original Tarjan algorithm cannot be performed efficiently with BDDs, notably association and update of an integer index to a state. We propose a novel *hybrid* adaption of the algorithm [52] that combines symbolic and explicit-state data structures. Keeping overhead to a minimum for efficiency is non-trivial. We maintain:

- the non-probabilistic transition relation $\mathcal{E}$ and the union $allsccs$ of all visited SCCs, stored as BDDs;
- a stack $stack$ and hash table M, used during depth-first search, whose size is linear in the number of states.

Generation of the SCC partition $\overline{\overline{S}_{SCC}}$ for incremental verification with BDDs is not a simple task either. The partition algorithm is not efficiently implementable with BDDs due to the condition $\exists C \in \overline{\overline{S}_{SCC}} \,.\, Succ(C_i) \cap C \neq \emptyset$. It requires that, in each iteration of the **for** loop in the algorithm, we scan the intermediate $\overline{\overline{S}_{SCC}}$ to decide if C_i needs to be included in $\overline{\overline{S}_{SCC}}$. For explicit-state data structures, this is preferable because it saves memory with very little time cost. However, it is better to generate a (sparse) matrix T to store the relation between SCCs. An entry $T[i,j] = 1$ means that C_i depends on C_j. Thus, if C_j is included in $\overline{\overline{S}_{SCC}}$, all C_i such that $T[i,j] = 1$ are included in $\overline{\overline{S}_{SCC}}$ too. Note that T can also be encoded symbolically in order to save space. This needs the following extra variables:

- A hash table M_2 to store the root index of each SCC. In the hash table, the root index is the key, as it is unique among all SCCs, and the pointer to BDD for the SCC is the value.
- A BDD M_3 to store the root index ($vlowlink$) of each state. A hash table can be used for the same purpose, but would use more space than a BDD.

The adapted Tarjan algorithm, which we call the *hybrid Tarjan algorithm*, begins with a call to the recursive function $hybrid\_tarjan$, shown in Algorithm 2, from the initial state with $index = 1$. The lines shaded grey are used to compute T. In Algorithm 2, x and y are integers, v, v' w are BDDs, each of which represents a single state, and scc is a BDD storing the set of states in the current SCC. $M[v]$ represents the corresponding value for the hash key v. Here we utilise a feature of most BDD implementations (including CUDD, which we use): equivalent BDDs are guaranteed to have the same pointer in memory. Thus, the pointer is used as the hash key for the BDD v. $M[v] = NULL$ means that the key v cannot be found in the table and $M[v] := NULL$ denotes that the key and its value are deleted from the hash table.

In the original Tarjan algorithm, each state v is associated with two values for the index of v and the minimum index $lowlink$ among the states in the SCC containing v. To reduce memory consumption, states that have already been identified in some SCCs are stored in $allsccs$, and the hash table M_2 only stores the attribute for the current state and states in the stack. As indicated in [61], only one attribute is actually needed in an elegant implementation. Indeed, only the value $lowlink$ is stored in the hash table. For the current state v, its attributes are stored in the local variables $vindex$ and $vlowlink$; the value $vlowlink$ from its successor states is obtained from the return value of function $hybrid\_tarjan$.

Theorem 1 *The hybrid Tarjan algorithm partitions a graph $\mathcal{E}$ into SCCs correctly.*

Algorithm 2 $hybrid\_tarjan(v)$

1: $vlowlink := index$; $M[v] := index$
2: $vindex := index$; $index := index + 1$
3: $succ := \emptyset$
4: **for all** $(v, v') \in \mathcal{E}$ **do**
5: $x := 0$
6: **if** $M[v'] = NULL$ **then**
7: **if** $v' \notin allsccs$ **then**
8: $x := hybrid\_tarjan(v')$
9: **end if**
10: **else**
11: $x := M[v']$
12: **end if**
13: **if** $x > 0 \wedge vlowlink > x$ **then**
14: $M[v] := x$; $vlowlink := x$
15: **else**
16: **if** $x = 0$ **then** $succ := succ \cup \{M_3[v']\}$ **end if**
17: **end if**
18: **end for**
19: **if** $vlowlink = vindex$ **then**
20: $vlowlink := 0$; $M[v] := NULL$; $scc := \{v\}$
21: $M_3[v] := vindex$
22: **while** $stack \neq \emptyset \wedge M[TOP(stack)] \geq vindex$ **do**
23: $w := TOP(stack)$; $POP(stack)$
24: **for all** k such that $T[M[w], k] = 1$ **do**
25: $T[M[w], k] := 0$; $T[vindex, k] := 1$
26: **end for**
27: $M[w] := NULL$; $scc := scc \cup \{w\}$
28: $M_3[w] := vindex$
29: **end while**
30: $allsccs := allsccs \cup scc$
31: $M_2[vindex] := scc$
32: **else**
33: $PUSH(stack, v)$
34: **end if**
35: **for all** $k \in succ$ **do** $T[vindex, k] := 1$ **end for**
36: **return** $vlowlink$

The proof can be found in [52].

The SCCs computed by the hybrid Tarjan algorithm are stored symbolically, which makes it impossible to adapt the improved value iteration algorithm in D5.2 to compute each iteration efficiently. This is because it requires access to individual elements of the SCCs, which is inefficient for BDD-based data structures. Our approach is to generate a corresponding explicit-state data structure: a sparse matrix. This can be done relatively efficiently and is then amenable either to value iteration, or in fact solution via linear programming. We also employ an additional optimisation: we treat trivial SCCs, containing a single state without self-loops, as a special case. Probabilities for these can be computed quickly and easily using value iteration on the symbolic data structure.

It is also interesting to note that, to speed up SCC decomposition using BDDs, it is preferable to perform precomputation *before* applying the hybrid Tarjan algorithm; this is the opposite situation to the explicit-state data structure case. This is because precomputation is more efficient when using BDDs and reduces the number of states that need to be explored by the hybrid Tarjan algorithm.

3.4.2 Online model repair

In this section, we discuss the novel technique used in *Repairer* to compute a good sample (valuation) for adjustable parameters in a CONNECTor. We first give a brief summary of related work in order to examine the technique properly. Finding a good sample for parameters has been investigated in [12] for Discrete-Time Markov Chains (DTMCs), where parametric DTMC model checking technique [46] is used as the underlying technique. This technique computes a regular expression over parameters to represent the probability symbolically. For any valuation of parameters, the actual probability is obtained by simply replacing the parameters with the valuation in the expression. The problem is that, for a large model or a large number of parameters, it takes a long time to build the expression over the parameters, which is not suitable for CONNECT settings: fast response time is needed once changes in the model are detected. Furthermore, this technique cannot be applied to MDPs, which are a generalisation of DTMC, due to the non-determinism in MDPs.

In [47] and [45], two related approaches were presented for a different purpose. Given a set of parameters and their domain, which determines a multi-dimensional space, both approaches try to find all areas in the multi-dimensional space where a property is satisfied in the model. In [47], there are two approaches to achieve this purpose. The first one partitions the multi-dimensional space into a grid and generates an instantiation of the parametric model on each vertex of the grid. The value of the property in question is then computed in the instantiation. It might require refinement on the grid size. The other approach is similar to the one presented in [46]. It computes a (potentially complex) symbolic expression over parameters to evaluate the probability given that the model is a CTMC. [45] employs the technique in [46] to detect the areas in the multi-dimensional space. It recursively partitions the space into regions, each of which represents a set of concrete models that have the same truth value of the property.

In the CONNECT setting, our goal is not to find all regions in the parameter space. One good sample suffices. In this deliverable, we adopt a different strategy to find a good sample from the one in [12]. A simple strategy that partitions the sample space into a grid and compute the metric value on each vertex usually does not scale well, as the number of vertices increases exponentially as the number of parameters increases. Therefore, we use random sampling to search for a good sample. Random sampling is a common practice to overcome the complexity introduced by high-dimensional space. The Monte-Carlo method [59] is a well-known technique that uses random samples. Many variants have been developed since it was proposed in 1940s. The Latin Hypercube Sampling (LHS) method [58] is one of the variants, which in practice usually generates more evenly distributed samples in the sample space than Monte-Carlo does. In Section 3.5.2, we will compare the performance of LHS and Monte-Carlo in detail on the Terrorist Alert scenario. Since we only need to acquire *one* good sample, the model repair procedure is terminated immediately once a good sample is found. If no good sample is found, then we could restart the procedure with a different set of LHS samples. In case no good samples are found after a certain number of restarts, Repairer reports that the CONNECTor cannot be repaired, and this problem should be handled at a higher level. The experimental results in Section 3.5.2 suggest that by combining proper refinement strategies, random sampling is an effective technique for online CONNECTor repair.

3.5 Case Study

We consider the CONNECT Terrorist Alert scenario, introduced in Deliverable D6.2 [29], depicting the critical situation that during a show in the stadium, the control center spots one suspect terrorist moving around. The alarm is immediately sent to the Police. The scenario has been already introduced in Deliverable D5.2 [28]; for the sake of completeness and clarity, we briefly recall it in the following. This description extends the one provided in Section 2.4, since more aspects of the scenario are involved in the dependability and performance analysis illustrated in the following.

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, for example a picture of a suspect terrorist, with the Police control center and with other policemen through a *Secured-FileSharing* application.

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

Within such an emergency situation, 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.

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.

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.

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.
- 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 two applications, *SecuredFileSharing* and *EmergencyCall* in this scenario represent the two Networked Systems, which are not a priori compatible. Hence, to allow a Policeman and the guards in the zone where the suspect has escaped to communicate we need to synthesise on-the-fly a CONNECTor. The needed mappings are shown in Figure 3.18 and briefly summarised below.

CONNECTor

- 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.

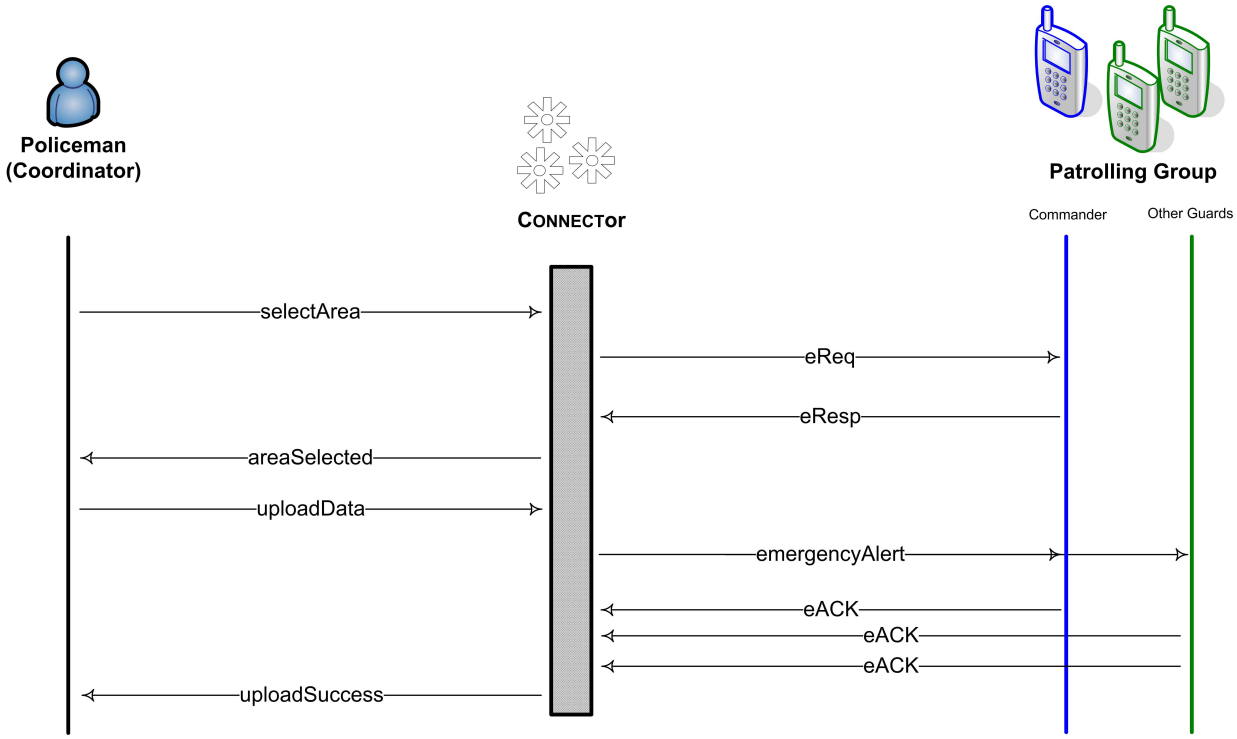


Figure 3.18: Terrorist Alert Scenario: Sequence Diagram of the Messages Exchange

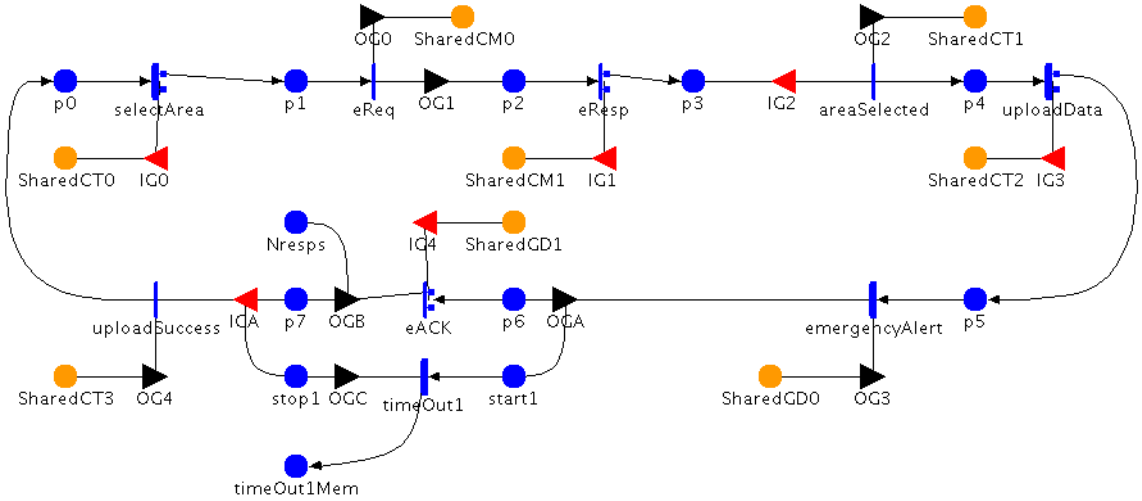


Figure 3.19: Terrorist Alert Scenario: SAN Model of the CONNECTor

```
1 double coverage() {
2   return ( (double) connector->Nresps->Mark() ) / ( guardNum + commNum );
3 }
```

Listing 3.1: Reward function of Coverage

3.5.1 Scenario analysis through the stochastic model-based approach

Taking as a reference the above described scenario, we performed two types of analyses for the CONNECTed system, using the Möbius [38] based implementation of the stochastic model-based analysis engine of DEPER. The former is carried out at pre-deployment time, in order to show the capability of the Enhancer module, described in Section 3.2, to produce a new enhanced model of the CONNECTed system. The latter is performed at run-time, after the deployment of the CONNECTor, in order to refine the accuracy of model parameters through on-line observations.

Figure 3.19 depicts the dependability and performance model of the synthesised CONNECTor built by DEPER at design time, using the SAN formalism [67]. We recall that this model is obtained through automatic transformation from the LTS specification of the networked system, that is the *SecuredFileSharing* and *EmergencyCall* in the considered scenario. The measures assessed in the evaluation are average estimates of *latency* and *coverage*.

Latency represents a performance indicator and is measured from when the control centre sends the initial request `selectArea` to when it receives `uploadSuccess`.

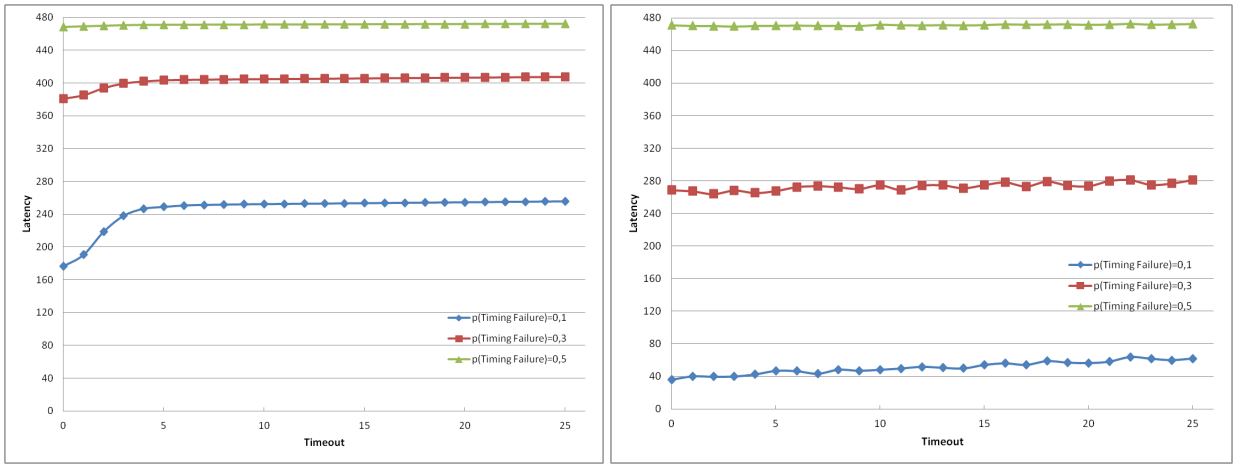
Coverage represents a dependability indicator and is given by the percentage of responses the control centre receives back within a certain time T. We have already provided the CPMM specification of a coverage property for the Terrorist Alert scenario (see Section 2.4). The corresponding reward function used in Möbius is shown in Listing 3.1, where the 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).

Pre-deployment analysis

The analysis we describe can be automated with the approach reported in [55]. In order to simplify the exposition, here we consider only failures between the CONNECTor and the guards' devices (which execute the *EmergencyCall* protocol), and we use the probing mechanism to contrast timing failures, and the majority voting mechanism to contrast value failure. Both mechanisms are introduced on the communication channel between the CONNECTor and the guards' devices (which follow the Emergency Call protocol).

The first analysis aims at assessing the trend of latency for different values of timeout, assuming three different values of timing failure probability between the CONNECTor and the guards. Figure 3.20(a), shows the value of latency (on the y axis) for the CONNECTed system without dependability mechanisms (the timeout value is reported on the x axis). Figure 3.20(b), shows the same analysis performed on the model enhanced with the probing mechanism. We can notice that, with the considered system parameters, the mechanism is able to reduce latency only in two out of three situations. In particular, when the timing failure probability is 0.3 and 0.1 the value of latency is markedly reduced with respect to the basic model. When the timing failure probability is 0.5 the latency has very similar values for both models; in fact, in such a case the failure probability has a too high value and the probing mechanism is a too light means to contrast its effects.

The second analysis is performed for three different probabilities of failure values between the CONNECTor and the guards. Figures 3.21(a) and 3.21(b) show the analysis results for the basic model and for the model enhanced with the majority voting mechanism. In this case, the mechanism is able to improve coverage in all considered cases. We can notice that, for the considered probability values, the coverage provided by the enhanced model is approximately double compared to the basic one.



(a) Basic model

(b) Enhanced Model with Probing Mechanism

Figure 3.20: Latency Assessment in Case of Riming Failure

Analysis based on on-line collected data

On-line data obtained through the Monitor Enabler are exploited by DEPER to refine the accuracy of (critical) model parameters values adopted in the pre-deployment analysis. As already described in the previous sections, DEPER and GLIMPSE interact each other to exchange requests for monitoring and to gather monitored data.

The measures assessed in this evaluation are again *latency* and *coverage*, based on the actual values of model parameters that have a hold on the measures chosen. A sensitivity analysis on the impact of model parameters on the assessment of these selected measures revealed that critical parameters to keep under observation on-line via the Monitor Enabler are: i) the time between `eReq` and `eResp` events, for the latency measure, and ii) the occurrences of `emergencyAlert` events for the coverage measure. Refining the pre-deployment analysis knowledge on the values assumed for such parameters by real observations constitutes a fundamental step in enhancing the accuracy of the analysis results. In fact, should the initial forecast for these parameters deviate from what is evidenced through repeated executions, a new analysis round needs to be triggered to understand whether the dependability and performance requirements are still met by the CONNECTed system.

An example of request message sent by DEPER to GLIMPSE, in order to trigger the monitoring of the critical transition for latency aspects, is shown in the Listing 3.2.

The GLIMPSE infrastructure, more specifically its Manager component, receives the DEPER requests and sets up the ComplexEventProcessor with the provided rule.

According to the scenario, the CONNECTor sends an `eReq` message to the commanders of the patrolling groups operating in a given area of interest.

The event generated by the Probe instrumented into the peer software component is shown in Figure 3.22 and flows in into the GLIMPSE infrastructure stream of events.

When the commanders reply, another event is fired and sent on the CONNECT bus, the `eResp` event.

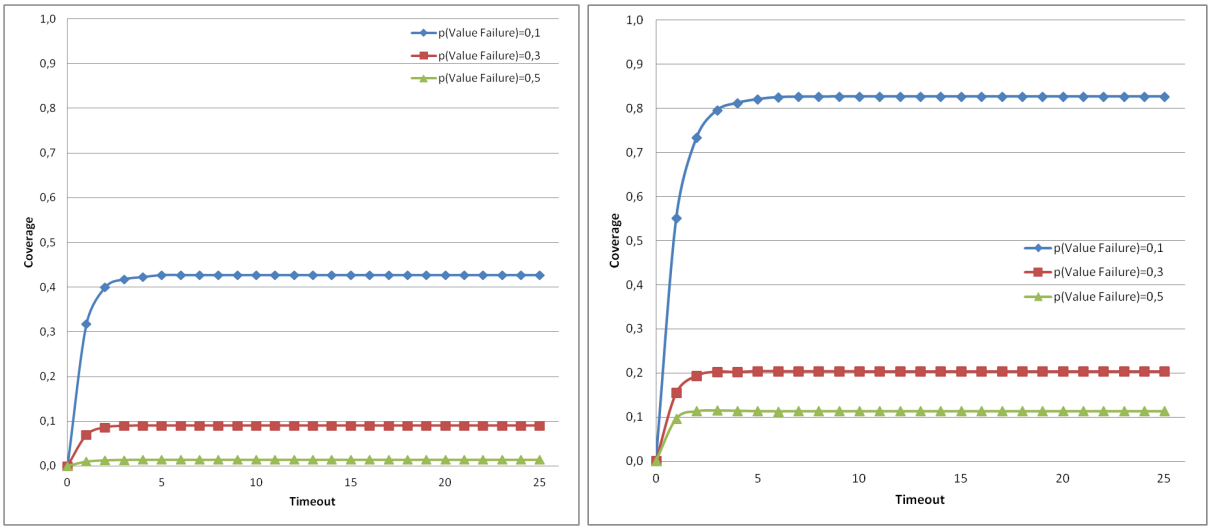
The rule `computation time` (lines (20-28) in Listing 3.2) uses the timestamp impressed into the two different events to infer latency, and matches the parameters: `connectorID`, `sequenceID`, `ConnectorInstanceID`, and `ConnectorInstanceExecutionID` to check that the events are generated from the same CONNECTor. This rule allows to calculate the duration between `eReq` and `eResp` events (line 35) and to provide it to DEPER (line 40-41).

Indeed, the rule `pending request` in the Listing 3.2, (lines (48-54)), computes the number of incoming requests into the CONNECTor and provides it to DEPER.

We first consider the steps to refine the accuracy of the *failure probability* of the communication channel between the *EmergencyCall* application and the CONNECTor. In order to get statistically sig-

```
<?xml version="1.0" encoding="UTF-8"?>
<ComplexEventRuleActionList>
  <Insert RuleType="drools">
  <RuleName>Computation Time</RuleName>
  <RuleBody>
  declare ConnectBaseEventImpl
      @role(event)
      @timestamp(timestamp)
  end
  declare SatisfiedRequest
      duration : float
      incoming : SimpleEvent
      outcoming: SimpleEvent
  end
  rule "computation time"
  no-loop
  salience 999
  dialect "java"
  when
    $aEvent:ConnectBaseEventImpl(this.data=="eReq",
          this.getConsumed == false);
    $bEvent:ConnectBaseEventImpl(this.data=="eResp",
      this.getConsumed == false,
      this.getConnectorID == $aEvent.getConnectorID,
      this.getConnectorInstanceID == $aEvent.getConnectorInstanceID,
      this.getConnectorInstanceExecutionID ==
      $aEvent.getConnectorInstanceExecutionID,
      this after $aEvent);
  then
    $aEvent.setConsumed(true);
    $bEvent.setConsumed(true);
    SatisfiedRequest sr = new SatisfiedRequest();
    sr.setIncoming($aEvent);
    sr.setOutcoming($bEvent);
    sr.setDuration(DroolsUtils.latency($aEvent.getTimestamp(),
                $bEvent.getTimestamp()));
    insert(sr);
    retract($aEvent);
    retract($bEvent);
    ResponseDispatcher.NotifyMe(drools.getRule().getName(),
            "DePer module", sr.getDuration());
  end
  rule "pending request"
  no-loop
  salience 999
  dialect "java"
  when
    $total : Number()
    from accumulate($nEvent : ConnectBaseEventImpl(data=="eReq")
          from entry-point "DEFAULT",
          count($nEvent))
  then
    ResponseDispatcher.NotifyMe(drools.getRule().getName(),
          "DePer Module", "PENDING: " + $total);
  end
    </RuleBody>
 </Insert>
</ComplexEventRuleActionList>
```

Listing 3.2: Sample Request from DEPER Enabler to Monitor



(a) Basic model

(b) Enhanced Model with Majority Voting Mechanism

Figure 3.21: Coverage Assessment in Case of Value Failure

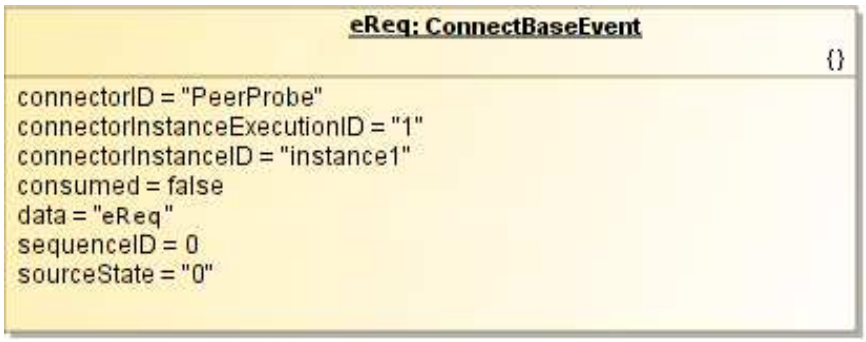


Figure 3.22: The eReq Event Sent by the PeerProbe

nificant estimations from the analysis of the data gathered from the Monitor, we fixed the confidence level to 95%, the confidence interval to 0.1, and the variance to 0.01. We accumulated data generated from several executions of the CONNECTor in scenario's configurations where the number of guards was varying. In each configuration, executions have been performed until the mean value of `emergencyAlert` message occurrences notified by Monitor stabilizes within the assumed confidence interval. From such mean value, the mean failure probability we are interested in is obtained as $1 - (number\ of\ guards/number\ of\ emergencyAlert)$. Then, applying the iterative algorithm presented in Section 3.2 to the mean failure probability for each scenario's configuration, the overall mean failure probability is obtained.

Figure 3.23 shows the trend of the coverage (on the y axis) for different values of the failure probability (on the x axis). Also, the threshold coverage line as specified in the requirement (set to the value 0.8) is reported. Not surprisingly, as the failure probability increases, coverage decreases. The value of failure probability assumed during pre-deployment dependability analysis was 0.05, and the coverage value obtained through the pre-deployment analysis is 0.9, fully satisfying the requirements, as shown in Figure 3.23.

Table 3.1 summarizes the data involved in this experiment to obtain (as the average of the values reported in the last column) the refined value of failure probability of 0.1416 for the parameter under observation, a clearly divergent value calling for a new evaluation of the coverage measure.

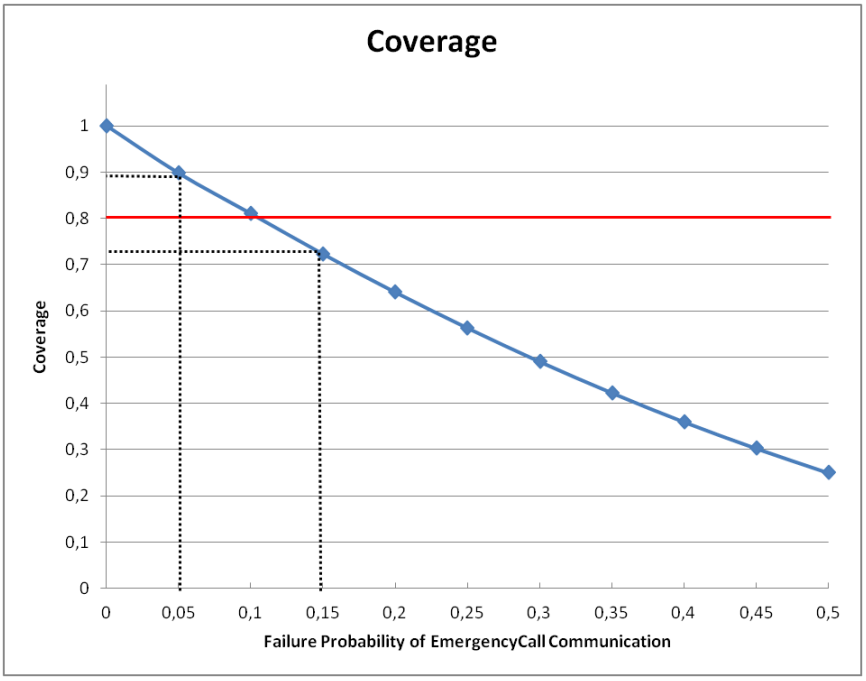


Figure 3.23: Trend of Coverage as a Function of Failure Probability of the EmergencyCall Channel

Number of Guards	Occurrences of ‘‘emergencyAlert’’	Failure probability
22	18.94	0.139
33	28.84	0.126
44	37.96	0.137
55	46.54	0.154
110	93.27	0.152

Table 3.1: Elaboration of Data from Monitor to Update the Failure Probability Parameter

The coverage value after updating the failure probability parameter is 0.73 (Figure 3.23), not a satisfactory value anymore. The CONNECTor needs to be improved; therefore DEPER informs the Synthesis Enabler about the analysis results and appropriate actions are taken by Synthesis (typically, a new CONNECTor is synthesised).

Now, we move to the steps to refine the accuracy of the model parameters critical for the assessment of the *latency* indicator. They are the execution time of the model transitions `eReq` and `eResp` in Figure 3.19. These transition execution time are represented by an exponential distribution, with rate 1. Similarly to the previous case of coverage, executions have been performed and the time durations of the transitions under observations collected from Monitor. Table 3.2 summarizes the mean values for the time duration of the two transitions (in time units). Through the *probability plotting paper* method [10], it is then possible to estimate the actual value of the distribution rate, that results to be 0.89.

Transition	Timing duration
eReq	9.650855
eResp	10.647591

Table 3.2: Timing Values from Monitor

Figure 3.24 shows the trend of *latency* (on the *y* axis) at increasing values of *Timeout* (on the *x* axis). The latency threshold specified in the requirements (30 time units) is also depicted. The figure includes three plots, corresponding to: (i) the results of the pre-deployment analysis; (ii) the results of the analysis after the parameters influencing latency have been updated; and (iii) the results of the analysis after both the parameters influencing latency and coverage have been updated. It is worth noting that

latency exceeds the required threshold only when all model parameters under on-line observation have been updated, for values of Timeout bigger than 21 time units.

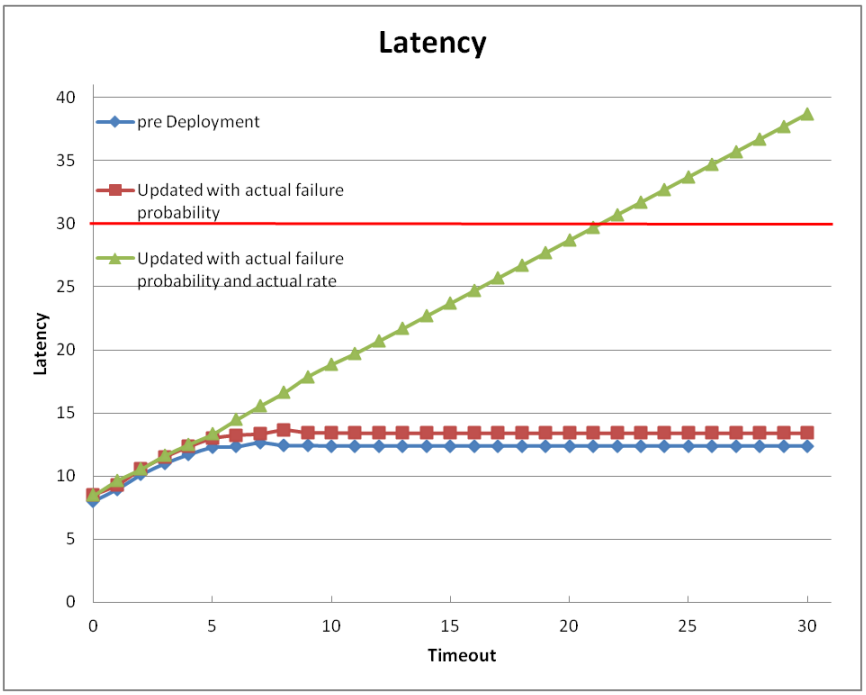


Figure 3.24: Trend of Latency as a Function of Timeout

Similarly, Figure 3.25 shows the trend of *coverage* (on the *y* axis) at increasing values of *Timeout* (on the *x* axis). As in the previous Figure 3.24, we show the pre-deployment analysis results, those of the analysis performed after updating the value of the failure probability, and those relative to the analysis where both coverage and latency related parameters have been updated at run-time. It can be noted that pre-deployment estimation of coverage was too optimistic: if the coverage requirement is set higher than 0.73, the synthesised CONNECTor fails to meet it, whichever be the assumed value for the Timeout.

3.5.2 Scenario analysis through the stochastic model checking approach

In this section, we examine the performance of the online model repair technique presented in Section 3.4.2 on the Terrorist Alert scenario. As specified in D5.2, we assign rate $R1$ to all transitions between the control center and the CONNECTor, and assign $R2$ to those between the CONNECTor and the guards. $R1$ and $R2$ represent the communication speed, which may be adjusted as the CONNECTed system evolves. We consider the coverage property in the following experiments. In this model, the percentage of coverage increases as $R1$ and/or $R2$ increase, which means the coverage is monotonic to $R1$ and $R2$. In order to generate interesting results, we feed the repairer a fixed value for the coverage and ask it to find a good sample of $(R1, R2)$ which makes the model match the fixed coverage value within an error bound ϵ.

In the experiments, we fixed both the number N of commanders and the numbers M of other guards to be two. Since we only change the rate of transitions, the number of states, which is 1774, does not change as $R1$ and $R2$ vary. We also assume the range of $R1$ and $R2$ to be $(0, 0.5]$. The experiment was performed on an AMD Phenom(tm) 9600B Quad-Core Processor with 8GB memory running Fedora 12 x86_64 Linux. Four threads were generated to evaluate samples in parallel. LHS and Monte-Carlo samples are generated by DAKOTA [6].

The first experiment is designed to compare LHS and Monte-Carlo on different configurations with respect to the coverage value, number of samples and the error bound. The results are given in Table 3.3. In addition to the running time, we report in column “Test” the number of samples that has been tested, as

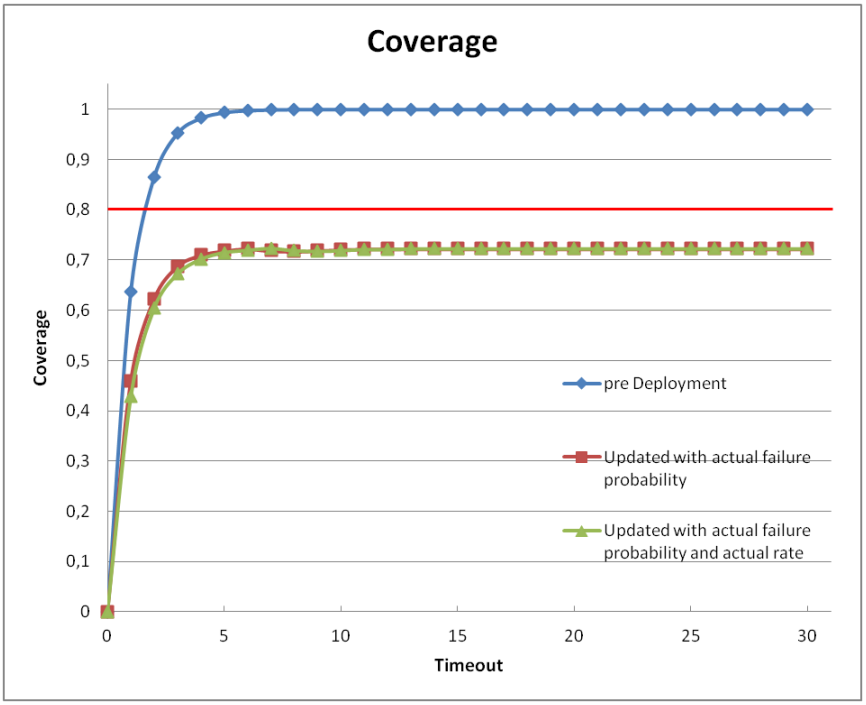


Figure 3.25: Trend of Coverage as a Function of Timeout

the model repair procedure is terminated immediately when a good sample is found. In Table 3.3, we also demonstrate the performance of a grid-based sampling approach. For this example, the sample space is partitioned into a $n \times n$ grid, where $n = \lceil m^{1/2} \rceil$ and m is the number of samples in LHS and Monte-Carlo approaches. It is unexpected that the grid-based sampling approach is very slow, even when only a few samples are tested. The main reason is that it took longer for the model checking algorithm to terminate on the instantiated models which are stiff, i.e., the difference between $R1$ and $R2$ is large, than on others. Since the grid-based sampling approach is very slow, we do not show it in the following experiments.

Table 3.3: Online Model Repair Results

Coverage	Num of Samples	Error Bound ϵ	LHS		Monte-Carlo		Grid	
			Time (s)	Tested	Time (s)	Tested	Time (s)	Tested
0.8	100	0.1	0.675	4	1.616	21	192.691	8
		0.01	4.321	56	2.542	50	256.432	14
		0.001	(4.476)	100	(4.618)	100	(453.368)	100
	1000	0.1	0.576	4	0.675	4	382.301	201
		0.01	1.988	48	2.077	48	602.617	43
		0.001	22.843	379	28.05	801	(1350.191)	1000
0.9	100	0.1	0.619	4	0.612	4	266.187	14
		0.01	0.825	11	1.332	20	(452.998)	100
		0.001	1.058	20	1.253	28	(448.871)	100
	1000	0.1	0.595	4	0.596	4	594.523	44
		0.01	1.387	15	0.95	15	592.158	47
		0.001	6.045	145	7.385	160	(1354.835)	1000

In Table 3.3, the time in parentheses (in the third line) means that no good sample was found among the 100 generated samples. In other cases, a good sample was found fairly quickly, except for the case where the coverage is 0.8, the number of samples is 1000 and the error bound is 0.001. Hence, random

sampling is an effective approach to search for a good sample in high dimensional sample space. Note that in most cases, the performance of LHS is slightly better than that of Monte-Carlo due to the evenly distributed samples. Since LHS is still a random sampling method, it could happen that no good sample can be generated by LHS, and hence further sampling or refinement is necessary.

Table 3.4 shows the experimental results when no good samples are found in the first round of sampling. In the second experiment, we fix the coverage value to be 0.8, the number of samples to be 100 and the error bound to be 0.001. This is the case in the first experiment where LHS and Monte-Carlo fail to generate a good sample. In this experiment, we generate 100 samples by LHS and Monte-Carlo up to five refinement iterations. Since both sampling methods generate random samples, a single experiment can lead to random results. Thus we repeat the experiment 12 times in order to have a better view of the performance.

Table 3.4: Experimental Results for $Coverage = 0.8, Num\ of\ samples = 100, \epsilon = 0.001$

Experiment No.	LHS		Monte-Carlo	
	Time (s)	Tested	Time (s)	Tested
1	17.478	398	(20.697)	500
2	(36.132)	500	(21.73)	500
3	(22.452)	500	27.007	273
4	(22.062)	500	16.757	348
5	(21.794)	500	(25.989)	500
6	(22.499)	500	(21.47)	500
7	(27.995)	500	11.359	228
8	(19.675)	500	(26.787)	500
9	(33.524)	500	15.684	296
10	12.865	288	(24.507)	500
11	9.496	211	18.644	359
12	17.491	374	(24.206)	500

In Table 3.4, the time in parentheses has the same meaning as before: no good sample is found after five iterations of sampling. In this experiment, LHS does not show any advantages over Monte-Carlo. On the contrary, Monte-Carlo succeeds more times than LHS does. This experiment suggests that simply repeating the sampling process does not guarantee that a good sample can be found. For LHS, the main reason is that the total 500 LHS samples in the five iterations are less evenly distributed than 500 LHS samples generated in one iteration. This leads to the third experiment where we examine in detail the performance of different number of samples in one iteration. The experimental results are listed in Table 3.5, where the coverage and the error bound are still fixed to 0.8 and 0.001 respectively.

From Table 3.5 we can conclude the performance of LHS is more predictable than Monte-Carlo, as a large set of LHS samples is distributed more evenly than a small set of LHS samples. Note that the long execution time using Monte-Carlo for 300, 400 and 500 samples was caused by the long computation time on the stiff models.

In the next experiment, we revise the strategy for refinement. We begin with 100 samples. In every following iteration, the number of samples is doubled from the previous iteration, which means that we test 200 samples in the second iteration, 400 in the third one, 800 in the fourth one and 1600 in the last refinement iteration. We name this strategy *binary exponential resampling*. As in the second experiment, we repeat this experiment 12 times. The results are presented in Table 3.6. In most of cases, this strategy works well by using LHS. The time variance between the longest and shortest time for Monte-Carlo is much larger than that of LHS, which coincides with the conclusion from the previous experiment. It also happened once that we did not obtain any good examples after the last refinement iteration by using Monte-Carlo.

In the last experiment, we show a different refinement strategy, which is called *closest resampling*. Let b the given coverage value in the property and c the coverage value computed on the instantiated model by a sample. We define *distance* d to be the absolute value of the difference between b and c, i.e.,

Table 3.5: Experimental Results for $Coverage = 0.8, \epsilon = 0.001$

Num of Samples	LHS		Monte-Carlo	
	Time (s)	Tested	Time (s)	Tested
100	(4.685)	100	(7.646)	100
200	5.973	148	8.945	171
300	(11.514)	300	65.877	225
400	2.137	38	(74.349)	400
500	(22.049)	500	(66.719)	500
600	12.486	278	14.199	278
700	13.21	298	4.545	39
800	17.081	477	19.286	381
900	29.707	785	41.627	427
1000	28.614	585	7.362	162

Table 3.6: Experimental Results for $Coverage = 0.8, Num\ of\ samples = 100, \epsilon = 0.001$ **with binary exponential resampling**

Experiment No.	LHS		Monte-Carlo	
	Time (s)	Tested	Time (s)	Tested
1	6.683	155	38.991	998
2	16.075	170	18.528	349
3	20.833	152	9.991	102
4	7.785	141	(131.678)	3100
5	66.081	981	27.264	654
6	72.969	204	10.537	174
7	9.542	238	10.574	155
8	18.125	228	63.386	788
9	12.414	325	17.507	384
10	7.18	166	80.068	1581
11	22.923	586	36.312	400
12	17.58	366	8.471	121

$d = |b - c|$. Let S be a sample, and S_{R1} and S_{R2} the value of $R1$ and $R2$ in sample S respectively. Let l_{R1} (l_{R2} resp.) and u_{R1} (u_{R2} resp.) be the lower and upper bound of $R1$ ($R2$ resp.). The strategy works as follows. If no good sample is found in n samples, then we chose the sample S that has the shortest distance and regenerate n samples in the new sampling space $(l'_{R1} \leq R1 \leq u'_{R1}) \wedge (l'_{R2} \leq R2 \leq u'_{R2})$, where $l'_{R1} = S_{R1} - \delta_{R1}$, $u'_{R1} = S_{R1} + \delta_{R1}$, $l'_{R2} = S_{R2} - \delta_{R2}$, and $u'_{R2} = S_{R2} + \delta_{R2}$ with $\delta_{R1} = (u_{R1} - l_{R1})/20$ and $\delta_{R2} = (u_{R2} - l_{R2})/20$. Note that if $S_{R1} - \delta_{R1} < l_{R1}$, then $l'_{R1} = l_{R1}$; if $S_{R1} + \delta_{R1} > u_{R1}$, then $u'_{R1} = u_{R1}$. The new bounds l'_{R2} and u'_{R2} are processed in the way. If we still cannot obtain a good sample in the reduced space, the refinement is repeated with $l_{R1} = l'_{R1}$, $u_{R1} = u'_{R1}$, $l_{R2} = l'_{R2}$, $u_{R2} = u'_{R2}$. In this experiment, we allow four refinement iterations. Table 3.7 shows the experimental results for smaller error bounds from 0.0001 to 0.001.

From the last experiment, we also observe that LHS finds a good sample faster than Monte-Carlo, although LHS does not always test fewer samples than Monte-Carlo does. Comparing Table 3.6 and Table 3.7, we can conclude that *closest resampling* is more effective than *binary exponential resampling* for this scenario. We need more case studies to check whether this conclusion is true in general.

Table 3.7: Experimental Results for $Coverage = 0.8, Num\ of\ samples = 100$ with closest resampling

Error	LHS		Monte-Carlo	
Bound ϵ	Time (s)	Tested	Time (s)	Tested
0.0001	8.268	251	12.599	208
0.0002	7.551	223	11.995	226
0.0003	5.318	115	12.833	208
0.0004	5.827	139	11.383	210
0.0005	6.21	155	11.301	208
0.0006	5.953	147	10.001	164
0.0007	6.694	184	8.374	112
0.0008	5.836	143	8.191	109
0.0009	4.907	101	9.018	133
0.001	5.698	137	9.12	139

3.6 Conclusions and Future Work

During this third year, the work on dependability and performance analysis concentrated on the extension and refinement 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. In particular, the activity mainly concentrated on investigating the adaptation features of DEPER: i) on one side, the refined definition and implementation of the Enhancer module, which performs a form of pre-deployment adaptation to try to overcome deficiencies of the CONNECTor specification as revealed by the analysis through pre-defined dependability mechanisms; ii) on the other side, the refined definition and implementation of the Updater module, which interacts with the Monitor Enabler to refine the accuracy of model parameters through on-line observations, thus providing adaptation of the off-line analysis performed at pre-deployment time to cope with changes in, or inaccurate estimates of, model parameters.

We also extended our work in D5.2 on incremental verification to cope with large systems, which need to be modelled in a symbolic way. Further, we started to investigate the online model repair technique using sampling methods. Both techniques are aimed to achieve fast performance for usage at run-time. Since random sampling methods do not guarantee that a good one can be produced within a certain number of samples, refinement strategies are necessary for our purpose. We have demonstrated two refinement strategies *binary exponential resampling* and *closest resampling*. The former increases the number of LHS samples as the number of refinement iterations increases, while the latter confines the sampling space around the closest sample. The *closest resampling* strategy works well for the Terrorist Alert scenario. However, better refinement strategies are still desirable. How to utilise the information obtained from the generated samples to guide the refinement is an important direction to study in the final year of the project. It is also important to search for good heuristics to choose a proper initial number of samples.

During the last year, we plan to further refine and integrate the DEPER enabler within the broader CONNECT architecture. In particular, we are already working at consolidating the loop with the Synthesis Enabler, so that Synthesis can profitably embed in the synthesis of the CONNECTor the indications coming from the analysis. In fact, the analysis of basic fault-tolerance mechanisms/patterns, among an available set, to react to selected failure modes experienced by the CONNECTed system is a very important feedback to the Synthesis Enabler towards the synthesis of a dependable CONNECTor.

Another direction of activity will be the set up and running of a demonstrator that exploits the DEPER analysis both at pre-deployment stage, with the loop with Synthesis to enhance the CONNECTor specification, and on-line through the interaction with Monitor to adapt the pre-deployment analysis to cope with changes in, or inaccurate estimates of, model parameters.

4 Security in CONNECT

By enabling the interoperability between heterogeneous Networked Systems, the CONNECT architecture raises several interesting security problems, at different levels. In the deliverable D5.2 [28], we have first presented a detailed analysis of the threat model concerning the different actors involved in CONNECT, namely:

- the **Networked systems**, which are systems manifesting the will to connect to other systems for fulfilling some intent identified by their users and the applications executing upon them.
- the **Enablers**, which are networked entities in the environment of networked systems incorporating all the intelligence and logic offered by CONNECT for enabling connection between heterogeneous networked systems. The Enablers constitute the CONNECT enabling architecture.
- the **CONNECTors**, which are the emergent product of the action of the enablers in order to satisfy the will of the Networked Systems to interact. More formally, a CONNECTor is defined in Deliverable D3.3 [31] as a piece of software that only allows feasible interactions between sequences of required actions and sequences of provided actions.

In this context, we first assume that the Enablers, which are defined by trusted entities, are always trustworthy, and thus are not concerned with security problems. Hence, we have identified three main security issues:

- Issues concerning the communication between the Networked Systems and the CONNECTor. A typical example is to ensure that the communication are encrypted, since the communication channels cannot be always trusted. This kind of issues can be addressed at the synthesis level, by including within the synthesis process the corresponding requirements. These issues are therefore addressed in WP3, see e.g., D3.3 [31].
- Issues concerning the behaviour of the CONNECTor. Although the CONNECTor can be trusted, its interaction with Networked Systems, which are not always trusted, could lead to some potential attacks. A typical example is the Anonymizer described in Section 3.2, which anonymize the connection between the CONNECTor and the Networked Systems, in order to avoid denial-of-services attacks. These kind of issues can be solved by defining security properties, as stated by the CONNECT Property Meta-Model (CPMM), and therefore are addressed in Chapter 2.
- Issues concerning the protection of the data of the Networked Systems, and of the Networked Systems in general. Indeed, each Networked Systems has its own security policy, that is the correct way to interact with it, and the CONNECT architecture must ensure that it will not compromise the Networked Systems. Although some parts of the policy can be directly included in the protocol of the Networked Systems (*e.g.*, a given part of the interface should not be accessible), in general, a security policy needs to be monitored at run-time. We focus on this specific problem in this chapter.

In this chapter, we focus on the definition of the Security Enabler with a particular eye to how it works for guaranteeing security at run-time through the usage of the Security-by-Contract-with-Trust ($S \times C \times T$ for short) framework described in the previous years of the project and briefly recalled here (Section 4.2). In particular, since trust aspects are presented in Chapter 5, we present here the part related to the monitoring and enforcement of the security policy.

4.1 Security Enabler Overview

According to Deliverable D1.3 [30], the CONNECTor architecture, described in Figure 4.1, works as follows: a NS1, coming with a Policy 1, sends a message to the CONNECTor, which mediates it in order to allow the communication with NS2, coming a Policy 2. The roles of the policies of each NS is to describe which messages can each NS accept, or conversely, which message would be non secure. For instance, a NS could require that any message should be cryptographically signed by a trusted entity, or that the number of simultaneous connections should be below some threshold.

In this context, the Security Enabler receives violation events directly from probes inserted into the CONNECTor informing the decision to perform the adaptation loop. Indeed, the role of the Security Enabler is to ensure a secure communication among Networked Systems (NSs). In CONNECT, this means that the Security Enabler contributes to:

- obtain a secure CONNECTor at design phase, *i.e.* a CONNECTor able to satisfies security request (hereafter called security policies) of NS involved in the communication (WP3);
- assuring that the communication is secure at run-time, *i.e.* that a CONNECTor satisfies the security policies for the Networked Systems (NSs) it is connecting (WP5). This is made by using the Security-by-Contract-with-Trust framework described in the previous years of the project.

Hence, in this deliverable we describe how the Security Enabler works on the CONNECTor at run-time and the implementation of the described S×C×T functionalities.

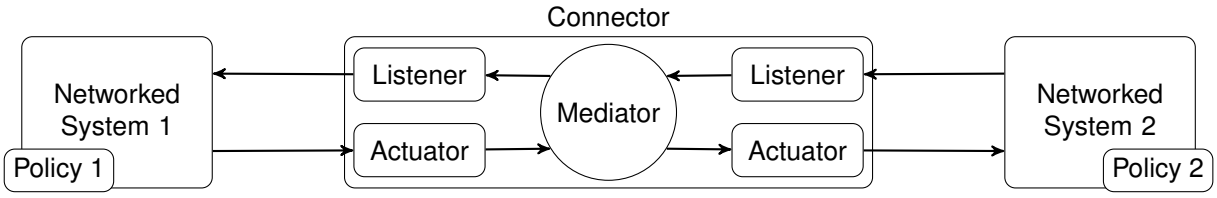


Figure 4.1: CONNECT architecture

4.2 Security-by-Contract-with-Trust

The Security-By-Contract-With-Trust framework has been introduced and deeply explained in the previous two years of the project. During the third year of the project, we worked on the implementation of the SxCxT functionalities according to the CONNECT architecture. Hence, here we just briefly recall the key point of the framework.

The S×C×T Workflow The Security-by-Contract-with-Trust framework is based on three corn-stones: the CONNECTor, the *contract* of the CONNECTor, *i.e.* the description of its behaviour, and the *security policy* required by NSs, that consists on the description of the behaviours allowed by NSs. The security policies can either be local (*e.g.* the data sent must be signed) or global (*e.g.* no more than three NS can be connected to a particular NS).

Hence, the Security Enabler expects as input the policies of each NS together with the CONNECTor. According to the NS security policies, the CONNECTor is instrumented, *i.e.* some security probes are added (see D1.3 [30]). This new CONNECTor is returned and it is functionally equivalent to the original one, except that it will send a Security Exception to the Monitoring Enabler when the policy is violated.

Once the Security Enabler receives those inputs, according to the S×C×T workflow [34] in Figure 4.2, it works according to the following steps:

- **Step 1-*Trust Assessment*:** 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 Deliverable D5.2 [28]. 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 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:

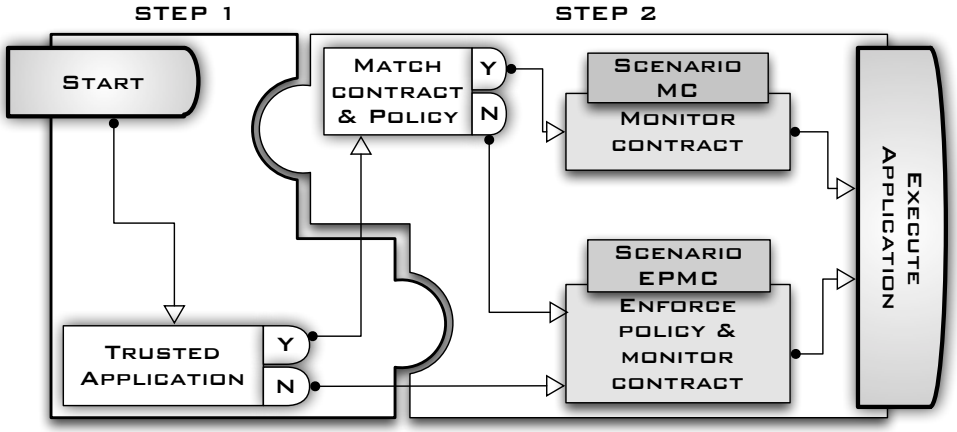


Figure 4.2: The Extended Security-by-Contract Application Workflow

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). In both these scenarios, the monitoring infrastructure consists in a *policy decision point* (PDP) and *Policy Enforcement Points* (PEPs). The PDP holds the actual security state and is responsible for accepting or refusing new actions, for the contract monitoring operations and for the trust vector updating. PEPs are both in charge of intercepting actions to be dispatched to the PDP and preventing the execution of not allowed operations. Following [23, 36], we assume that both contracts and policies are specified through the same formalism.

- **Step 4-*Trust Feedback Inference*:** Finally, the trust module parses the S$\times$C$\times$T produced logs and infers trust feedback.

4.3 Networked System Policy

As we have stated in the previous section, one of the corn-stone of the Security-y-Contract-with-Trust framework is the policy expressed by NSs. In this section, we describe how such policy are structured and we present the XML schema through which each NS is able to define its own policy.

An NS policy is the description of which method calls of the NS are secure or non-secure. Intuitively:

- a policy is defined by a list of rules;

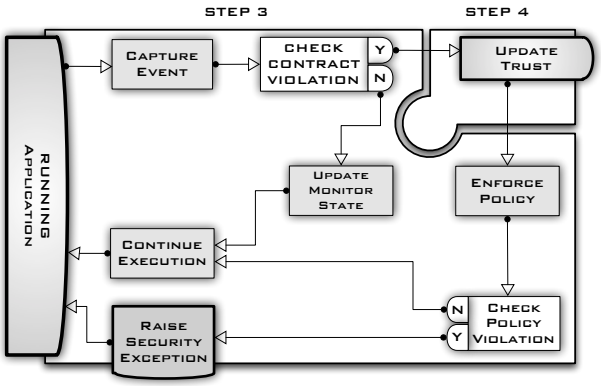


Figure 4.3: The Contract Monitoring Configuration in the MC Scenario

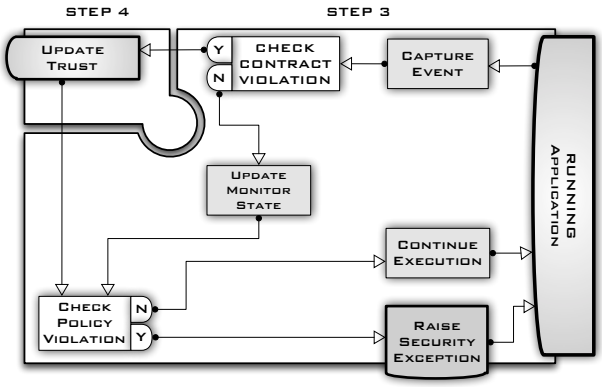


Figure 4.4: The Contract Monitoring Configuration in the EPMC Scenario

- a rule is defined by a signature (*i.e.* the method over which the rule applies), some parameters (*i.e.* the parameters of the method), and some actions;
- an action is defined by a guard (*i.e.* the condition on which the action is triggered) and a reaction (*i.e.* the behaviour of the action when the guard is true).

For instance, a rule could state that if a method m is called with some arguments $args$ satisfying some property, then the policy (which can contain many such rules) is violated. We provide here an XML-based language, which follows the previous structure (the XML schema is given below, and some examples of policies are given in Section 4.6):

Moreover, a policy can define local and global variables, which can be used to specify global policies. For instance, in order to specify that a given method can only be called a number n of times over all the instances of the CONNECTOr, the policy can declare a global variable that will be incremented each time the method is called, and an action will be triggered when the variable will reach n. The Security Enabler is responsible for maintaining a global state, corresponding to all the global variables. Local variables act in the same way, but with a scope reduced to the current instance of the CONNECTOr. Note that, as described in Chapter 5, trust levels are generated by the Trust Enabler as Java classes, that are included in the CONNECTOr. Hence, the policy can directly refer to trust levels, as to any other method of the CONNECTOr. Moreover, a trust feedback, following the definition of Section 5.2.3, can be included in the reaction of a rule, in a way that the Trust Enabler can get the corresponding feedback when a rule is violated.

Finally, each policy is evaluated by the (PDP) (Step 3 of SxCxT), which is a Java module reading the policy and deciding if a specified method violates the policy; its method *allow()* gets as input the method

signature and parameters, and returns a Boolean value specifying whether the method is allowed (true) or not (false).

XML Schema A policy is defined as an XML document, defined using the following schema. For the sake of exposition, we do not reintroduce the usual arithmetic, boolean and string types, represented here by the types `AexpType`, `BexpType` and `SexpType`, respectively, and we just define the base types `typeType` and the complex types `expType`.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:simpleType name="typeType">
      <xs:restriction base="xs:string">
        <xs:enumeration value="int"/>
        <xs:enumeration value="bool"/>
        <xs:enumeration value="string"/>
       </xs:restriction>
</xs:simpleType>

<xs:group name="expType">
  <xs:choice>
    <xs:group ref="AexpType"/>
    <xs:group ref="BexpType"/>
    <xs:group ref="SexpType"/>
  </xs:choice>
</xs:group>
```

A policy can declare some variables, using the element `declarationsType`. As said, these variables can be either local (*i.e.* only for the given CONNECTor) or global (*i.e.* across different CONNECTors).

```
<xs:complexType name="declType">
  <xs:sequence>
    <xs:element name="type" type="typeType"/>
    <xs:element name="varname" type="xs:string"/>
    <xs:element name="scope">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="session"/>
          <xs:enumeration value="local"/>
          <xs:enumeration value="global"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="default">
      <xs:complexType>
        <xs:sequence>
          <xs:group ref="expType"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="declarationsType">
    <xs:sequence>
      <xs:element name="var" type="declType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
```

A rule is defined by the signature of the method over which the rule applies, the parameters of the methods and an action. An action is defined by the guard triggering the action, a possible modification of the variables of the policy, and a reaction, which is either “allow” or “deny”.

```
<xs:complexType name="ruleType">
  <xs:sequence>
    <xs:element name="signature" type="xs:string"/>
    <xs:element name="parameters" type="parametersType"/>
    <xs:element name="tryaction" type="tryactionType"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="parametersType">
  <xs:sequence>
    <xs:element name="par" type="parType" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="parType">
  <xs:sequence>
    <xs:element name="partype" type="typeType"/>
    <xs:element name="parname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="tryactionType">
  <xs:sequence>
    <xs:element name="guard">
      <xs:complexType>
        <xs:group ref="BexpType"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="assign" type="assignType" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="tryreaction">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="allow"/>
          <xs:enumeration value="deny"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="assignType">
  <xs:sequence>
    <xs:element name="varname" type="xs:string"/>
    <xs:group ref="expType"/>
  </xs:sequence>
</xs:complexType>
</xs:schema>
```

Finally, a policy is a set of variable declarations and rules.

```
<xs:element name="policy">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="polid" type="xs:integer" minOccurs="0"/>
      <xs:element name="polname" type="xs:string" minOccurs="0"/>
      <xs:element name="declarations" type="declarationsType" minOccurs="0"/>
      <xs:element name="rule" type="ruleType" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.4 Instrumented Concrete CONNECTor

When a policy needs to be monitored at run-time (for instance when it depends on the value of the arguments), then we need to instrument the concrete CONNECTor in order to add the corresponding calls to the security mechanism. A stated in Chapter 5 of Deliverable D1.3 [30], a concrete CONNECTor between two Networked Systems $NS_i = \langle S_i, M_i, s_{i0}, F_i, \rightarrow, \emptyset, \emptyset, \emptyset \rangle$ is a k-Coloured Application Automaton, which is defined as a tuple $\langle S, M, s_0, F, \rightarrow, \xrightarrow{\gamma}, P, \cong \rangle$ where:

- S maps to the union of the states of each system,
- M maps to the union of the messages that can be sent to (denoted !) or received from (denoted ?) each system. A message is made up of many fields, some are mandatory, others are optional and others can be calculated (e.g., message length).
- s_0 is the initial state of NS_1,
- F maps to the union of the final states of both systems,
- $\rightarrow$ maps to the union of transitions of both systems,
- $\xrightarrow{\gamma}$ defines the transition between states belonging to different systems while performing a translation between the data received from one system, transformed and sent to the other.
- $\cong$ is a partial order relation specifying the semantic equivalence between a message $n \in M_{i=\{1,2\}}$ and a sequence of messages $m_1 \cdots m_k, m_{j=1\ldots k} \in M_{3-i}$.

The Security Enabler follows a transparent approach, and given a k-Coloured Application Automaton $\langle S, M, s_0, F, \rightarrow, \xrightarrow{\gamma}, P, \cong \rangle$, the Security Enabler returns a k-Coloured Application Automaton $\langle S, M', s_0, F, \rightarrow, \xrightarrow{\gamma}, P, \cong \rangle$, where every parameter but M is unchanged, and where M is transformed into M', such that each message in M' corresponds to a message in M with an identical signature.

More formally, the set M contains, among others, output actions of the form $\bar{\beta} =< \bar{op}, i, o >$, meaning that the operation op will be called on the input i and produce the output o. For each method $\bar{op}$ that appears in at least one rule of the security policy of an NS (we assume here that the namespace will automatically avoid any name conflict), we create a new method op', with the same signature, such that op' first checks if $\bar{op}, i, o$ satisfies the policy, in which case the method is called as normal, otherwise a security exception is sent to the Monitoring Enabler.

From an implementation point-of-view, as we described in the following sections, we consider that the set M is enclosed in a Java library package, and therefore the Security Enabler simply returns a new library package, in which the set M' is enclosed. The new library package provides the same interface than the original one, and so there is no need to update the rest of the concrete CONNECTor.

4.5 Security Enabler Implementation overview

In order to implement he Security Enabler, we made the choice to make it callable by any generic RMI Client. This choice has been done in order to satisfy two requirements: make the enabler available through method invocations (one for each service that the enabler exposes), and make it available remotely (in order to be called by partner's modules). We only assume that this RMI client get both NS Policies and the CONNECTor before invoking enabler's methods. The exact way the client should get this information depends on the global final architecture of the system.

The typical interaction flow between the enabler and a client is described in Figure 4.5, and consists of the three following steps:

- **Negotiation:** the RMI client collects the NS policies and sends them to the Security Enabler, which returns a new policy that is an aggregation of all the declarations and rules contained in NS policies.

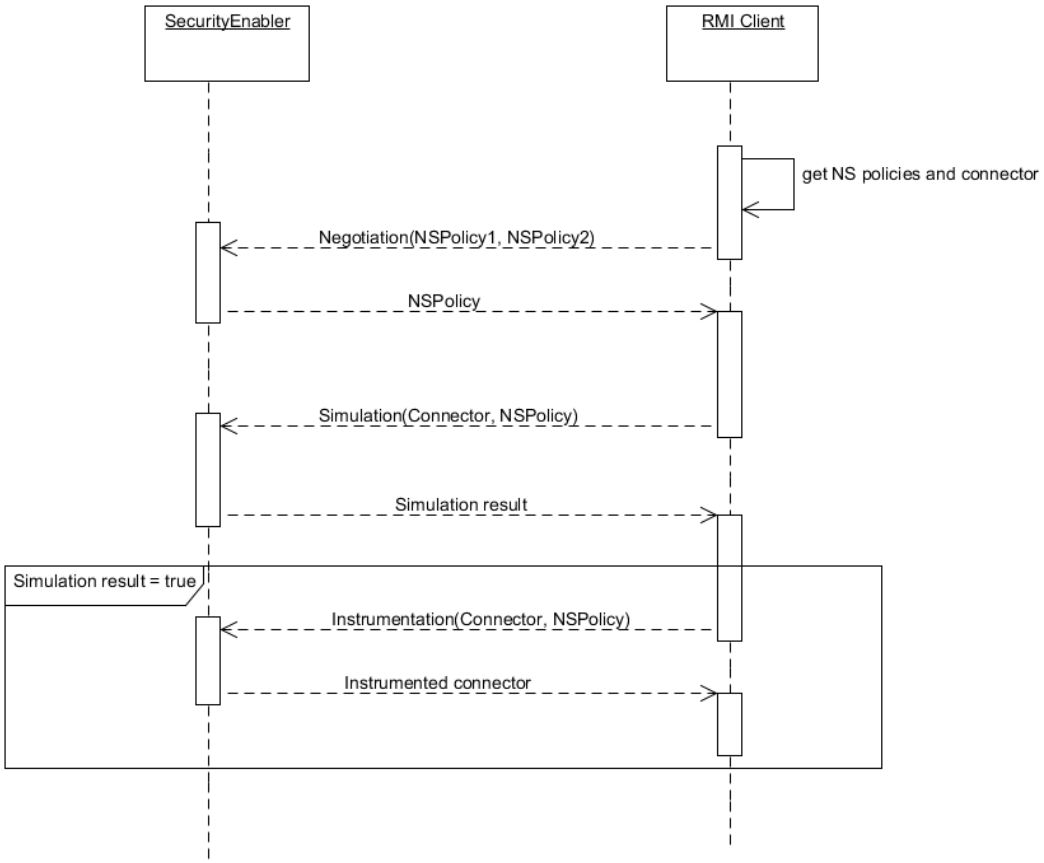


Figure 4.5: The Security Enabler Sequence Diagram

- **Simulation (Step 2 of SxCxT):** the Security Enabler takes as input the CONNECTor and the policy obtained by the Negotiation phase, and decides through simulation matching algorithms if the CONNECTor should be instrumented or if the contract satisfies the policy. The output parameter of the simulation service is a Boolean, describing whether the CONNECTor should be instrumented (true) or not (false); the actual implementation of the simulation matching is currently a dummy one, *i.e.* always returning true, and we plan to implement a concrete simulation matching in future work. Note that this step and the previous could be done in a single step, in order to optimize the communication between the client and the Security Enabler, however we prefer to present them as two different logical steps, since they accomplish different tasks.
- **Instrumentation:** If needed, the Security Enabler provides an instrumented version of the CONNECTor, as described in Section 4.4.

4.5.1 Code instrumentation

The CONNECTor policy is enforced using *code instrumentation*, that is, we enrich the actual code with security checks in a way that is transparent for the CONNECTor. For implementation purposes, we consider here that the methods invoked by the CONNECTor are available in a library package. More specifically, the Security Enabler goes through the following steps:

- All the Java classes are extracted from the library package.
- Each class *"C.class"* is renamed as *"C_wrpd.class"*. A new class "*C.class*" is created (thus replacing the original one) with the same methods, so that "C.class" and "C_wrpd.class" have the same signature.
- Each method body of the new class "C.class" is defined as:
  - If its signature is not contained inside the policy rules, the method body simply calls his corresponding method of the original class, *i.e.* "C_wrpd.class".
  - Else, the method body is defined as:

```
1  if(PDP.allow(<method signature>, <method parameters>)){
2  C_wrpd.method();
3  } else {
4  <Send a security exception event to the
5  monitoring enabler through GlimpseProbe object.>
6  }
```

- The policy and a pre-compiled Java package that contains PDP classes are inserted inside the library package.
- The wrapper and original renamed classes are reinserted inside the library package.
- The modified library package is returned to the RMI client.

Note that at this point, the CONNECTor will not stop when a security violation is detected. Instead, the exception is sent to the Monitoring Enabler, which forwards it to the CONNECT infrastructure, where the decision needs to be made, for instance by the Deployment Enabler, to stop or not the CONNECTor. However, several constraints need to be taken into account for this decision:

- If the CONNECTor keeps executing after a security violation, nothing is guaranteed about future violations. In other words, further policy violations can go undetected. Moreover, a policy violation can have serious consequences.
- By specification, the execution of the CONNECTor must respect its contract. This implies that the Security Enabler cannot modify the behavior of the CONNECTor.
- Stopping a CONNECTor can effectively stop a connection between two NSs.

These different constraints can be conflicting. For instance, consider a drone with the ability to take pictures, and a policy forbidding to take pictures of a particular area. If a NS is connected to the drone and asks to take a picture while the drone is above the forbidden area, then either the drone does not send the picture, which is in violation with the functional contract, or the CONNECTor is stopped, leading the drone to be without controller (and thus possibly crashing), or the picture is sent, thus violating the policy. Clearly, there is no generic answer for this problem, which is beyond the scope of CONNECT, and for now, we assume that the CONNECT infrastructure makes the decision whether to stop or not the CONNECTor.

4.5.2 Prototype

At the implementation level, the Security Enabler is modeled as a Java OSGi Bundle. The following tools have been used for the component development:

- **Eclipse Helios IDE** as develop environment.
- **Eclipse Equinox 3.7** as OSGi container.
- **Apache BCEL 5.2** libraries for the CONNECTor instrumentation.
- **JDom 1.1** libraries for XML Policy parsing.
- **Java RMI** for the invocation of the methods exposed by the Security Enabler.
- **JMS** for the communication with the Monitoring Enabler.

The class diagram in Figure 4.6 represents the Security Enabler structure and the components involved in its lifecycle.

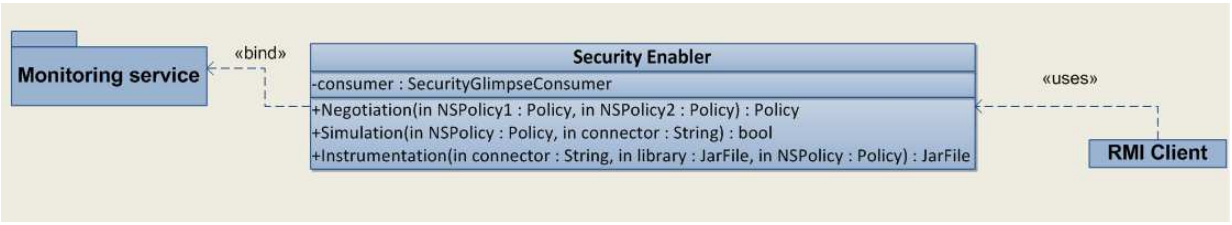


Figure 4.6: The Security Enabler Class

At startup time, the Security Enabler instantiates a *GlimpseConsumer* object and registers it in a specific channel of the Monitoring Enabler, in order to be advised when a Security Exception is thrown by some CONNECTor; the Security Enabler itself during the CONNECTor instrumentation puts a *GlimpseProbe* object inside the libraries used by CONNECTor, which has the task to send this security exception to the Monitoring Enabler's channel.

The instrumentation is done at *bytecode* level, the language in which the Java compiler transforms the Java code before it is re-transformed in machine language by the Java Virtual Machine. We use the **Apache BCEL** libraries to modify the CONNECTor library at bytecode level.

Once the library is instrumented, it can be used by the CONNECTor in the same way it uses the original library, because these modifications are transparent to the user (the instrumented library has the same classes with the same methods).

4.5.3 Interaction with the Monitoring Enabler

When a violation to the security policy is detected, a security exception is raised. This exception is forwarded to the Monitoring Enabler, which can in turn forward it to Deployment Enabler, which can decide whether to stop or not the CONNECTor.

The Security Enabler uses the *Glimpse API* provided by the partner responsible for the Monitoring Enabler; in particular two objects are used:

- A **Glimpse Probe** that sends an event to a specific channel of the Monitoring Enabler.
- A **Glimpse Consumer** that is registered to a specific channel and receives all the messages sent by any Probe to this channel.

At startup time, the Security Enabler instantiates a *GlimpseConsumer* object and registers it in a specific channel of the Monitoring enabler, in order to be advised when a Security Exception is thrown by some CONNECTor.

The *GlimpseProbe*, instead, is instrumented inside the libraries used by CONNECTors, and it is responsible for sending Security Exception events to the Monitoring Enabler when some method calls are violating the policy, on the same channel where the Security Enabler Glimpse Consumer is registered.

In this way, the Security Enabler can be informed when a CONNECTor is violating the policy, and can make some reaction (currently it only prints to screen the exception message).

4.6 GMES Example

In order to illustrate the different concepts presented here, we introduce a very simple example, inspired by the GMES scenario, presented in Deliverable D6.3 [32]. GMES (Global Monitoring for Environment and Security) is the European Programme for the establishment of a European capacity for Earth Observation started in 1998. A particular thematic of this programme is the emergency management service, which directs efforts towards a wide range of emergency situations; in particular it covers catastrophic circumstances.

In the context of the CONNECT project, we take as example a joint forest-fire operation. Each European nation has its own organization, resources and strategies to handle such fires so when a neighbor country provides support it is often with different means, resources or protocols. In order to illustrate the Security Enabler, we consider the case where the Command Center (CC), in charge of organizing the operation, wants to communicate with an Unmanned Aerial Vehicle (UAV), for instance to take pictures of the area where the fire is. These two entities are respectively represented by the Networked Systems NS_{cc} and NS_{uav}. Note that for the sake of exposition, we simplify the interfaces of such systems, and only focus on the points relevant to the Security Enabler. A real-world implementation would be of difference scale, but with the same complexity.

- NS_{cc} has two methods, `sendRequest(caller, uav)`, which sends a data request from `caller` to `uav` and `uploadData(data)`, which stores `data`.
- NS_{uav} has two methods, `dataRequest(id)`, which considers a data request from `id`, and a method `dataSend(data, id)`, which sends `data` to `id`.

In this case, the CONNECTor simply associates `sendRequest` and `dataRequest` on the one hand, and `uploadData` and `dataSend` on the other hand. We therefore assume that the CONNECTor is associated with a package P, providing the two following classes:

```
1  package P;
2
3  class CC {
4     public static void sendRequest(String caller, String uav) {
5     // Call to the actual method
6     }
7     public static void uploadData(String data) {
8     // Call to the actual method
9     }
10 }
11
12 class UAV {
13    public static void dataRequest(String id) {
14    // Call to the actual method
15    }
16    public static void dataSend(String data, String id) {
17    // Call to the actual method
18    }
```

```
19 }
```

We also consider two simples policies, P_{cc} and P_{uav}, for N_{cc} and N_{uav}, respectively:

- P_{cc} states that when `uploadData(data)` is called, `data` must be signed:

```
<policy id=Pcc>
  <rule>
    <signature>uploadData</signature>
    <parameter>data</parameter>
     <action>
       <guard> isSigned(data) </guard>
       <reaction> allow </reaction>
     </action>
  </rule>
</policy>
```

- P_{uav} states that when `dataRequest(id)` is called, `id` must have a proper licence.

```
<policy id=Puav>
  <rule>
    <signature>dataRequest</signature>
    <parameter>id</parameter>
     <action>
       <guard>hasLicence(id)</guard>
       <reaction> allow </reaction>
     </action>
  </rule>
</policy>
```

The generated policy of the CONNECTor is:

```
<policy id=Pconnect>
   <rule>
     <signature>Pcc :uploadData</signature>
     <parameter>data</parameter>
     <action>
       <guard> isSigned(data) </guard>
       <reaction> allow </reaction>
     </action>
  </rule>
  <rule>
    <signature>Puav :dataRequest</signature>
    <parameter>id</parameter>
    <action>
      <guard>hasLicence(id)</guard>
      <reaction> allow </reaction>
    </action>
  </rule>
</policy>
```

Finally, the CONNECTor is instrumented as follows.

```
1  package P;
2
3  class CC {
4     private CCunsafe wrap;
5
6     public static void sendRequest(String caller, String uav) {
7        // No security check required
```

```
8       wrap.sendRequest(caller, uav);
9     }
10    public static void uploadData(String data) {
11        if (org.connect.iit.security.PDP.allow("uploadData", data))
12            wrap.uploadData(data)
13        else
14            throw new SecurityException("unsigned data");
15    }
16 }
17
18 private class UAV {
19    public static void dataRequest(String id) {
20        if (org.connect.iit.security.PDP.allow("dataRequest", id))
21            wrap.dataRequest(id);
22        else
23            throw new SecurityException("no licence");
24    }
25    public static void dataSend(String data, String id) {
26    // No security check required
27        wrap.dataSend(data, id);
28    }
29 }
30
31
32 private class CC_unsafe {
33    public static void sendRequest(String caller, String uav) {
34    // Call to the actual method
35    }
36    public static void uploadData(String data) {
37    // Call to the actual method
38    }
39 }
40
41 private class UAV_unsafe {
42    public static void dataRequest(String id) {
43    // Call to the actual method
44    }
45    public static void dataSend(String data, String id) {
46    // Call to the actual method
47    }
48 }
```

Figure 4.7 illustrates the usage of the Security Enabler: on the left is a (simplified) interface of the drone, allowing to take pictures, without the instrumentation: the picture can be taken without restriction. On the right, the same interface is presented, but the underlying code has been instrumented by the Security Enabler, and when the user tries to take a picture violating the policy, the action is blocked and an error message is raised. It it worth noticing that the interface looks identical in both situations, since the Security Enabler only wraps around existing code. Note also that in this example, the CONNECTor is not stopped, as it is not the responsibility of the Security Enabler to stop it.

4.7 Conclusions and Future Work

During the third year, the work on Security in WP5 focused on defining the mechanism for the run-time enforcement of NS policies in the CONNECTor. In order to do so, we have presented the *instrumentation* approach, which consists in substituting each method call to an NS method in the concrete CONNECTor by a method call with a similar interface, but which first checks if the security policy is satisfied. Any violation to the security policy results in sending a security exception to the Monitoring Enabler. We have successfully implemented a prototype of this mechanism, and have illustrated it with a simple example inspired from the GMES scenario.

During the fourth year, we need to adapt the existing prototype of the policy/contract matching verifier in the context of CONNECT. We plan to cope with this task in the remaining period, since in the third year

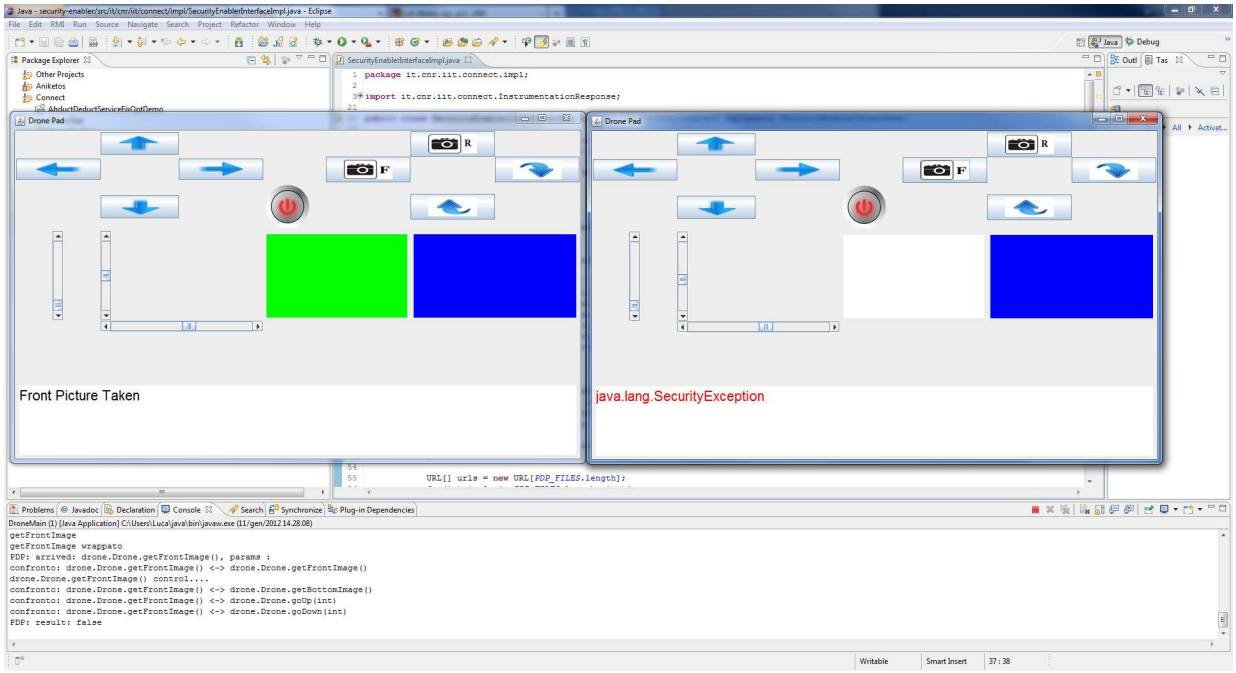


Figure 4.7: Screenshot of the Application

we mostly focused on the code instrumentation tool, as it was the most critical part in terms of integration with the rest of the architecture.

Indeed, the policy/contract matching verifier is quite transparent for the user, and the current approach can simply be considered as using a verifier that always states that the policy and the contract do not match, thus meaning that the policy must be enforced. We can therefore bring in the verifier in an incremental way, such that in some cases, when the policy and the contract match, no monitoring is required.

5 Trust Management in CONNECT

In Deliverable D5.2 [28], we introduced TMDL (Trust Model Description Language) as the basis to express and to compose a wide range of trust management systems and thereby support trust management across heterogeneous networked systems. In D5.2 we have illustrated the proposed language through the Terrorist Alert scenario (taken from Deliverable D6.2 [29]), where two heterogeneous trust models (i.e., Guard and Police) are modeled, composed and mediated. The composition is specified in terms of mapping rules between roles of the original models. Rules are then processed by a set of mediation algorithms to overcome the heterogeneity of the trust metrics, relations and operations associated with the composed trust models.

During Y3, we enhanced and improved the TMDL language and provided its complete and final XML representation. We also implemented several dedicated tools that (i) guide developers to check and create a valid TMDL description; (ii) automatically generate from such description the Java code of the corresponding trust management system; and (iii) enables the composition of any given trust management system according to given mapping rules.

5.1 Trust Enabler Overview

In previous WP5 deliverables, we introduced the CONNECT trust management system and focused on the trust assessment of the NSs in order to:

- Enable CONNECT Enablers to retrieve/update the trustworthiness of any discovered NS.
- Enable each CONNECTed NS to use its trust management system to assess the trustworthiness of any NS that it is CONNECTed to.

Networked Systems are heterogeneous and often implement different trust management systems. The CONNECT trust management system hence needs to be enriched continuously to be able to interact and interoperate with the trust management system of discovered NSs. To do so, we design the Trust Enabler to support and continually adapt and enhance the CONNECT trust management system.

The Trust Enabler is not responsible for monitoring the behavior of NSs, but instead provides an interface to other Enablers to be warned of any monitored misbehavior, for instance: Security Enabler warn the Trust Enabler if any policy is violated and hence the trustworthiness of the corresponding NS is decreased.

In Figure 5.1, we show a sketch of the CONNECT Trust Management System. It is mainly made up of the following roles:

- The Trust Enabler Role (R_{TE}): It represents the entry point of the trust management system. It provides to the CONNECT Enablers, all trust operations that allow to retrieve and update the trustworthiness value of the discovered NSs.
- The NS Role (R_{NS}): It is the role given by the Trust Enabler to all discovered NSs. To do so, each discovered NS is considered by the CONNECT trust management system as a participant that plays the R_{NS} role.
- The Trust Enabler Mediation Role (R_{mTE}): this role performs a mediation between the CONNECT trust management system and the NS trust management system. In other words, it plays the role of a recommender that can be requested by the R_{TE} to retrieve NS's trustworthiness values and also propagate any CONNECT feedback.
- The NS mediation Role (R_{mNS}): Similarly to R_{mTE}, the R_{mNS} is a mediator that plays the roles of trusted recommenders to bridge the trust management systems of the CONNECTed NSs, transparently.

The Trust Enabler is built with specific modules that are triggered automatically when an NS is discovered (i.e., the "TMDL Loader") and when a CONNECTor is deployed (i.e., the "Mapping Loader"). The "TMDL loader" processes the given TMDL description of a discovered NS and produces a TMDL description of a mediator(R_{mTE}) that enables the R_{TE} role to interact with the trust management system of the

discovered NSs. Similarly, the "Mapping Loader" processes the TMDL description of the CONNECTed NSs and generates a TMDL description of a trust mediator (R_{mNS}) that is able to compose the trust management systems of these NSs.

Then comes the role of the "Trust Role Generator" module which processes the TMDL description of the mediators and generates the corresponding Java code. Instances of these mediators are then deployed within the CONNECT Trust Management System and bound to the Interaction Enabler to be able to interact using any middleware protocol handled by that Enabler.

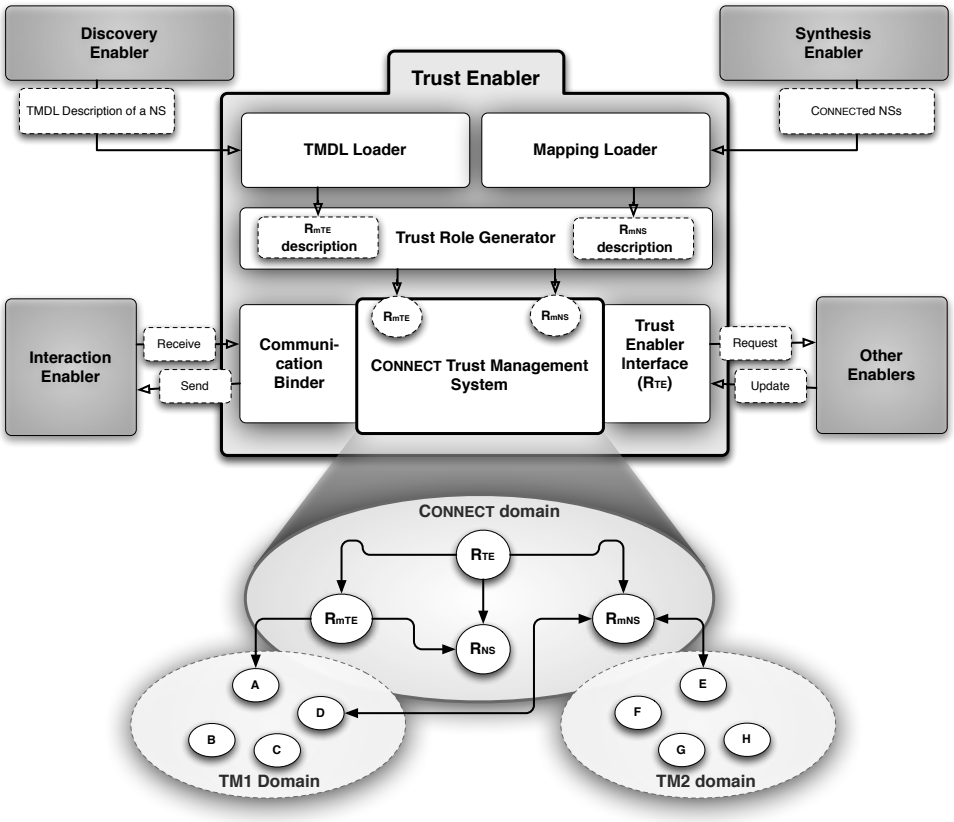


Figure 5.1: The Trust Enabler Architecture

As described above, the Trust Enabler requires some engines able to process TMDL descriptions, generate trust mediators and deploy instances of those mediators into the CONNECT trust management system.

The next section details the final version of the XML syntax of TMDL, while Section 5.3 illustrates the composition of trust management systems based on given TMDL descriptions. Then, Section 5.4 introduces all the tools that have been implemented to support the Trust Enabler, i.e., tools that process TMDL descriptions and perform code generation, deployment and composition. Those tools will be concisely composed in the next period to implement the Trust Enabler.

5.2 Trust Model Description Language

In order to recall the functioning of the Trust Model Description Language, let us dissect the following very generic definition of a trust relationship: *A trustor trusts a trustee with regard to its ability to perform a specific action or to provide a specific service* [44].

- *Trustor* and *Trustee*: are participants of the trust model and can be identified by the *Roles* they play.

- *Trustor trusts a trustee*: represents a trust *Relationship* that links a participant (trustor) to another (trustee). The action *trust* is a *Decision* that is based on the evaluation of a specific trust *Metric*.

- *Perform a specific action* or *Provide a specific service*: represent the *Context* where a specific trust relationship is established.

In summary, to define a trust model, we have to identify the following trust elements: (i) The Domains where trust elements are defined (see Section 5.2.1) ; (ii) The Roles played by the participants involved (see Section 5.2.2) ; (iii) The Metrics for assessing the trustworthiness of the participants (see Section 5.2.3) ; (iv) The Contexts where trust is required (see Section 5.2.4) ; (v) The trust-related Relations that link participants (see Section 5.2.5) and finally (vi) The Decisions provided by the model from the assessment of trust relationships (see Section 5.2.6).

All the TMDL trust elements can be enriched semantically by making reference to specific ontologies. Since TMDL is an XML-based language, semantic references are annotated with SAWSDL[1] components, namely: (i) *modelReference* to identify the corresponding ontological concept, (ii) liftingSchemaMapping and (iii) loweringSchemaMapping to specify the mappings between semantic data and trust element data (see Table 5.1).

Element	Description	Type	Optional	Instances
annotation	TMDL Element	Tag	Yes	One
—modelReference	SAWSDL Attribute	URL	No	One
—loweringSchemaMapping	SAWSDL Attribute	URL	Yes	One
—liftingSchemaMapping	SAWSDL Attribute	URL	Yes	One

Table 5.1: SAWSDL Annotation: XML Definition

We define a custom scenario from the GMES use case (see D6.3 [32]) in order to illustrate the feasibility of the different materials provided by this chapter. We consider a scenario that involves four stakeholders namely, a Pilot Fireman, a Chief Fireman, a Drone and a reputation Server. This server maintains and provides the reputation of the Firemen that are authorized to pilot Drones. The Pilots' reputation is assessed and updated according to the number of steering hours and the Drone feedback that reports any mis-control that might be made by the Pilot. The system rewards best Pilots with a First Class Grade to confirm their good reputation and allows them to be selected by their Chief for sensitive and difficult missions.

5.2.1 Trust domain

The trust domain represents a logical structure that administrates a consistent trust management system. Thus, describing a trust model starts with defining at least one domain.

Element	Description	Type	Optional	Instances
domains	TMDL Element	Tag	No	One
—domain	TMDL Element	Tag	No	Multiple
——name	TMDL Attribute	String	No	One
——annotation (Tab. 5.1)	TMDL Element	Tag	Yes	One
——Roles (Tab. 5.3)	TMDL Element	Tag	NO	One
——Metrics (Tab. 5.4)	TMDL Element	Tag	NO	One
——Contexts (Tab. 5.5)	TMDL Element	Tag	NO	One
——Relations (Tab. 5.6)	TMDL Element	Tag	NO	One
——Decisions (Tab. 5.10)	TMDL Element	Tag	NO	One

Table 5.2: Trust Domain: XML Definition

[1] SAWSDL: Semantic Annotations for WSDL, http://www.w3.org/2002/ws/sawsdl/

As illustrated in Table 5.2, a trust domain is specified by a *name* and might make a reference to a semantic concept (*annotation* tag). Inside each Trust Domain we define all the elements that are managed by this domain, namely: the list of Roles, the list of Metrics, the list of Contexts, the list of Relations and the list of Decisions.

5.2.2 Trust role

A trust role is a high level representation of a group or a category of participants from the standpoint of trust management, in a similar way to the role-based access control model [41]. The Role can specify a category of users (e.g., Fireman, customer, etc.), a category of processes that act on behalf of the user (e.g., agent) or a category of resources (e.g., reputation server, Drone, file, service, etc.). Hence, participants are identified by the role they play.

A trust domain can contain several Trust Roles. Each Role has a name, might refer to a semantic concept and can inherit from sup-role(s) (see Table 5.3). We define inheritance between roles, so that a derived role extends all the sup-role capabilities, For instance, in our use case, we can identify different categories of Firemen such as: Pilot and Chief, which perform all the generic Fireman behavior.

The inheritance property is very useful to avoid duplicating similar behaviors among similar roles. However, sup-roles might not correspond to any endpoint entity ; they just define a common behavior. As in programming languages, we define such roles as being abstract.

Element	Description	Type	Optional	Instances
roles	TMDL Element	Tag	No	One
—role	TMDL Element	Tag	No	Multiple
——name	TMDL Attribute	String	No	One
——abstract	TMDL Attribute	Boolean	No	One
——inheritance	TMDL Attributes	XPath (role)	Yes	Multiple
——annotation	TMDL Element	Tag	Yes	One

Table 5.3: Trust Role: XML Definition

In the following XML fragment we define the Fireman (line 2), the Pilot (line 3-5) and Chief (lines 6-8) roles:

```
1 <roles>
2  <role name="Fireman" abstract="true"/>
3  <role name="Pilot">
4    <inheritance role="//@domains/@domain.0/@roles/@role.0"/> //GMESFireman
5  </role>
6  <role name="Chief">
7    <inheritance role="//@domains/@domain.0/@roles/@role.0"/> //GMESFireman
8  </role>
9 </roles>
```

For the sake of clarity, in further XML examples, we replace ”XPath” values that refer to an element by its domain name and its name, e.g., ”//@domains/@domain.0/@roles/@role.0” is annotated by ”GMES-Fireman”.

5.2.3 Trust metric

In order to assess the trustworthiness of any participant, the application has to provide the metrics used for the trust assessment process. In TMDL, each Trust Metric is identified by a name and has a type that represents the type of data used to assess and measure trust (see Table 5.4).

There is no generally accepted metric type. It can be a binary value, a probability value, a label value, a complex value, etc. Due to this variety of types, we chose to rely on the flexibility of the complexType element of the XML schema (XSD). Therefore, a Trust Metric can be defined as an enumeration (semantic

label), a boolean, a decimal (float), an integer, a date, a time, or a complex sequence of these last enumerated types.

Element	Description	Type	Optional	Instances
metrics	TMDL Element	No	No	One
—metric	TMDL Element	No	No	At least One
——name	TMDL Attribute	No	No	One
——annotation(Tab. 5.1)	TMDL Element	Tag	Yes	One
——complexType	XSD Element	Tag	No	One

Table 5.4: Trust Metric: XML Definition

Example: In the XML fragment below, we define the metric *Feedback*, which includes: the identifier (line 4) of the transaction subject to the feedback along with the list of possible Drone miss-control level (lines 5-11).

```
1  <metric name="Feedback">
2    <complextype>
3      <sequence>
4        <element name="IDTransaction" type="integer"/>
5        <element name="ErrorLevel" type="integer">
6          <restriction>
7            <enumeration value="0"/>
8            <enumeration value="1"/>
9            <enumeration value="2"/>
10         </restriction>
11       </element>
12     </sequence>
13   </complextype>
14 </metric>
```

5.2.4 Trust context

The trust context is the situation in which trust relationships are meaningful and relevant. For instance, in the context of "Car repairs", customers have to trust a mechanic to fix their car, or in the context of "FireFighting" Chiefs have to trust Pilot to steer Drones efficiently.

As for the Trust Role, Trust Context is simply identified by a name and should refer to a given ontology.

Element	Description	Type	Optional	Instances
contexts	TMDL Element	Tag	No	One
—context	TMDL Element	Tag	No	At least One
——name	TMDL Attribute	String	No	One
——annotation(Tab. 5.1)	TMDL Element	Tag	Yes	One

Table 5.5: Trust Context: XML Definition

5.2.5 Trust relation

Now that Roles, Metrics and Contexts have been defined, we can describe trust Relations. In fact, relationships established by participants are identified in the trust model by relations between Roles. We have defined three types of relations [66], namely: (i) relationOneToOne which categorizes the trustor's (One) personal opinion regarding its trustee (One), (ii) relationOneToMany represents a transitive relation that a trustor (One) can establish with an unknown trustee through trusted recommenders (Many) and

Element	Description	Type	Optional	Instances
relations	TMDL Element	Tag	No	One
—relationOneToOne —relationManyToOne —relationOneToMany	TMDL Element	Tag	Yes	Multiple
——name	TMDL Attribute	String	No	One
——annotation(Tab. 5.1)	TMDL Element	Tag	Yes	One
——trustor	TMDL Attribute	XPath(role)	No	One
——trustee	TMDL Attribute	XPath(role)	No	One
——metric	TMDL Attribute	XPath(metric)	No	One
——context	TMDL Attribute	XPath(metric)	No	One
——derivedFrom	TMDL Attribute	XPath(relation)	Yes	Multiple
——operations (Tab. 5.7- 5.9)	TMDL Element	Tag	Yes	One

Table 5.6: Trust Relation: XML Definition

(iii) relationManyToOne which means that third party recommenders (Many) trust a trustor to manage the reputation of a trustee (One).

As illustrated in Table 5.3, the three relation types have the same overall structure; they have a trustor and a trustee that refer to predefined roles and are defined within a given context and evaluated with a given metric.

Thus, each relationship between participants is an instance of a defined relation, where the trustor and the trustee participants respectively have to play the trustee and the trustor role of the corresponding relation in the model.

An additional attribute (derivedFrom) can be set to state explicitly whether the relation is derived from other relations in such a way that any modification of any relation that this relation is derived from systematically affects its metric value.

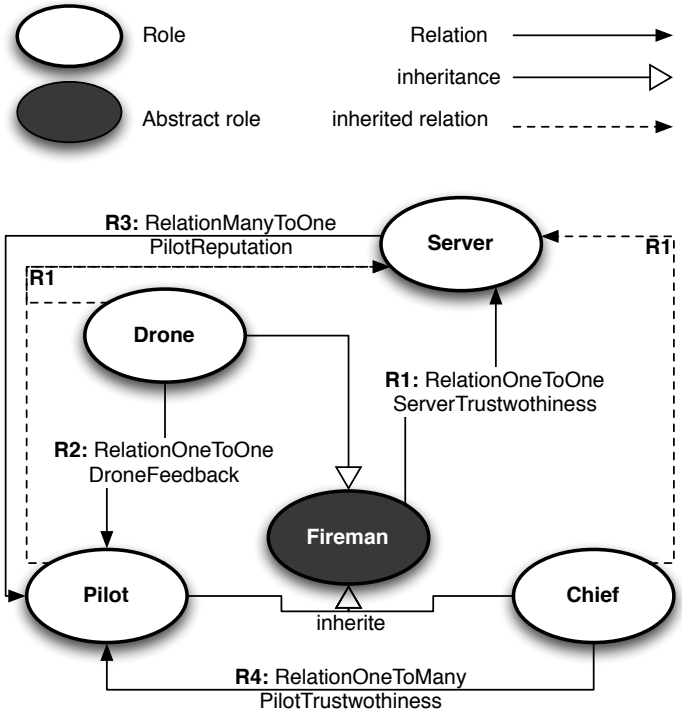


Figure 5.2: GMES Trust Relations

Figure 5.2 shows all the trust relations that link GMES roles. Firemen trust the reputation server to manage Pilots' reputation. We represent this as a RelationOneToOne, which we call "ServerTrustworthiness". The Server manages the reputation of Pilots through a ManyToOne relation, which we call PilotReputation. As introduced in the use case, these reputations are updated respectively from feedback received from Drones. The trust relations are illustrated in Figure 5.2 by a OneToOne relation named DroneFeedback.

Each relationship is evaluated by a metric value that have to be set or retrieved. To do so, two kinds of operations are defined, namely, operationGet and operationSet. Given a target relation $TRelation$, the operationGet and operationSet of TRelation are implemented by the related trustor as follows:

- *MetricTypeOfRelation TrustorRole::getTRelation(TrusteeRole trustee)*
  Any participant that plays the role $TrustorRole$ can perform the operation"get" to retrieve the trust value corresponding to the $trustee$ that plays the $truseeRole$ over the $TRelation$. This trust value is of the metric type of "TRelation" (i.e., $MetricTypeOfRelation$).

- *void TrustorRole::setTRelation(TrusteeRole trustee, MetricTypeOfRelation trustValue)*
  Any participant that plays the $TrustorRole$ can perform the operation "set" to update or set the trust value of an instance of the $TRelation$ with the input $trustValue$.

Often, operations have to be invoked remotely by other roles. For instance, the Chief can ask the server to get the reputation of a Pilot. To do so, the reputation server has to enable Chiefs to invoke the operation getPilotReputation remotely. In such a situation, this operation must be defined with the related relation in order to specify how accessible they are. This feature is enabled with the AccessFrom attribute. The AccessFrom attribute specifies OneToOne relations from which their trustors are directly authorized to invoke the required trustee's operation remotely (see Table 5.7 - 5.9).

However, though the meaning of the operations (i.e., get and set) is the same whatever the relation is, performing the operation is related to the type of the relation and might need to be defined with specific "parameters". For some operations the list of parameters is implicit, and hence no definition is required. On the other hand, those that are performed over explicit parameters have to be defined under the element operations (see Table 5.7 - 5.9).

In the following, for each type of relation we list: (i) all possible operations with an implicit list of parameters and (ii) all possible operations with an explicit list of parameters:

RelationOneToOne :

- Operations with an implicit list of parameters:
  - *MetricTypeOfRelation TrustorRole::getTRelation(TrusteeRole trustee)*
    The operationGet of the relationOneToOne does not require any additional parameter to retrieve the trust value of the trustee since it simply returns the local trust value of the specified trustee.
  - *void TrustorRole::setTRelation(TrusteeRole trustee)*
    The trustor initiates a relationship with the specified trustee with any a priori knowledge (bootstrapping).
  - *void TrustorRole::setTRelation(TrusteeRole trustee, MetricTypeOfRelation trustValue)*
    The trustor sets or updates its relationship with the trustee by a specific and given trustValue.
- Operations with an explicit list of parameters:
  - *void TrustorRole::setTRelationWithWRelation(TrusteeRole trustee, WRelation withRL)* The trustor sets or updates its relationship with the trustee with the trust value of an instance (withRL) of the

Relation WRelation that assesses that trustee. For instance, Alice can update her relationship with Bob after receiving her friend Charlie's opinion about Bob. For relations of type OneToOne, this operation is optional and is only specified if it has to be performed by the trust management system. An operationSet hence has to be defined in order to fill the $withRelation$ attribute that refers to the corresponding relation (i.e., WRelation) (see Table 5.7).

In TMDL, all the parameters that have to be explicitly provides to define any operation related to a OneToOneRelation (e.g., accessRelation and withRelation) are given by Table 5.7.

Element	Description	Type	Optional	Instances
relationOneToOne	TMDL Element	Tag	Yes	Multiple
...				
—operations	TMDL Element	Tag	Yes	One
——operationGet	TMDL Element	Tag	Yes	One
———accessFrom	TMDL Attribute	XPath(relation)	No	Multiple
——operationSet	TMDL Element	Tag	Yes	One
———annotation(Tab. 5.1)	TMDL Element	Tag	Yes	One
———accessFrom	TMDL Attribute	XPath(relation)	Yes	Multiple
———withRelation	TMDL Attribute	XPath(relation)	Yes	Multiple

Table 5.7: OneToOneRelation Operation: XML Definition

RelationManyToOne :

- Operation with an implicit list of parameters:
  - *MetricTypeOfRelation TrustorRole::getTRelation(TrusteeRole trustee)*
    As for the RelationOneToOne, the operationGet does not require any input parameter to retrieve the trustee's trust value since it is managed locally by the trustor of the relation.
- Operation with an explicit list of parameters:
  - *void TrustorRole::setTRelationWithWRelation(TrusteeRole trustee, WRelation withRL)*
    Similarly to the RelationOneToOne, the RelationManyToOne is updated with the recommendations (i.e., withRL) of a trustee's recommenders. For instance, the Server updates the reputation of Pilot with Drones recommendations feedback. However, unlike the RelationOneToOne, this type of relation requires the definition of an operationSet because it is the only way to describe how to set and update that kind of relation (see Table 5.8). This operation must also be accessible to participants that provide their recommendations, hence the accessFrom attribute is required.

In Table 5.8, we provide the XML Syntax of all the operations related to a ManyToOneRelation that are defined with an explicit list of parameters (e.g., accessRelation and withRelation).

In the following XML example, we define the relation PilotReputation and we give details of the operationGet (line 5) that enables Chief to retrieve Pilots' reputations and the operationSet (lines 6-7) that enables Drones to give their feedback and to update the corresponding Pilot' reputation:

```
1 <relationManyToOne name="PilotReputation"
2       trustorRole="GMESServer" trusteeRole="GMESPilot"
3       withinContext="GMESFireFithing" metricType="GMESReputation">
4   <operations>
5     <operationGet accessFrom="GMESServerTrustworthiness"/>
6     <operationSet accessFrom="GMESServerTrustworthiness"
7                   withRelation="GMESDroneFeedback"/>
8   </operations>
9 </relationManyToOne>
```

Element	Description	Type	Optional	Instances
relationManyToOne	TMDL Element	Tag	Yes	Multiple
...				
—operations	TMDL Element	Tag	Yes	One
——operationGet	TMDL Element	Tag	Yes	One
———accessFrom	TMDL Attribute	XPath(relation)	No	Multiple
——operationSet	TMDL Element	Tag	No	One
———annotation(Tab. 5.1)	TMDL Element	Tag	Yes	One
———accessFrom	TMDL Attribute	XPath(relation)	No	Multiple
———withRelation	TMDL Attribute	XPath(relation)	No	Multiple

Table 5.8: ManyToOneRelation Operation: XML Definition

RelationOneToMany :

- Operations with an explicit list of parameters:
  - *MetricTypeOfRelation TrustorRole::getTRelationFromFRelation(TrusteeRole trustee)*

    A OneToMany relation is not managed locally by the trustor of this relation, it has to be retrieved transitively from recommenders. This type of operation is required and needs to be described by giving the relations that perform this operation. To do so, two attributes have to be provided, namely: fromRelation that refers to the relation (FRelation) linking the trustor to the recommenders and the withRelation that makes reference to all relations that are expected, by the trustor, to link the recommender with the requested trustee.

  - *void TrustorRole::setTRelationFromFRelationWithWRelation(TrusteeRole trustee, WRelation withRL)*

    The relationOneToMany is not managed by its trustor, and hence the trustor cannot directly update the relation with any feedback (withRL), but it can be propagated. In other words, the withRL will serve to update all the instances of the FRelation and be further propagated to the recommenders. In their turn, the recommenders update their relationships that are sensitive to withRL. Therefore, as described in Table 5.9, the operationSet is defined with two attributes: fromRelation and withRelation.

In Table 5.9, we detail the TMDL Syntax of all the operations related to a ManyToOneRelation that have to be defined with an explicit list of parameters (e.g., accessRelation, fromRelation and withRelation).

Element	Description	Type	Optional	Instances
relationOneToMany	TMDL Element	Tag	Yes	Multiple
...				
—operations	TMDL Element	Tag	Yes	One
——operationGet	TMDL Element	Tag	Yes	One
———annotation(Tab. 5.1)	TMDL Element	Tag	Yes	One
———accessFrom	TMDL Attribute	XPath(relation)	Yes	Multiple
———fromRelation	TMDL Attribute	XPath(relation)	No	One
———withRelation	TMDL Attribute	XPath(relation)	No	Multiple
——operationSet	TMDL Element	Tag	Yes	One
———annotation(Tab. 5.1)	TMDL Element	Tag	Yes	One
———accessFrom	TMDL Attribute	XPath(relation)	Yes	Multiple
———fromRelation	TMDL Attribute	XPath(relation)	No	One
———withRelation	TMDL Attribute	XPath(relation)	No	One

Table 5.9: OneToManyRelation Operation: XML Definition

Example: Chiefs assess (operationGet) the trustworthiness of Pilots transitively (PilotTrustworthiness) by composing the Pilot's reputation(fromRelation: GMESPilotReputation) with the trustworthiness value that they give to the server (withRelation:GMESServerTrustWorthiness) that returns that reputation.

The following XML fragment describes the example below. The transitive relation is detailed in lines 1-3 and the corresponding operationGet and operationSet are respectively defined in lines 5-6 and lines 7-8.

```
1 <relationOneToMany name="PilotTrustwothiness"
2       trustorRole="GMESChief" trusteeRole="GMESPilot"
3       withinContext="GMESFireFighting" metricType="GMESTrustworthiness">
4   <operations>
5               <operationGet fromRelation="GMESServerTrustworthiness"
6                                   withRelation="GMESPilotReputation"/>
7   </operations>
8 </relationOneToMany>
```

5.2.6 Trust decision

The Trust Decision represents the output of the trust model, i.e., all the decisions that are related to and based on assessed trust relationships. For instance, according to the user's reputation a decision could be: In order to consider a user trustworthy (i.e., "isTrustworthy"), his/her reputation has to be higher than 0.5 or a GMES Server rewards Pilots with a "First Class Grade" if their reputation is higher than 75% (i.e. 0.75), etc.

As illustrated in Table 5.10, a trust decision is defined by a name within a specific domain and might refer to a given ontology. The implementation logic that allows a decision to be made is described as an operation called "operationMake".

Given a target decision $TDecision$, the operationMake is performed by the role that is referenced by the Element $decisionMaker$ as follows:

- Object DecisionMaker::makeTDecision(decisionTarget target):
  The $decisionMaker$ is able to make a decision concerning participants that play the role $DecisionTarget$ with relations that assess that target (i.e., relationships that have the target as trustee). As illustrated in Table 5.10 an operationMake has at list to be defined with the $decisionMaker$, the $decisionTarget$ and the $withRelation$ parameters.

Element	Description	Type	Optional	Instances
decisions	TMDL Element	Tag	Yes	One
—decision	TMDL Element	Tag	No	At least One
——name	TMDL Attribute	String	No	One
——annotation(Tab. 5.1)	TMDL Element	Tag	Yes	One
——operations	TMDL Element	Tag	No	One
———operationMake	TMDL Element	Tag	No	Multiple
———accessFrom	TMDL Attribute	XPath(relation)	Yes	Multiple
————decisionMaker	TMDL Attribute	Xpath(role)	No	One
————decisionTarget	TMDL Attribute	Xpath(role)	No	One
————withRelation	TMDL Attribute	Xpath(relation)	No	Multiple

Table 5.10: Trust Decision: XML Definition

In the following example, we illustrate the decision related to the "First Class Grade" (line 1) which concerns the GMES Pilot (line 5). This grade is given according to the Pilot reputation (line 5) through the relation GMESServerTrustworthiness (line 4).

```
1 <decision name="FirstClassGrade">
2   <operations>
3     <operationMake decisionMaker="GMESServer"
4       accessFrom="GMESServerTrustworthiness"
5       decisionTarget="GMESPilot" withRelation="GMESPilotReputation"/>
```

```
6   </operations>
7 </decision>
```

5.3 Composing Trust Models

The aim of composing two different trust models (TM_X and TM_Y) is to provide participants of a given model (which we call target model TM_T) the ability to interact (i.e., create, assess and retrieve relationships) with participants that come from another model (which we call source model TM_S).

In D5.2, we introduced a trust-centric composition process that relies on a set of mapping rules Ψ_{xy} defining how roles (i.e., r_S) from the source model are mapped ($\bowtie$) to roles(i.e., r_T) from the target model, as follows:

$$\begin{aligned} \Psi_{xy} &= \{\psi^k_{ST} | \psi^k_{ST} = r_S \bowtie r_T\} \\ Where\ S,T &\in \{x,y\},\ S \neq T \end{aligned} \tag{5.1}$$

During Y3 we enhanced and adapted the trust composition process according to the actual enhanced formalization of TMDL. We focused on a concrete implementation of trust tools to support given TMDL descriptions and to perform mediation. This required enhancement of the algorithms introduced last year to provide a concrete implementation of the composition process. In more detail, we identified and fully implemented the two following composition processes:

- The first process is to merge trust models. It aims at creating a new trust model that results from merging the two given ones. We call this process "Merging Composition". The composition is hence permanent and leads to extension of the behavior of the trust models' roles according to the given mapping rules.
- The second process is to create a cooperation that might happen for a limited period of time. The objective of this process is to not modify the behavior of existing roles, but create a set of $mediation\ roles$ capable of bridging the two given models. We call this process "Mediation Composition".

In order to define the composition process formally, in the table below, we illustrate all the notations that will be used:

Notation	Description
tor_l	Trustor role of the relation l
tee_l	Trustee role of the relation l
req_l	The trustor role of the access relation defined for operations of the relation l or $null$ if no access relation is defined
$\mathbb{D}_x$	The set of $Domains$ defined in the trust model x
$\mathbb{R}_x$	The set of $Roles$ defined in the trust model x
$\mathbb{M}_x$	The set of $Metrics$ defined in the trust model x
$\mathbb{C}_x$	The set of $Contexts$ defined in the trust model x
$\mathbb{L}_x$	The set of $reLations$ defined in the trust model x
$\mathbb{S}_x$	The set of $deciSions$ defined in the trust model x
$l_x \approx l_y$	The relation l_s is defined to be similar to l_r
$l_x \leftarrow l_y$	The relation l_x is derived from the relation l_y

Table 5.11: TMDL Formal Notations

5.3.1 Merging composition process

In order to merge two exiting trust models, we define two mapping operators, namely: the extension ($\lhd\!\!\vdash$) and the equivalence ($\equiv$) operators (i.e., $\bowtie \in \{\lhd\!\!\vdash, \equiv\}$). As consequence:

- The extension rule $r_S \lhd\!\!\vdash r_T$ leads to extend the inheritance attribute of the source role r_S with the target role r_T.
- The equivalence rule, $r_S \equiv r_T$ leads to define a new role (r_{new}) that inherits both the source r_S and the target r_T roles.
  Formally, $r_S \equiv r_T \models (r_{new} \lhd\!\!\vdash r_S) \wedge (r_{new} \lhd\!\!\vdash r_T)$

Thus, the merging composition, denoted by $\bigoplus_{\Psi_{xy}}$, of two trust models TM_x and TM_y, introduces a new trust model TM_z, as follows:

$$\begin{aligned} TM_z &= TM_x \bigoplus_{\Psi_{xy}} TM_y \\ &= < \mathbb{D}_x, \mathbb{R}_x, \mathbb{M}_x, \mathbb{C}_x, \mathbb{L}_x, \mathbb{S}_x > \bigoplus_{\Psi^{xy}} < \mathbb{D}_y, \mathbb{R}_y, \mathbb{M}_y, \mathbb{C}_y, \mathbb{L}_y, \mathbb{S}_y > \\ &= \left\langle \begin{array}{lcl} \mathbb{D}_z & = & \mathbb{D}_x \cup \{d_{xy}\} \\ \mathbb{R}_z & = & \mathbb{R}_x^+ \cup \mathbb{R}_y^+ \cup \mathbb{R}_{xy} \\ \mathbb{M}_z & = & \mathbb{M}_x \cup \mathbb{M}_y \\ \mathbb{C}_z & = & \mathbb{C}_x \cup \mathbb{C}_y \\ \mathbb{L}_z & = & \mathbb{L}_x \cup \mathbb{L}_y \\ \mathbb{S}_z & = & \mathbb{S}_x \cup \mathbb{S}_y \end{array} \right\rangle \end{aligned} \tag{5.2}$$

Where the $\mathbb{R}_x^+$ and $\mathbb{R}_y^+$ notations means that, in addition to the roles of each models, each set contains also the roles that are extended according to the given extension rules. Finally, the $\mathbb{R}_{xy}$ is the set of all new roles issued from an equivalence rule. These new roles are hence defined within a new domain d_{xy}.

5.3.2 Mediation composition process

Since the cooperation is temporarily, the mediation process preserves the existing trust models, and can be applied at runtime by deploying some recommenders (i.e, mediation roles) that are able to perform operations across models (i.e., mediation). This mediation process hence generates a new trust mediation model where the required mediation roles are defined.

In the context of cooperation, we express mapping rules with the play operator ($\triangleright$). The play rule $r_S \triangleright r_T$ basically defines which role form the target model, participants of the source model (identified by their role) are going to play when they visit the target model. Thus, in order to help roles of the target model to establish relationships with the source role as if it is a target role, for each mapping rule we define a new mediation role. This role is going to work as (i) a $recommender$ to enable roles of the target model that want to assess the source role to request the mediation that works also as (ii) a $requestor$ able to retrieve and update the trustworthiness of the source role from the target model requests. As illustrated in Figure 5.3, applying a play rule is performed in four steps:

1. *Build the target relation set* ($\mathbb{L}_t^{r_t}$): It represents all the relations (l_t) that assess the target role (r_t) and give a remote access to their get or set operations. In this case the mediator can inherit from the trustor role of that relation and hence be a recommender, Formally:

$$\mathbb{L}_t^{r_t} = \{l \in \mathbb{L}_t |\ (tee_l = r_t) \wedge (req_l \neq null)\} \tag{5.3}$$

2. *Build source relation set* ($\mathbb{L}_s^{r_s}$): It represents all the relations (l_s) that assess the source role (r_s) and provide a remote access to their operation. The mediator thus can be a requestor able to answer/propagate any request that come from the target models. To do so, the mediation role will have to inherit from the requestor role (req_{l_s}) of the relation. Formally:

$$\mathbb{L}_s^{r_s} = \{l \in \mathbb{L}_s |\ (tee_l = r_s) \wedge (req_l \neq null)\} \tag{5.4}$$

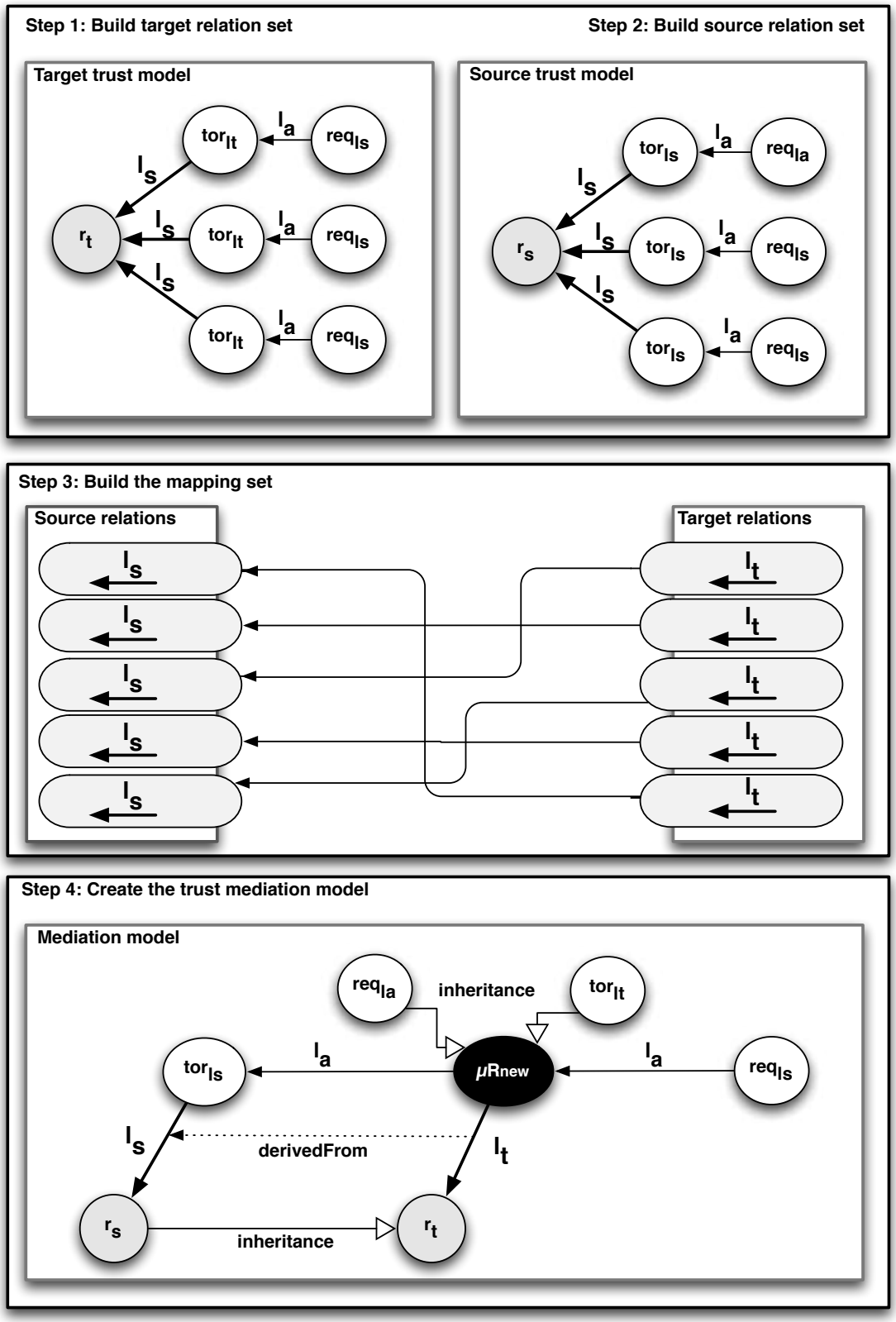


Figure 5.3: The Mediation Process

3. *Build the mapping set* $\mathbb{M}$*:* in order to allow the mediation role to be requested by the role of the target model and answer by requesting roles of the source model, the mediation process build the mapping set that contains pairs of similar relations, i.e., it has to find for each target relation a corresponding similar source relation. The similarity is mainly based on the trust context and have to be validated by the administrator. However, if relations are semantically annotated, similarity can be automatically states by interrogating their corresponding ontology. Formally:

$$\mathbb{M} = \{(l_t, l_s) \in \mathbb{L}_t^{r_t} * \mathbb{L}_s^{r_s} \mid l_t \approx l_s\} \tag{5.5}$$

4. *Create the trust mediation model:* Now that all inputs are found, we are able to create the trust mediation model. It defines the new mediator role μr_{new} that inherits both the trustor roles of the target relations (tor_{l_t}) and the requestor roles req_{l_s} of the source relations. All target relations that are managed by the mediation role have to be derived from source relations since the mediator will request the source relation to assess the target relation. Therefore, whenever the mediator is solicited for a target relation, it will perform the corresponding operation on the derived source relation. Formally:

$$r_S \rhd r_T \models \forall (l_t, l_s) \in \mathbb{M} | \ (\mu r_{new} \lhd\!\rhd req_{l_s}) \wedge (\mu r_{new} \lhd\!\rhd tor_{l_t}) \wedge (l_t \leftarrow l_s) \tag{5.6}$$

Summarizing, the mediated composition, denoted by $\bigotimes_{\Psi_{xy}}$, of two trust models TM_x and TM_y, which produces the trust mediation model μTM_{xy}, is defined as follows:

$$\begin{aligned}
\mu TM_{xy} &= TM_x \bigoplus_{\Psi_{xy}} TM_y \\
&= < \mathbb{D}_x, \mathbb{R}_x, \mathbb{M}_x, \mathbb{C}_x, \mathbb{L}_x, \mathbb{S}_x > \bigoplus_{\Psi^{xy}} < \mathbb{D}_y, \mathbb{R}_y, \mathbb{M}_y, \mathbb{C}_y, \mathbb{L}_y, \mathbb{S}_y > \\
&= \left\langle \begin{array}{lcl} \mathbb{D}_{xy} & = & \mathbb{D}_x^- \cup \mathbb{D}_y^- \cup \{d_m\} \\ \mathbb{R}_{xy} & = & \mathbb{R}_x^- \cup \mathbb{R}_y^- \cup \mu\mathbb{R} \\ \mathbb{M}_{xy} & = & \mathbb{M}_x^- \cup \mathbb{M}_y^- \\ \mathbb{C}_{xy} & = & \mathbb{C}_x^- \cup \mathbb{C}_y^- \\ \mathbb{L}_{xy} & = & \mathbb{L}_x^- \cup \mathbb{L}_y^- \\ \mathbb{S}_{xy} & = & \emptyset \end{array} \right\rangle
\end{aligned} \tag{5.7}$$

$\mathbb{D}_x^-, \mathbb{D}_y^-, \mathbb{R}_x^-, \mathbb{R}_y^-, \mathbb{M}_x^-, \mathbb{M}_y^-, \mathbb{C}_x^-, \mathbb{C}_y^-, \mathbb{L}_x^-, \mathbb{L}_y^-$ are sub sets of the original models' trust sets and contain only the trust attributes that lead to the creation of the mediation roles μr_{new}. d_m represents the mediation domain where all the mediation roles included in $\mathbb{R}$ are defined.

5.4 TMDL Tools

As illustrated in Figure 5.4, the TMDL language gives rise to two software tools. The former is a TMDL editor plugin (see Section 5.4.1) for Eclipse, generated with EMF (Eclipse Modeling Framework[2]) (Figure 5.4 white boxes). The latter is a Java application that we called $iMTrust$ (Figure 5.4 grey boxes). $iMTrust$ allows developers to (i) generate Java code of trust management systems from TMDL descriptions (see Section 5.4.2), (ii) emulate, test and deploy a network of participants that plays the roles described in the corresponding TMDL (see Section 5.4.3) and finally (iii) compose heterogeneous trust management systems (see Section 5.4.4).

5.4.1 TMDL editor plugin

As illustrated in Figure 5.4, we use the EMF modeling tools[3] to automatically generate the TMDL Core API that will serve to parse, create and validate TMDL descriptions annotated with $tmdl$ extension (step 1). We also use EMF tools to generate an Eclipse editor (step 2) that guides developers to create valid $tmdl$ files.

[2]EMF: www.eclipse.org/emf

[3]The EMF project is a modeling framework and code generation facility for building tools and other applications based on a structured XMI data model called "ecore model".

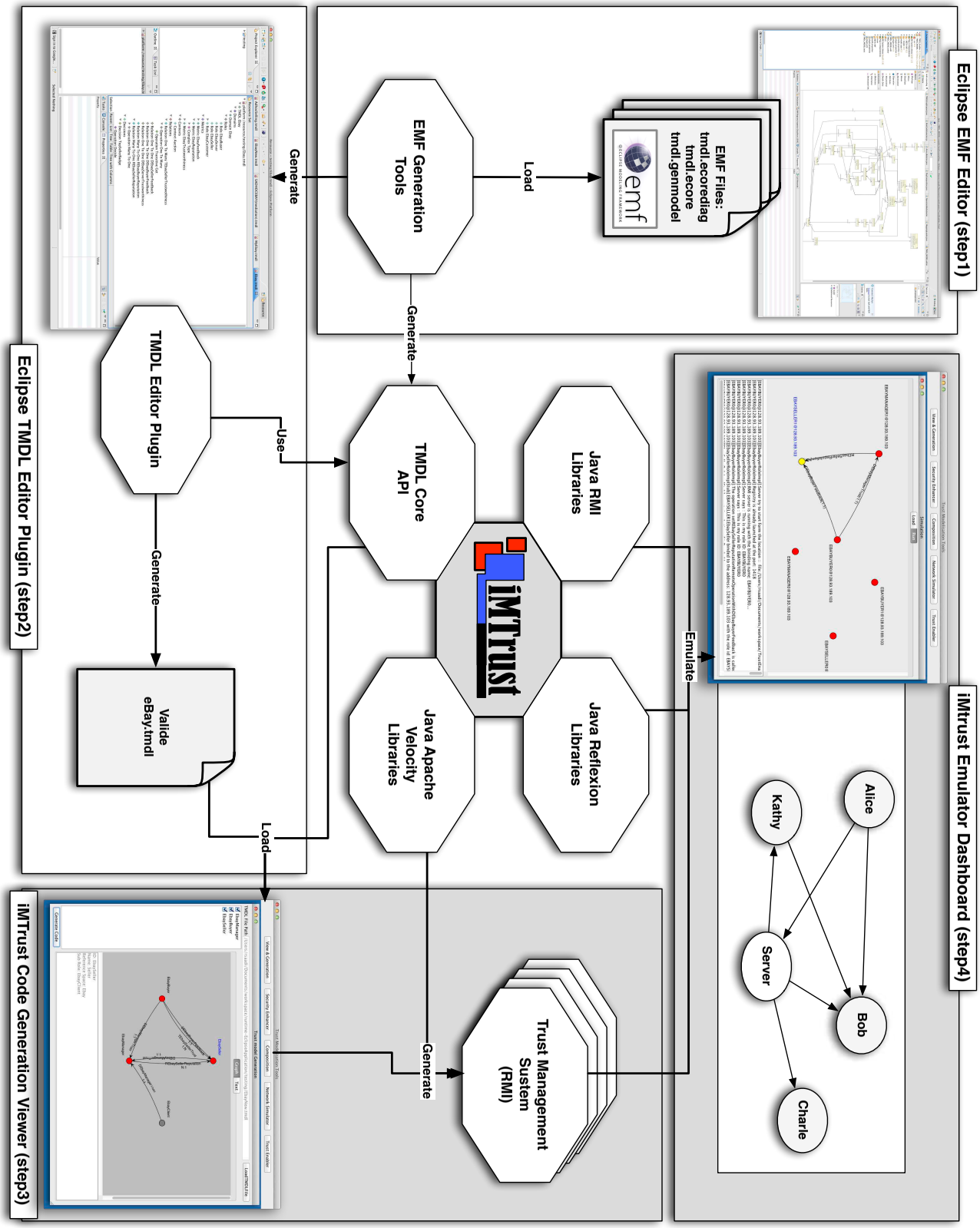


Figure 5.4: Code Generation Process

EMF File	Description
tmdl.ecorediag	UML description of the TMDL model
tmdl.ecore	XMI description of the TMDL model
tmdl.genmodel	XMI description to generate TMDL Java classes and plugins

Table 5.12: EMF TMDL Files

To do so, we create three $ecore$ files (see Table 5.12), namely: (i) $tmdl.ecorediag$ which represents a graphic UML representation of TMDL, (ii) $tmdl.ecore$ that is generated from the UML model, and finally (iii) $tmdl.genmodel$ that contains all the specifications that are required to generate the TMDL core API and the editor plugin, such as: TMDL definition URL and TMDL Java packages.

5.4.2 TMDL role code generation

In order to help developers to automatically generate their trust management system automatically we provide a tool that processes a given TMDL file and generates Java code for each non-abstract role all the operation that it is allowed to perform. Participants are hence ready to interact with each others by simply running the operations corresponding to the role(s) they play. For instance, parsing the GMES model (gmes.tmdl), the application is able to generate four pieces of code corresponding to each non-abstract role, namely, for Pilots, Chiefs, Drones and Servers. However, the generated code has to be slightly completed with the application logic to be fully functional.

The TMDL code generation tool creates for each role a separate project that is named with the corresponding role's name, we call this role the $main\ role$ of the project. All projects of the same model are generated into a workspace that can be easily imported by Eclipse.

We used Apache Velocity[4] for code generation. Velocity is a Java-based template engine that provides a template language that make reference to objects defined in Java code. Its aim is to ensure clean separation between the presentation parts and the application parts. In Table 5.13, we list the name of the velocity template that are defined to generated TMDL Java files.

Trust model package folder	Description	Velocity template
inter	roles definition interfaces	roleInter.vel
metric	metric implementation classes	metricImpl.vel
relation	relation definition and relation handler classes	relation.vel relationHandler.vel
role	main role implementation classes	roleImpl.vel roleImplBinding.vel
stub	trustee roles implementation classes	roleImplStub.vel
logic	relation implementation classes (application logic)	relationImpl.vel decisionImpl.vel

Table 5.13: Generated Model Structure

Each project is composed of six source folders (see Table 5.13) as follows:

- *The interface folder ($inter/$):* It contains the interface of the main role and the interfaces of all roles that it might interact with.
- *The metric folder ($metric/$):* It contains the classes that implement all the metrics that are manipulated by the main role.
- *The relation folder ($relation/$):* It contains the classes that define all relations that are managed or requested by the main role, and all the classes that handle each relation and implement all its related operations (i.e., operationGet and operationSet).

[4] Apache Velocity: http://velocity.apache.org/

- *The role folder ($role/$):* It represents the access root of the project, since its implements the classes of the main role and all its sup-roles.
- *The trustee folder ($stub/$):* It contains the classes that implement all the trustee roles that might interact with the main role.
- *The application logic folder ($logic/$):* It contains the classes that express the application logic. These classes implement all the methods that have to be filled by the developer to assess metrics and to provide corresponding decisions.

	RelationOneToOne	RelationManyToOne	RelationOneToMany
Bootstrapping	$Caller$: $operationSet$ $Input$: $null$ $Output$: $M(CR)$	$Caller$: $operationSet$ $Input$: $null$ $Output$: $M(CR)$	
Updating	$Caller$: $operationSet$ $Input$: $M(CR)/M(WR)$ $Output$: $M(CR)$		
Concatenation			$Caller$: $operationGet$ $Inputs$: $M(FR)$: $M(WR)$ $Output$: $M(CR)$
Aggregation		$Caller$: $operationSet$ $Input$: $M(WR)$ $Output$: $M(CR)$	$Caller$: $operationGet$ $Input$: $M(CR)[]$ $Output$: $M(CR)$

CR: Current Relation, WR: With Relation, FR: From Relation and M(R) the metric type of the relation R.

WR and FR represents the relation that are defined according to the caller operation.

Caller refers to the operation that calls the corresponding logic based-operation.

Input(s) and Output are the type of the Input(s) and the Output trust values, respectively.

Table 5.14: Logic-based Operations

As part of the application logic, developers will have to define how their trust management system bootstraps trust relationships, as well as updates, concatenates and aggregates trust values. As described in Table 5.14, these logic-based operations are related to the relation type and are called when an operationGet or an operationSet is triggered, as follows:

- *Bootstrapping:* Trust bootstrapping aims to initialize trust relationships in order to efficiently start the system and also allow newcomers to join the running system [63]. It is performed by trustors to initialize relationships that are managed by them, namely, RelationOneToOne and RelationManyToOne. To do so, every time that operationSet is triggered for an unknown trustee the bootstrapping method is called and hence a new relationship is established. This method cannot be generated and has to be completed by the developer. For instance, most existing solutions simply initialize trust relation with a fixed value (e.g., 0.5 [48], a uniform Beta probabilistic distribution [50], etc.).
- *Updating:* The updating method is related to relations of type OneOne and is called whenever the operationSet is called with input parameters to update the trust value of an existing OneToOne relationship, either with the trustor personal opinion (of type M(CR)) or with another relationship (of type M(WR)). This method is also application-specific and depends on how the trust management system updates direct relationships. For instance, updating existing trust relations according to peers recommendations [64]; assessing trustees in different contexts (e.g., fixing a car, babysitting, etc.) and then bootstrapping and updating unknown trust values from known ones of similar or correlated contexts [63, 7].
- *Concatenation:* It is related to relations of type OneToMany and is called when an operationGet is triggered. The concatenation operation enables to weight a given recommendation (i.e., withRelation) with it recommenders trustworthiness (i.e., fromRelation). The trust management system has to define how this method is implemented if it consider relations of type OneToMany. For instance, implementing the concatenation as a multiplication when the operation's operands (trust value) are of type float and are in the range of [0,1].
- *Aggregation:* The aggregation method is used to aggregate recommendations. It is hence related to relations of type ManyToOne and OneToMany. In the case of OneToOne relations, when an operationSet is triggered the aggregation operation is called to aggregate the operationSet input

(i.e., a recommendation given by the withRelation trust value) to the last cumulated reputation. This operation is application specific, for instance, it might based on Bayesian probability [60]. Whereas, for relationships of type OneToMany, the aggregation method is called to aggregate when multiple trustee recommendations (assessed with the concatenation operation) are retrieved. This operation can also be implemented by different ways (e.g., minimum, maximum, average, etc.)

5.4.3 Trust network emulation

We provide a tool that is able to compile a trust model workspace and to generate any given numbers of participants. The generated code used RMI as communication protocol, hence each emulated participant is identified by an $identifier@location$, where:

- the $identifier$ is made up of the name of the role that the participant plays and a random integer
- the $location$ is simply the IP address of the machine where the emulated participant is running (e.g., customer1@10.0.0.5).

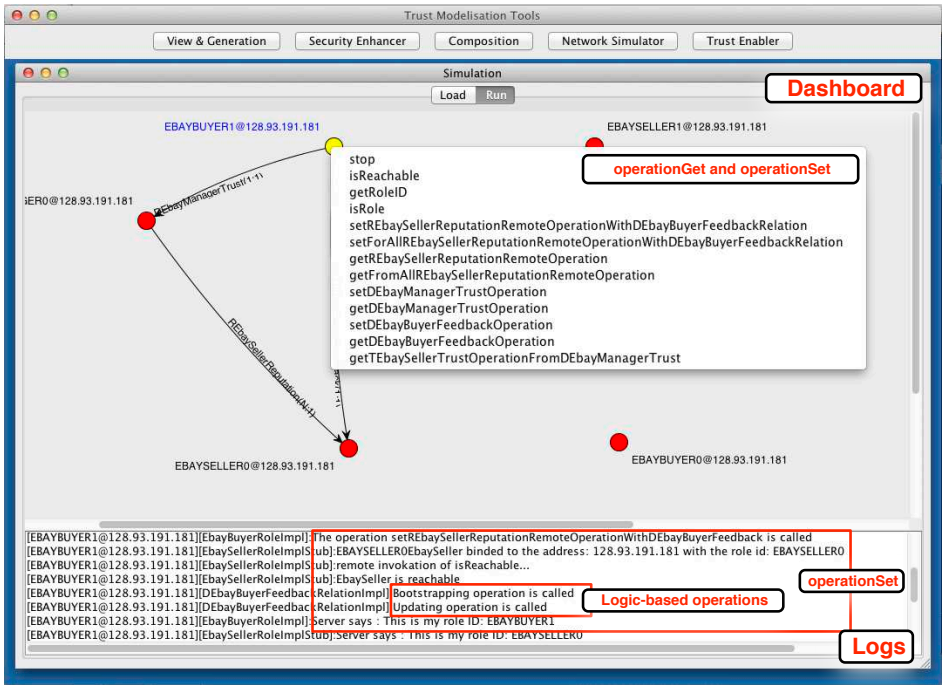


Figure 5.5: The Emulator Dashboard

As illustrated in Figure 5.5, the emulator displays a dashboard where all the emulated participants are represented by nodes. We use Java reflection to parse all the methods that are provided by each participant and also to execute them. Input parameters are annotated in the code (using a custom Java annotation), so that using reflection any inputs parameters is dynamically filtered and checked while input values are entered. When an operation is triggered, all generated relationships appear as edges that link the trustor of each relationship to its trustee.

This tool is very useful to test the generated trust management system. In fact, since the emulator uses Java reflection, developers can perform advance testing by enhancing the generated classes with additional methods that have to be rigorously annotated, to be able to launch them directly from the dashboard. Furthermore, testing resilience to security threats can be fulfilled by simulating malicious participants. This can simply expressed by deriving from the same role different categories of participants (i.e., malicious and safe) using the inheritance property. Then, reflecting these sup-role behaviors can be done by working on the model and/or after generating their corresponding code specifying their logic-based operations differently.

5.4.4 Composition tool

The composition tool provides an interface that loads and composes two trust models from their TMDL descriptions. It implements the processes detailed in Section 5.3, namely, the merging and the mediation processes.

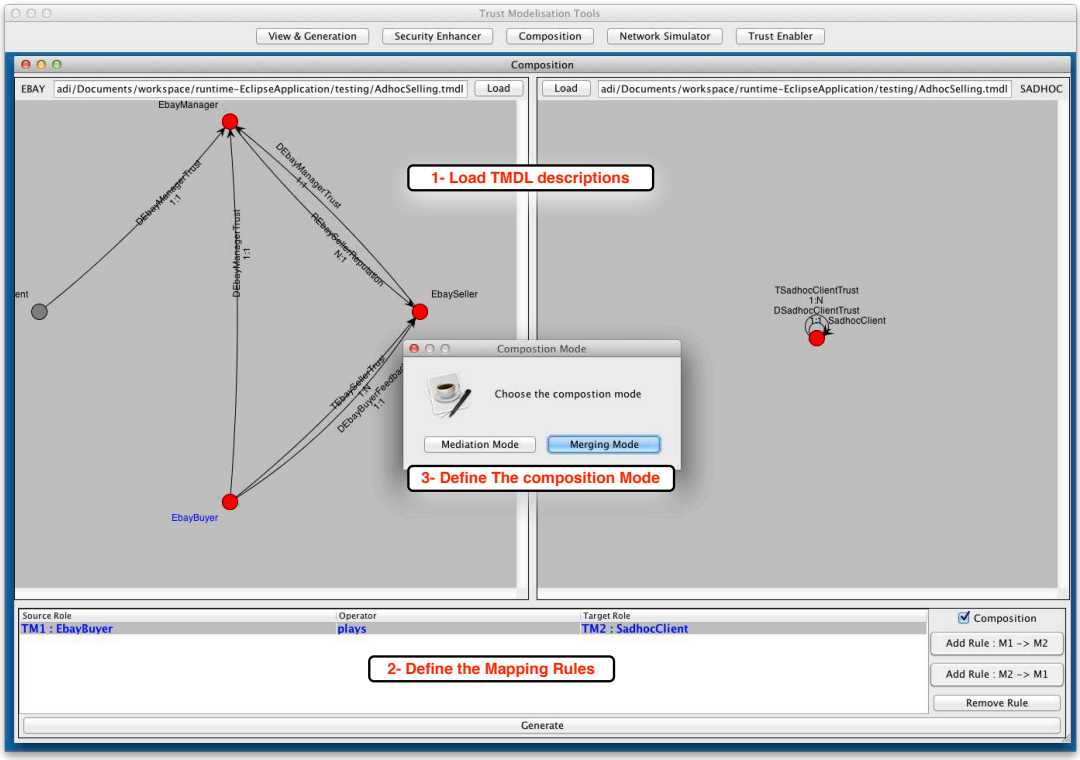


Figure 5.6: The Composition Tool Interface

As illustrated in Figure 5.6, the interface is divided in two parts, one to load the TMDL description, and the other one to specify the mapping rules. Actually, the mapping rules are defined manually, but we plan to enhance next version of the tool with an ontology checker. Therefore, mapping rules can be inferred automatically by processing semantic annotations of the TMDL attributes. Once the rules are specified, pushing the button generate (in the bottom of the interface) performs the related composition algorithm and produces a TMDL description of the new composed model. The result description can be processed by the Generation tool to generate the Java code of any participant of the corresponding trust management system.

5.5 Conclusions and Future Work

During Y3, we have enhanced the TMDL Language. We have also provided dedicated tools to help developers to specify trust models and generate trust management systems as well as support mediation cross heterogeneous trust domains.

As future work, we identify three objectives, namely, (i) implementing the trust Enabler with the aforementioned tools ; (ii) integrating TMDL with the CONNECT property meta-model and (iii) defining the interactions between the trust Enabler and other CONNECT Enablers explicitly.

5.6 Annex (GMES.tmdl)

```
<?xml version="1.0" encoding="UTF-8"?>
<tmdl:TMDL xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
                                        xmlns:tmdl="http://arles.inria.fr/tmdl_1.0">
  <domains>
    <domain name="GMES">
      <roles>
        <role ID="GMESFireman" name="Fireman" abstract="true"/>
        <role ID="GMESPilot" name="Pilot">
          <inheritance role="GMESFireman"/>
        </role>
        <role ID="GMESChief" name="Chief">
          <inheritance role="GMESFireman"/>
        </role>
        <role ID="GMESDrone" name="Drone">
          <inheritance role="GMESFireman"/>
        </role>
        <role ID="GMESServer" name="Server"/>
      </roles>
      <metrics>
        <metric ID="GMESFeedback" name="Feedback">
          <complextype>
            <sequence>
              <element name="IDTransaction" type="integer"/>
              <element name="ErrorLevel" type="integer">
                <restriction>
                  <enumeration value="0"/>
                  <enumeration value="1"/>
                  <enumeration value="2"/>
                </restriction>
              </element>
            </sequence>
          </complextype>
        </metric>
        <metric ID="GMESProbability" name="Probability">
          <complextype>
            <sequence>
              <element name="proba" type="decimal"/>
            </sequence>
          </complextype>
        </metric>
      </metrics>
      <contexts>
        <context ID="GMESFireFighting" name="FireFighting"/>
      </contexts>
      <relations>
        <relationOneToMany ID="TGMESPilotTrustworthiness" name="PilotTrustworthiness"
                trustorRole="GMESChief" trusteeRole="GMESPilot" withinContext="GMESFireFighting"
                metricType="GMESProbability">
          <operation>
            <operationGet fromRelation="DGMESServerTrustworthiness" withRelation="
                RGMESPilotReputation"/>
          </operation>
        </relationOneToMany>
        <relationOneToOne ID="DGMESServerTrustworthiness" name="ServerTrustworthiness"
                trustorRole="GMESFireman" trusteeRole="GMESServer" withinContext="
                    GMESFireFighting"
                metricType="GMESProbability"/>
        <relationOneToOne ID="DGMESDroneFeedback" name="DroneFeedback" trustorRole="GMESDrone"
                trusteeRole="GMESPilot" withinContext="GMESFireFighting" metricType="
                    GMESFeedback"/>
        <relationManyToOne ID="RGMESPilotReputation" name="PilotReputation" trustorRole="
            GMESServer"
                trusteeRole="GMESPilot" withinContext="GMESFireFighting" metricType="
                    GMESProbability">
          <operation>
            <operationGet accessFrom="DGMESServerTrustworthiness"/>
```

```
62                <operationSet accessFrom="DGMESServerTrustworthiness" withRelation="
                      DGMESDroneFeedback"/>
63              </operation>
64            </relationManyToOne>
65          </relations>
66        </domain>
67    </domains>
68 </tmdl:TMDL>
```

6 Conclusions and Future Work

In this deliverable we have described the results achieved in Y3 within WP5. This is a broad scope workpackage addressing CONNECTability, i.e., how to guarantee and assess the relevant non-functional properties of CONNECTed systems at synthesis time and at run-time. In particular, in this deliverable we have reported the advancements achieved on:

CPMM: this is the CONNECT meta-model for CONNECTability properties and their associated metrics. It had been already introduced in D5.2 [28]. In the third year we have revised its specification, specifically the definition of properties and of the events through which they are measured, and provided an implementation of the Model2Code transformation from CPMM to the Drools Fusion specification accepted in input by the GLIMPSE monitor.

DEPER: we have presented the latest implementation of the Dependability&Performance analysis Enabler, including some basic fault-tolerance mechanisms/patterns that can be provided to the Synthesis Enabler, as well as an approach based on Latin Hypercube Sampling (LHS) method to construct a new CONNECTor that differs from the previous one on the values of adjustable parameters. We have completed the interaction with the monitoring enabler for run-time adaptation. We have also extended incremental verification to use symbolic implementations, proposing a novel hybrid adaption of the Tarjan algorithm that combines symbolic and explicit-state data structures. This approach alleviates the limit of the size of models that have explicit state data structures (which are used to store the state space and transition relation) can handle.

SxCxT: in the second year we had extended the security-by-contract paradigm into the SxCxT methodology, that incorporates trust considerations and takes into account CONNECT threats. In the reporting period we have completed the implementation of the relative Security Enabler, which performs code instrumentation of the CONNECTor in transparent way. Moreover, the Security Enabler is also now interacting with the Monitoring Enabler in order to deal with security violations at run-time.

Trust: the formal definition of TMDL (Trust Model Description Language) language has been completed and a corresponding enabler supporting the composition of different trust management systems has been developed. We have also released a TMDL editor.

6.1 Prototype implementation

The above summarised scientific advances have all been instantiated in several supporting tools. The prototype software is released as an integral part of this report. Thus, in the associated Appendix-Prototypes document D5.3P, we provide the list of released tools and enablers, along with essential information and the respective URLs from which they can be downloaded. 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 iMTrust set of tools, including the associated iTMDL Editor;
- the GLIMPSE run-time monitoring infrastructure, which can interoperate with DePer and SxCxT.

6.2 Future Work

In the next period, we will complete the refinement and experimentation with the CONNECTability enablers. Concerning the specific plans for improvement and completion of each enabler individually, they are already reported in closure of each relative chapter and we will not repeat them here. More importantly,

with the various enablers now in quite mature status, in the remaining months before project completion the activity of WP5 will be actively focussed on further experimenting their application on the CONNECT scenarios under refinement in WP6 [32]. A challenge will consist into setting up a whole integrated scenario for CONNECTability demonstration, and also into further pushing their integration and interaction with the other CONNECT Enablers, notably Synthesis and Learning, for which we already have achieved some promising results [39, 17]. In particular, we have already started to develop a joint broader framework integrating Synthesis, Learning, DEPER and GLIMPSE for supporting a full cycle of runtime adaptation triggered by observation of non-functional behaviour. This will entail a closed loop between DEPER and Synthesis for either adapting a CONNECTor or for starting a new synthesis, as we preliminary describe in [15].

Bibliography

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

[2] Esper: Event stream and complex event processing for java. http://www.espertech.com/products/esper.php.

[3] Java Enterprise System Monitoring Framework. http://download.oracle.com/docs/cd/E19462-01/819-4669/geleg/index.html.

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

[5] Acceleo. http://www.eclipse.org/acceleo/.

[6] B. Adams, W. Bohnhoff, K. Dalbey, J. Eddy, M. Eldred, D. Gay, K. Haskell, P. Hough, and L. Swiler. Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 5.0 user's manual. Technical Report SAND2010-2183, December 2009. Updated December 2010 (Version 5.1).

[7] 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.

[8] Akamai Technologies, Inc. Akamai sureroute for failover and performance, 2003.

[9] D. G. Andersen. Mayday: Distributed Filtering for Internet Services. In *4th Usenix Symposium on Internet Technologies and Systems*, Seattle, WA, March 2003.

[10] V. Barnett. Probability plotting methods and order statistics. *Journal of the Royal Statistical Society*, Series C (Applied Statistics) Vol. 24(1):95–108, 1975.

[11] W. Barth. *Nagios. System and Network Monitoring*. No Starch Press, u.s. ed edition, 2006.

[12] E. Bartocci, R. Grosu, P. Katsaros, C. Ramakrishnan, and S. Smolka. Model repair for probabilistic systems. In *Proc. of 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'11)*, volume 6605 of *LNCS*. Springer, 2011.

[13] A. Bennaceur, F. Chauvel, P. Inverardi, V. Issarny, I. Matteucci, F. Martinelli, R. Spalazzese, and M. Tivoli. Reasoning about and Harmonizing the Interaction Behavior of Networked Systems at Application- and Middleware- Layer. Rapport de recherche, Feb. 2011.

[14] A. Bertolino, A. Calabrò, F. Di Giandomenico, and N. Nostro. Dependability and Performance Assessment of Dynamic CONNECTed Systems. In *11th International School SFM 2011*, volume LNCS 6659, pages pp. 350–392, Bertinoro, Italy, 2011. Springer.

[15] A. Bertolino, A. Calabrò, F. Di Giandomenico, N. Nostro, P. Inverardi, and R. Spalazzese. On-the-fly Dependable Mediation between Heterogeneous Networked Systems, 2012. submitted for publication.

[16] A. Bertolino, A. Calabrò, F. Lonetti, A. Di Marco, and A. Sabetta. Towards a model-driven infrastructure for runtime monitoring. In Troubitsyna [71], pages 130–144.

[17] A. Bertolino, A. Calabrò, M. Merten, and B. Steffen. Never-stop learning: Continuous validation of learned models for evolving systems through monitoring. *ERCIM News*, 2012(88), 2012.

[18] 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.

[19] R. Bloem, H. N. Gabow, and F. Somenzi. An algorithm for strongly connected component analysis in log symbolic steps. In *Proc. FMCAD'00*, volume 1954 of *LNCS*, pages 37–54. Springer, 2000.

[20] R. T. Braden. RFC 1122: Requirements for Internet hosts — communication layers, Oct. 1989.

[21] R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[22] J. Burch, E. Clarke, K. McMillan, D. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proc. LICS'90*, pages 428–439. IEEE Computer Society Press, 1990.

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

[24] S. Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. *Data & Knowledge Engineering*, 14(1):1–26, 1994.

[25] 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.

[26] CONNECT Consortium. Deliverable 5.1 – conceptual models for assessment & assurance of dependability, security and privacy in the eternal CONNECTed world, 2010.

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

[28] CONNECT Consortium. Deliverable 5.2 – Design of Approaches for Dependability and Initial Prototypes, 2011.

[29] CONNECT Consortium. Deliverable 6.2 – Experiment scenarios, prototypes and report Iteration 1, 2011.

[30] CONNECT Consortium. Deliverable 1.3 – Revised CONNECT Architecture, 2012.

[31] CONNECT Consortium. Deliverable 3.3 – Modeling of Application- and Middleware-layer Interaction Protocols, 2012.

[32] CONNECT Consortium. Deliverable 6.3 – Experiment scenarios, prototypes and report Iteration 2, 2012.

[33] G. Costa, P. Degano, and F. Martinelli. Modular Plans for Secure Service Composition. In *Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security*, pages 41–58, Paphos, Chypre, 2011.

[34] 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. ISSN (print): 2093-5374, ISSN (on-line): 2093-5382.

[35] G. Costa, F. Martinelli, V. Issarny, R. Saadi, and I. Matteucci. Security and Trust. In *SFM'11 - 11 th International School on Formal Methods for the Design of Computer, Communication and Software*, pages 393–416, Bertinoro, Italy, 2011.

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

[37] 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.

[38] D. Daly, D. D. Deavours, J. M. Doyle, P. G. Webster, and W. H. Sanders. Möbius: An extensible tool for performance and dependability modeling. In B. R. Haverkort, H. C. Bohnenkamp, and C. U. Smith, editors, *11th Int. Conf., TOOLS 2000*, volume 1786 of *LNCS*, pages 332–336. Springer Verlag, 2000.

[39] F. Di Giandomenico, A. Bertolino, A. Calabrò, and N. Nostro. An approach to adaptive dependability assessment in dynamic and evolving CONNECTed systems. *International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS)*, 2012. to appear.

[40] A. Di Marco, C. Pompilio, A. Bertolino, A. Calabrò, F. Lonetti, and A. Sabetta. Yet another metamodel to specify non-functional properties. In D. Bianculli, S. Guinea, A. Metzger, and A. Polini, editors, *QASBA*, ACM International Conference Proceeding Series, pages 9–16. ACM, 2011.

[41] 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.

[42] R. Gentilini, C. Piazza, and A. Policriti. Computing strongly connected components in a linear number of symbolic steps. In *Proc. SODA'03*, pages 573–582, 2003.

[43] C. Gonzalez-Perez and B. Henderson-Sellers. A powertype-based metamodelling framework. *Software and Systems Modeling*, 5:72–90, 2006.

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

[45] E. M. Hahn, T. Han, and L. Zhang. Synthesis for pctl in parametric markov decision processes. In *Proc. of 3rd NASA Formal Methods Symposium*, volume 6617 of *LNCS*. Springer, 2011.

[46] E. M. Hahn, H. Hermanns, and L. Zhang. Probabilistic reachability for parametric markov models. In *Proc. of 16th International SPIN Workshop*, volume 5578 of *LNCS*. Springer, 2009.

[47] T. Han. *Diagnosis, Synthesis and Analysis of Probabilistic Models*. PhD thesis, University of Twente and RWTH Aachen University, 2009.

[48] 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.

[49] 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.

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

[51] A. D. Keromytis, V. Misra, and D. Rubenstein. Sos: Secure overlay services. In *In Proceedings of ACM SIGCOMM*, pages 61–72, 2002.

[52] M. Kwiatkowska, D. Parker, and H. Qu. Incremental quantitative verification for Markov decision processes. In *Proc. IEEE/IFIP International Conference on Dependable Systems and Networks (DSN-PDS'11)*, pages 359–370. IEEE CS Press, 2011.

[53] 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.

[54] M. Mansouri-Samani and M. Sloman. GEM: a generalized event monitoring language for distributed systems. *Distributed Systems Engineering*, 4(2):96–108, 1997.

[55] 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.

[56] P. Masci, N. Nostro, and F. D. Giandomenico. On enabling dependability assurance in heterogeneous networks through automated model-based analysis. In Troubitsyna [71], pages 78–92.

[57] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.

[58] M. McKay, R. Beckman, and W. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):Technometrics, 1979.

[59] N. Metropolis and S. Ulam. The monte carlo method. *Journal of the American Statistical Association*, 44:335–341, 1949.

[60] 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.

[61] E. Nuutila and E. Soisalon-soininen. On finding the strongly connected components in a directed graph. *Information Processing Letters*, 49:9–14, 1994.

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

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

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

[65] S. Röttger and S. Zschaler. Tool Support for Refinement of Non-functional Specifications. *Software and Systems Modeling*, 6(2):185–204, 2007.

[66] R. Saadi, M. Rahaman, V. Issarny, and A. Toninelli. Composing trust models towards interoperable trust management. *Trust Management V*, pages 51–66, 2011.

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

[68] L. Subramanian, I. Stoica, H. Balakrishnan, and R. H. Katz. Overqos: offering internet qos using overlays. *SIGCOMM Comput. Commun. Rev.*, 33:11–16, January 2003.

[69] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.

[70] K. S. Trivedi. *Probability and Statistics with Reliability, Queueing and Computer Science Applications*. John Wiley & Sons, New York, second edition, 2002.

[71] E. Troubitsyna, editor. *Software Engineering for Resilient Systems - Third International Workshop, SERENE 2011, Geneva, Switzerland, September 29-30, 2011. Proceedings*, volume 6968 of *Lecture Notes in Computer Science*. Springer, 2011.

[72] W. Zeng, Y. Zhu, H. Lu, and X. Zhuang. Path-diversity P2P overlay retransmission for reliable IP-multicast. *Multimedia, IEEE Transactions on*, 11(5):960–971, aug. 2009.

[73] Z. Zhou, Z. Peng, J.-H. Cui, and Z. Shi. Efficient multipath communication for time-critical applications in underwater acoustic sensor networks. *IEEE/ACM Trans. Netw.*, 19:28–41, February 2011.