An approach to adaptive dependability assessment in dynamic and evolving connected systems

Antonia Bertolino, Antonello Calabrò, Felicita Di Giandomenico, Nicola Nostro

An approach to adaptive dependability assessment in dynamic and evolving connected systems

Antonia Bertolino, Antonello Calabrò, Felicita Di Giandomenico, Nicola Nostro

Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo"
Consiglio Nazionale delle Ricerche
via Moruzzi 1, 56100 Pisa, Italy
(antonia.bertolino, antonello.calabro, nicola.nostro, digiandomenico)@isti.cnr.it

Abstract

Complexity, heterogeneity, interdependencies and, especially, evolution of system/services specifications, related operating environments and user needs, are more and more highly relevant characteristics of modern and future software applications. Taking advantage of the experience gained in the context of the European project CONNECT, which addresses the challenging and ambitious topic of eternally functioning distributed and heterogeneous systems, in this paper we present a framework to analyse and assess dependability and performance properties in dynamic and evolving contexts. The goal is to develop an adaptive approach by coupling stochastic model-based analysis, performed at design time to support the definition and implementation of software products complying with their stated dependability and performance requirements, with run-time monitoring to re-calibrate and enhance the dependability and performance prediction along evolution. The proposed framework for adaptive assessment is described and illustrated through a case study. To simplify the description while making more concrete the approach under study, we adopted the setting and terminology of the CONNECT project.

Key words: Adaptation, Dependability, Evolving Heterogenous Systems, Model-based Assessment, Monitoring, Performance

1. Introduction

Modern software applications are increasingly pervasive, dynamic and heterogeneous. More and more they are conceived as dynamically adaptable and evolvable sets of components that must be able to modify their behaviour at run-time

to tackle the continuous changes happening in the unpredictable *open-world* settings [3]. Operating in the open-world poses a number unprecedented challenges to software systems, including:

- The reference specification of expected/correct operation is not a-priori available;
- Specifications are learnt/inferred, thus they can be incomplete, unstable, uncertain, with impact on all the software engineering processes built upon system specification;
- System components are assembled dynamically, with potential strong impact on interoperability in presence of heterogeneity;
- Assessment activities must accommodate change (and must be adaptable themselves), therefore special emphasis is on run-time assessment (possibly coupled with off-line analysis techniques, wherever possible), which is a new paradigm with respect to traditional assessment methods.

As a result of such prominent trends two related needs emerge.

On the one side, we observe that the interconnected components, which we refer to as the Networked Systems (NSs), are independently developed. The fast pace at which technology advances along diverging tracks can form gaps and establish separately evolving technological islands, between which communication is hampered. Thus the state of practice is that ad hoc bridging solutions need to be continuously developed to fill those communication gaps.

On the other side, the everyday life of modern and future society is growingly depending on the services provided by such highly complex and pervasive systems. In some cases their failures might even lead to catastrophic consequences in terms of damages to human life, environment, economy. Therefore, increasing importance is given to dependability and performance properties of such systems.

The European FP7 Future and Emerging Technology Project CONNECT addresses both needs, aiming at enabling seamless and dependable interoperability among NSs in spite of technology diversity and evolution. The ambitious goal of the project is to have eternally functioning distributed systems within a dynamically evolving open-world context. This is pursued through the on-the-fly synthesis of the CONNECTors through which heterogeneous NSs can communicate in dependable and secure way. Indeed, effective interoperability requires

to ensure that such on-the-fly CONNECTed systems provide the required non-functional properties and continue to do so even in presence of evolution, thus calling for enhanced and adaptive assessment frameworks.

In the context of the CONNECT project, approaches to both off-line and run-time analysis are under development to analyze and ensure the synthesis of CONNECTors with required dependability and performance levels. In particular, an assessment framework is proposed which combines continuous on-line assessment of non-functional properties through a lightweight flexible monitoring infrastructure with stochastic model-based analysis. The goal is to assess complex dependability and performance metrics through accurate analysis that adapts to the evolving context. Although not novel in its basic principles, this off-line and run-time integrated framework is proposed as a general, automated approach to fulfill the dependability and performance assessment needs in dynamic and evolving contexts.

In this paper, we initially point out the challenges of assessing non functional properties in dynamic CONNECTed systems and provide the context for our research objectives (Section 2). Then we introduce first separately the pre-deployment analysis method (Section 3) and the run-time monitor (Section 4) under development and hence their synergic usage (Section 5), through which adaptive assessment is pursued. A case study is also included (Section 6) to demonstrate the applicability of the integrated analysis framework. Finally we overview related work (Section 7) and draw conclusions (Section 8).

2. Context

Before introducing our approach, in this section we set the reference context within which we settled our study on dependability and performance assessment methodologies able to account and adapt to system and environment changes. In the following two sub-sections we first provide a brief overview of the already mentioned CONNECT project, tailored to investigate research on eternally connected systems despite heterogeneity and dynamic evolution, and then discuss some emerging issues when addressing the assessment of systems in such context.

2.1. *Overview of the EU* CONNECT *project*

Our research is carried out in the context of the FP7 "ICT forever yours" European Project CONNECT[1], belonging to the Future and Emerging Technologies track. CONNECT collects a consortium of partners whose expertise covers middleware, software engineering, formal methods, machine learning, software synthesis and systems dependability. The CONNECT world envisions dynamic environments populated by technologically heterogeneous Networked Systems (NSs), and by the components of the CONNECT enabling architecture, called the CONNECT enablers.

The ambition of the project is to have eternally functioning systems within a dynamically evolving context. To overcome interaction protocol heterogeneity at all layers the project introduces a revolutionary approach that dynamically generates the inter-mediator components to connect heterogeneous systems. This is achieved by synthesizing *on-the-fly* the CONNECTors through which the NSs communicate. The resulting emergent CONNECTors then compose and further adapt the interaction protocols run by the CONNECTed System. In brief, the NSs manifest the intention to connect to other NSs. The enablers are networked entities that incorporate all the intelligence and logic offered by CONNECT for enabling the required connection. The emergent CONNECTors produced by the action of enablers are called the CONNECTors, whereas as an outcome of the successful creation and deployment of CONNECTors we obtain the CONNECTed systems.

In Figure 1 we provide an overview of the CONNECT vision and architecture. We show in schematic form the enablers which are currently part of the CONNECT enabling architecture. From top to bottom, we see:

Discovery Enabler catches the requests for communication coming from the NSs and initiates the CONNECT process. We tend to make the minimum possible assumption on the information (called affordance) that NSs must provide;

Learning Enabler : we use active learning algorithms to dynamically determine the interaction behaviour of a NS and produces a model in the form of a labeled transition system (LTS);

Synthesis Enabler : from the models of the two NSs, this enabler synthesizes a mediator component through automated behavioural matching;

Deployment Enabler finally deploys and manages the CONNECTors.

[1] http://connect-forever.eu

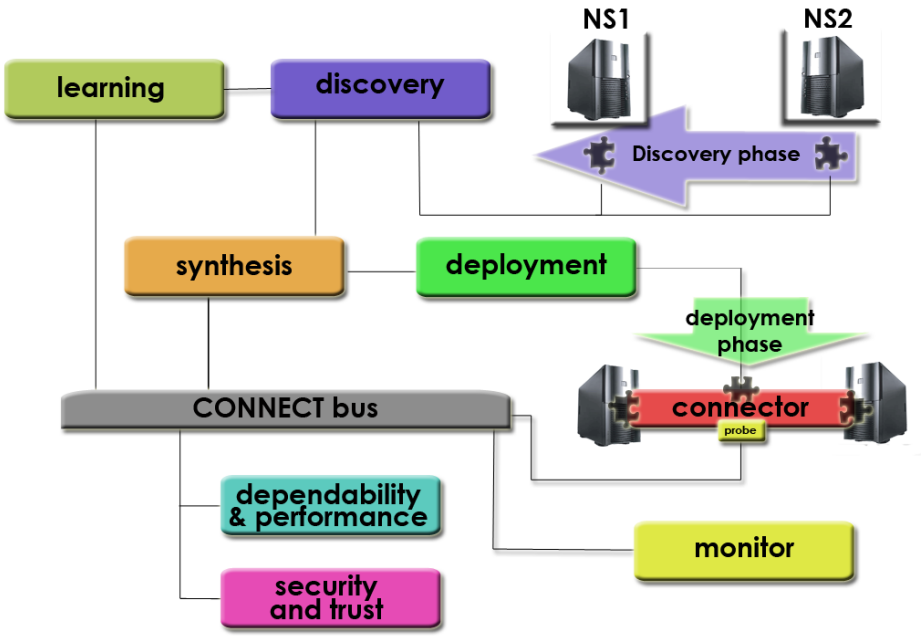

Figure 1: The CONNECT Architecture

Evidently, such a dynamic context strongly relies on one side on mechanisms for ensuring dependability, security and trust, and on the other side on functional and non-functional behaviour monitoring, through which run-time adaptation of CONNECTors is triggered. Hence the CONNECT architecture also includes the following important enablers:

Monitoring Enabler collects raw information about the CONNECTors behaviour and passes them to the enablers (the monitor's customers) who requested them; the CONNECT monitoring infrastructure is further described in Section 4;

CONNECT bus : all communication among the enablers and with the CONNECTors happens through a message bus, which is currently implemented by a simple message-based communication model as for instance the Java Messaging Service (JMS);

DEPER Enabler : this is the main focus of this paper and is described in detail in the next section;

Security and Trust Enabler collaborates with the synthesis enabler to satisfy possible security and trust requirements. It also continuously determines if the requirements are maintained at run-time, by receiving monitoring data from the monitoring enabler (similarly to the integrated approach we exemplify for dependability and performance in Section 5).

2.2. Challenges in dependability and performance assessment in evolving context

The need for research advancement in the assessment of evolving, ubiquitous systems is recognized by the dependability/resilience community, being indicated as one of the prominent research challenges in the research agenda set up by the ReSIST European Network of Excellence [9]. In fact, it is observed that, since current and future systems result from evolutions of pre-existing systems, as a consequence assessment should move from off-line and pre-deployment, to continuous and automated operational assessment. The traditional approaches to assessment, which dominate the current assessment practices, are: i) pre-deployment assessment, i.e. collecting data in a simulated environment (e.g. "model-based analysis", "statistical testing", "dependability benchmarking", etc.), and/or ii) processing the measurement data accumulated in real operation at a later stage, e.g. periodic reviews widely used in some safety-critical industries such as the nuclear sector. Both these categories of methods have shortcomings when dealing with evolution and dynamicity of the system under analysis. In fact, dealing with evolution and dynamicity raises two major challenges from the point of view of dependability and performance analysis:

- Pre-deployment assessment is limited by its nature: the impact on system dependability/performance cannot be known for unforeseen environments. Therefore, given the many possible variations occurring during software application lifetime, it would be necessary to analyze beforehand, through off-line analysis, all the possible scenarios which could take place at run-time, to be stored in a look-up table from which to retrieve the correct analysis upon scenario's occurrence. But this cumbersome activity is in general impossible to conduct at a sufficiently satisfactory level, especially for critical applications subject to strong dependability requirements. Resorting to processing the measurements collected in real operation at a later stage, e.g. in periodic reviews, may be inadequate as well, since by the time the observations are processed the operational environment may have changed to something not yet seen before.

- Pre-deployment assessment, however, plays an important role in providing a priori knowledge about how the system is expected to operate, especially if the simulated environment is "close" to the operational environment post-deployment, and to take appropriate design decisions. Stochastic model-based assessment has been widely recognized as a helpful means to cover this role [5]. Nevertheless, the unavoidable higher chance of inaccurate/unknown model parameters needs to be considered as a weakness that could result in too inaccurate analysis results, thus negatively impacting design decisions.

To contribute to overcome such deficiencies of current methods in assessing dynamic systems, we developed an approach which tries to combine the benefits of both pre-deployment and processing of data obtained from real executions, as illustrated in the following.

3. Pre-deployment stochastic model based analysis: the way to start supporting design decisions

As already mentioned, pre-deployment assessment is a crucial activity to drive the system design towards a realization compliant with the required level for quality of service indicators. In fact, it allows for early detection of design deficiencies, so as to promptly take the appropriate recovery actions, thus significantly saving in money and time with respect to discovering such problems at later stages. Also, it is central to the decision making process among alternative design solutions, again gaining in efficiency and better guarantee to end up with the "right" system. Stochastic model-based approaches are very suited and widely adopted for early prediction of dependability and performance metrics. Research has developed a variety of models, each one focusing on particular levels of abstraction and/or system characteristics, including State-Based Stochastic methods ([20]). These last use state-space mathematical models, expressed with probabilistic assumptions about time durations and transition behaviours; a short survey on State-Based Stochastic methods and automated supporting tools for the assisted construction and solution of dependability models can be found in [5].

The pre-deployment assessment part of our proposed method, tailored to dynamic and evolving systems assessment, exploits State-Based Stochastic modeling and analysis, which are embedded in an automated process. In the following, we overview the architecture and main functionalities of the Dependability and Performance enabler, introduced in Figure 1 and shortly referred to as DEPER.

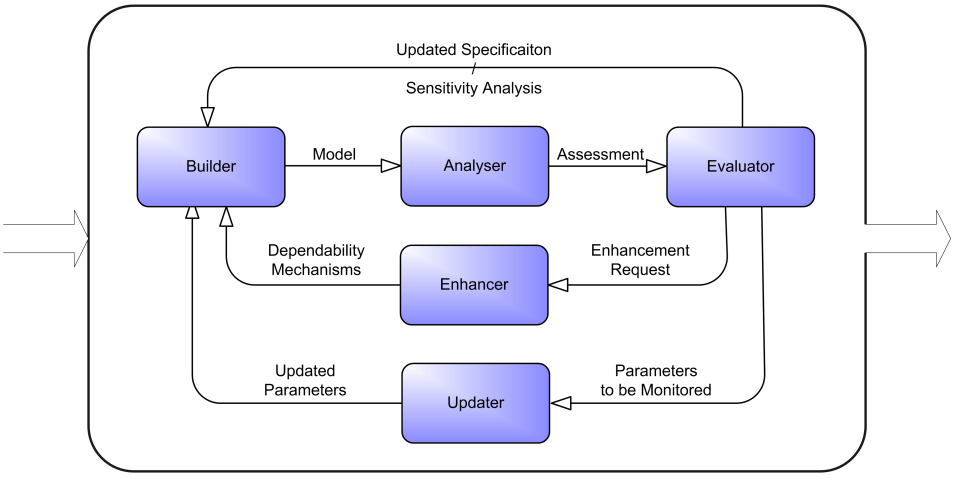

Figure 2: Architecture of the Dependability&Performance (DEPER) Analysis in CONNECT

DEPER supports the automated dependability and performance analysis and already partially described in [17, 4]. Figure 2 illustrates the five main modules composing DEPER, whose activities start from pre-deployment assessment of the generated bridging CONNECTors to subsequent refinements based on run-time observations of real networked systems and CONNECTors executions. A brief summary is provided for each module except for the *Udapter* module, which is fully described in Section 5, when focusing on the integration with monitoring. As already introduced in section 2.1, the DEPER enabler interacts with other enablers in the CONNECT framework to: i) be triggered on the analysis to perform and take in input both the specification of the system to analyze and the metric to assess together with the value required for it. The enablers contributing to this step are Discovery, Learning, Deployment and Synthesis; ii) provide the feedback from the analysis to Synthesis, which can then proceed with the deployment of the CONNECTor or refine it according to the feedback.

Builder The Builder module takes in input the specification of the CONNECTed system. This specification is given as Labelled Transition Systems (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 NSs 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. To allow for automated assessment, they are expressed in terms of transitions and states of the LTS specification of the NSs. 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 is in charge of checking whether the analysis results match with the guarantee, as requested by the networking systems willing to communicate, or not. Evaluator informs Synthesis about the outcome of the check and, in case of mismatch it may receive back a request to evaluate if enhancements can be applied to improve the dependability or performance level of the CONNECTed system, namely:

a) To take into account an alternative CONNECTor deployment (e.g., a deployment that uses a communication channel with lower failure rate). A new analysis is triggered, considering the updated specification of the CONNECTor.

b) 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). Such mechanisms then applied by the Enhancer module to model elements that are considered weak from the point of view of the metric under assessment.

Instead, in case the analysis results provided by Analyser match with the guarantee, the CONNECTor's design is considered satisfactory and ready to be deployed, thus terminating the pre-deployment analysis phase. However, because

of possible inaccuracy of model parameters due to potential sources of uncertainty dictated by the dynamic and evolving context, Evaluator also instructs the Updater module about its interaction with the Monitor enabler about on-line observation of events. Collection of such events allows to determine whether a new analysis needs to be performed, to properly adapt to changes (or unforeseen circumstances), as better detailed later in Section 5, when focusing on integration between model-based analysis and on-line monitoring.

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. 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, among those available in the category indicated by Evaluator; (ii) instructs the Builder module on the application of the selected dependability mechanism in the CONNECTed system model, in accordance with indications from Evaluator, and triggers a new analysis. 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's support to the design of this CONNECTor is completed. Otherwise, Enhancer makes a further attempt with the next dependability mechanism (if available), according to some internal predefined policies about the rank of available mechanisms and about how to apply them to model elements provided by Evaluator, and a new cycle with Builder, Analyzer and Evaluator is repeated. This loop ends either when a successful mechanism is found, or when all the mechanisms are exhausted. A library of models for triggering the generation of typical dependability mechanisms suitable to contrast two typical classes of failure modes that may happen during interactions has been defined and implemented (see [18]). Given the focus of this paper on the adaptation of model-based analysis through on-line observations, these mechanisms and related models will not be further treated.

4. The on-line view: incremental accumulation of observations through monitoring

The best way to provide a valid bridge between the observed system and the pre-deployment analyser is to insert a new layer, the monitoring layer, able to

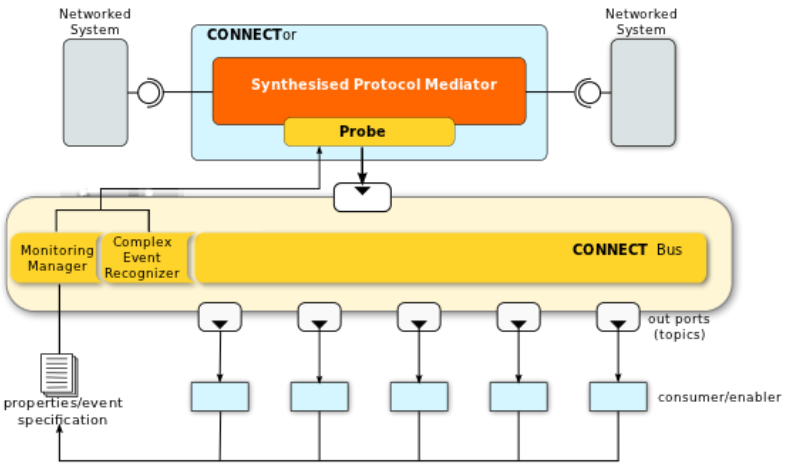

Figure 3: GLIMPSE architecture

gather and filter information useful to the dependability and performance analyses. Indeed, monitoring has been used for on-line dependability analysis since the advent of debuggers in the sixties.

In CONNECT we have developed a modular, flexible and lightweight monitoring infrastructure, called GLIMPSE[2]. Although expressly conceived for use in CONNECT, GLIMPSE infrastructure, shown in Figure 3, is totally generic and can be easily applied to different contexts. To provide a better communication decoupling, we adopted a publish-subscribe communication paradigm.

The lowest level of the monitoring is represented by the probe deployed into the CONNECTor; this probe monitors the messages exchanged among the NSs involved into the communication, possibly applying a local filter in order to decrease the amount of messages sent on the CONNECT bus. Note that such probes are non intrusive data collectors (proxies), i.e., they have no effect on the order and timing of events in the application and do not generate overhead on the communication or on the interacting services.

The second layer of the monitoring infrastructure is represented by the information consumers, the entities interested to obtain evaluation of a non-functional properties or interested to receive notification of occurrences of events/exceptions that may occurs into the CONNECTor.

[2]Glimpse is an acronym for Generic fLexIble Monitoring based on a Publish-Subscribe infrastructurE

With specific reference to this paper purposes, the Manager module is in charge to manage all the communication between the DEPER enabler (yet another consumer in the Monitor vision) and the Complex Event Processor (CEP). ✿[3] It analyzes the request message sent from DEPER and instruments the CEP. The message sent from DEPER enabler contains one or more rules related to nf-properties requirements that the monitor enabler must verify. This message is structured following a generic XSD schema (See listing: 1); we chose such standard format in order to easily allow the replacement of the CEP with any other one that from time to time can be considered more specific or efficient for usage.

The XML generated with this schema contains all the necessary information to interact with the specific knowledge-base used. In particular, into the field RuleName of the XML, DEPER will put the name of the request. The content of the RuleBody field is a rule, written using the *Drools* rule syntax that will be loaded on the GLIMPSE knowledge base. Drools is a rule engine based on Charles Forgy's Rete algorithm [13].

On an event-based monitoring infrastructure, as GLIMPSE, the gathered information is provided in form of events. An event is an atomic description, a smaller part of a more large and complex process at application level. In CONNECT, an event represents a method invocation on a remote web service: the invocation, coming from the producer to the consumer, is captured when it comes through the CONNECTor, encapsulated into a ConnectBaseEvent object, and sent through the CONNECT bus.

The detailed structure of a ConnectBaseEvent is described in Figure 4.

For completeness we note that, to provide a more abstract generation of a rule for monitoring non-functional properties, we are developing a Property Meta-Model (PMM) [16], from which users can generate their own rule model and, using a model-driven approach, this can then be translated directly to the desired/-more performant/available CEP language. We leave the metamodel outside the scope of the present paper.

5. Off-line and on-line integrated: the way to adapt assessment under uncertainty/evolution

After having introduced the pre-deployment and run-time analysis methods under development, we focus here on their synergic usage, through which adaptive

[3] Antonella: *qui va detto in che cosa consiste questo messaggio*

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

Listing 1: The Complex Event Rule XSD

Figure 4: The ConnectBaseEvent Interface

assessment is pursued. Basically, the dynamicity and evolution of the targeted environment lead to potential sources of uncertainty, which undermine the accuracy of the off-line analysis. To cope with this issue, adaptive dependability assessment is investigated, which exploits run-time monitoring to re-calibrate and enhance the dependability and performance prediction along time. In brief, the picture of the synergic usage is the following. At design time, stochastic model-based analysis is performed as a pre-deployment method to support the synthesis of a CONNECTor suitable to allow interoperability among the systems willing to connect under required dependability and performance levels. While the prediction so obtained plays an important role in guiding the building of the CONNECTor, it might suffer from unacceptable inaccuracy because of possibly limited knowledge at analysis time or successive context evolution. Through monitoring properly selected events at run-time and collecting them along several executions, we can identify changes that require to be accounted for by a new iteration of the model-based analysis.

5.1. Updater

As shown in Figure 2, Updater is the module of the DEPER architecture in charge of interacting with the Monitor enabler to refine the accuracy of model parameters through on-line observations. Inaccuracy of the non-functional values used in the off-line analysis at CONNECTor design time is mainly due to two possible causes: i) limited knowledge of the NSs characteristics acquired by DEPER/Discovery enablers; ii) evolution along time of the NSs, as naturally

accounted for in the CONNECT context.
Updater receives inputs from both internally to DEPER (from the Evaluator module) and externally (from 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 the 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 off-line estimated value used in the 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. Then, should the new values determined via inference techniques on on-line collected data be outside the reference range values, the consequence is that the synthesized CONNECTor does not meet anymore the stated requirements and re-adjustments at synthesis level are necessary. 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.

The activity diagram that describes the Updater phase is shown in Figure 5.

As reported in [24], 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 of the population (called *sample*) 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

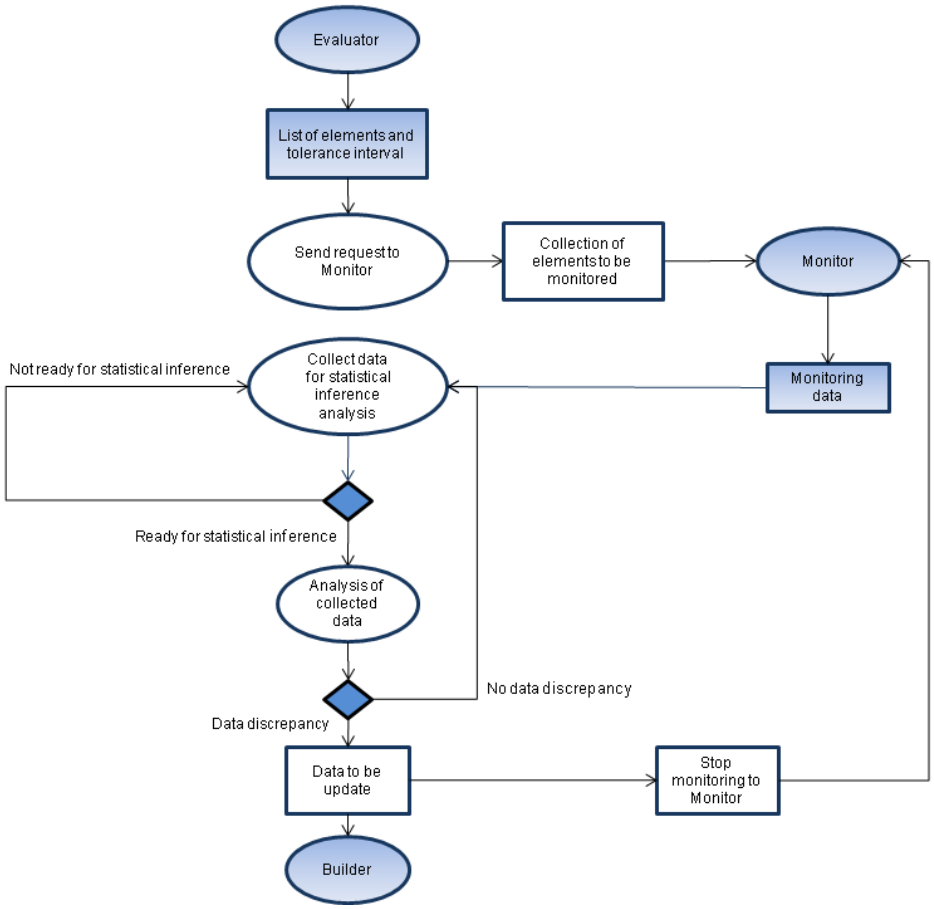

Figure 5: The Updater activity diagram

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, with a *confidence level* α. In this way we are confident that the confidence interval contains the real value of the parameter under analysis.
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:

$$n \geq \left\lceil \left(\frac{z_{\alpha/2} S^2(n)}{\Delta} \right)^2 \right\rceil \tag{1}$$

where the value of $z_{\alpha/2}$ is tabulated. When the sample size is relatively small ($n < 30$), we can use the Student *t* distribution [24].
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.

The iterative algorithm proceeds as follows:

1. $n_0 = \infty$ (initialization);
2. $n_1 = \left(\frac{2 \cdot t_{n_0;\alpha/2}}{\Delta} \right)^2 \cdot S^{*2}$;
3. $n_2 = \left(\frac{2 \cdot t_{n_1;\alpha/2}}{\Delta} \right)^2 \cdot S^{*2}$;
4. ...
5. until the last two results are the same.

Following this approach and considering fixed values of confidence interval and confidence level, we are able to define the sample size that the Monitor enabler has to send to the DEPER enabler in order to evaluate the monitored data.

5.2. Integration and interaction

The interaction between DEPER and Monitor can be analyzed through a simple sequence diagram shown in Figure 6, where we intentionally left out system start-up operations.

In detail, DEPER and Monitor interact by using a Publish/Subscribe protocol. The interaction starts when the DEPER enabler sends a JMS message whose payload contains an XML object rule generated using ComplexEventRule classes (as explained in Section 4).

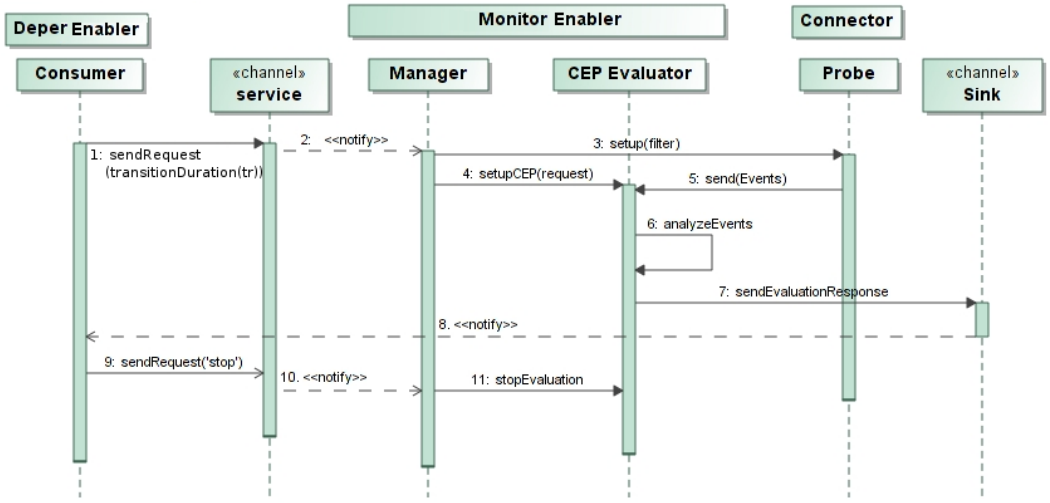

Figure 6: The interaction between DEPER and Monitor

Whenever Monitor receives a request message on the service channel, a new channel dedicated to the requesting enabler (DEPER in this case) is set up to communicate occurrences of the requested pattern.

Once the CONNECTor is deployed, data (events) derived from real executions are sent by the probe to the CONNECT bus. The Monitor enabler gathers those events and using the CEP component, correctly instructed through the ComplexEventRuleAction sent by the DEPER enabler, tries to infer one or more of the patterns on which the DEPER enabler is subscribed.

Upon occurrence of a relevant event the notification to the DEPER enabler is enacted by sending a JMS Message on the dedicated channel created on purpose in the initial phase of the communication (see Figure 6) on which payload is a ComplexEventResponse object (see Listing 2).

The DEPER enabler, in turn, performs a statistical analysis of the monitored observations and uses such information to check the accuracy of the model analysed before deployment. If the model parameters are found to be inaccurate, DEPER updates the model with the new values, and performs a new analysis. If the new analysis evidences that the deployed CONNECTor needs adjustments, a new synthesis–analysis cycle starts.

```
<?xml version="1.0" encoding="UTF−8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http :// www.example.org/ComplexEventResponse/"
  targetNamespace="http :// www.example.org/ComplexEventResponse/"
   attributeFormDefault =" qualified ">
  <element name="ComplexEventResponseList" type="tns:ComplexEventResponse" />
   <complexType name="ComplexEventResponse">
       <sequence>
        <element name="RuleName" type="string" maxOccurs="1" minOccurs="1" />
        <element name="NetworkedSystemSource" type="String" maxOccurs="1" minOccurs="1" />
        <element name="ResponseKey" type="string" maxOccurs="1" minOccurs="1" />
        <element name="ResponseValue" type="string" maxOccurs="1" minOccurs="1" />
       </sequence>
   </complexType>
</schema>
```

Listing 2: The Complex Event Response XSD

6. Case-study

In this section, we show how the integration between DEPER and GLIMPSE can be exploited in the context of a demonstrative scenario.

6.1. Terrorist alert scenario

We consider the CONNECT Terrorist Alert scenario [8], 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.

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

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

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.

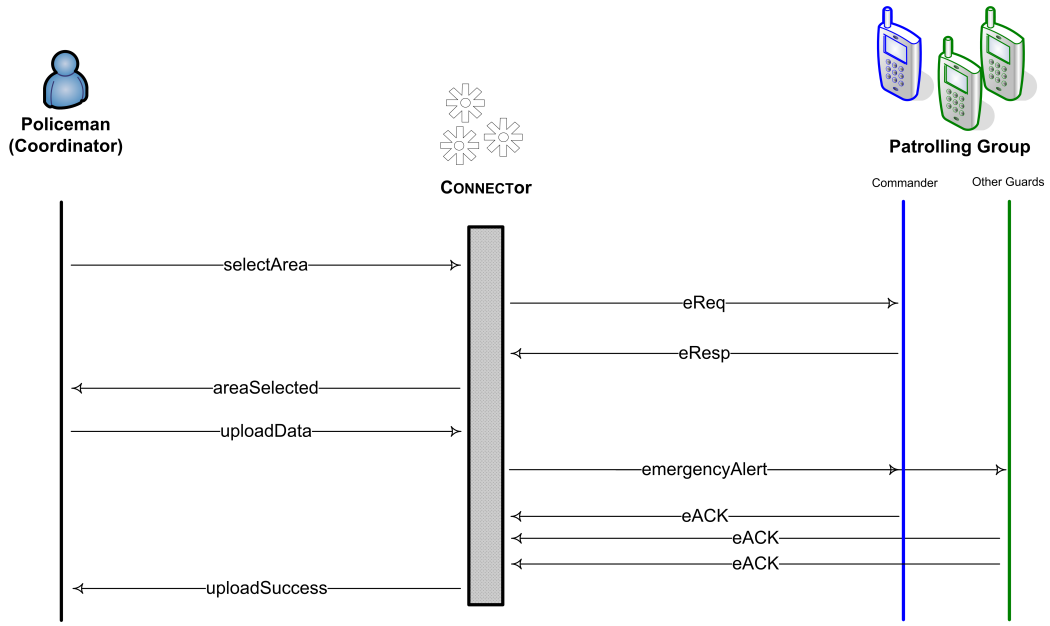

Figure 7: Terrorist Alert Scenario: Sequence diagram of the messages exchange

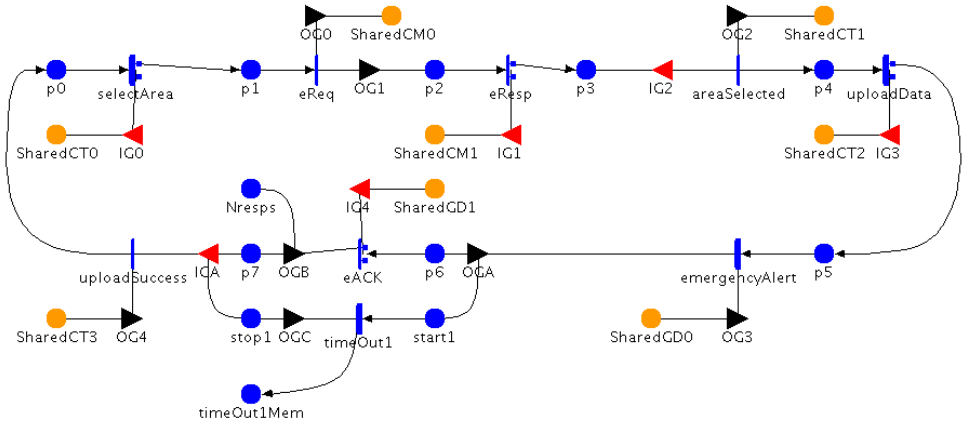

Figure 8: Terrorist Alert Scenario: SAN Model of the CONNECTor

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 synthesize on-the-fly a CONNECTor. The needed

mappings are shown in Figure 7 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.

6.2. On-line analysis

Taking as a reference the above described scenario, we focus in the following on the basic interactions between DEPER (in particular, its Updater module) and GLIMPSE enablers, performed to exchange requests for monitoring and to gather monitored data. Figure 8 depicts the dependability and performance model of the synthesized CONNECTor built by DEPER at design time, using the SAN formalism [23]. 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 *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. The 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) transitions `eReq` and `eResp`, for the latency measure, and ii) the transition `emergencyAlert` for the coverage measure. These model parameters represent the duration of the transitions executed by the NS requesting the communication. Refining the pre-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.

The GLIMPSE infrastructure, more specifically its Manager component, receives the DEPER requests and sets up the ComplexEventProcessor with the provided rule.
The events flowing in from Probes are structured in a `ConnectBaseEvent` object (see Figure 4), that provides all the necessary informations for an accurate pattern recognition.

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 from the Probe instrumented into the peer software component is shown in Figure 9 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) 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 latency (line 35) and to provide it to DEPER (line 40-41).

Indeed, the rule `pending request` in the Listing 3, (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 significant 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.

```
<?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: Sample Request from DEPER enabler to Monitor

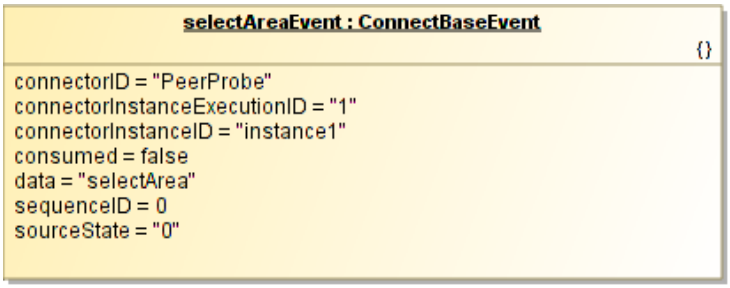

Figure 9: The selectArea Event Sent from Peer Probe

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 5.1 to the mean failure probability for each scenario's configuration, the overall mean failure probability is obtained. Table 1 summarizes the data involved in this experiment to obtain the refined value of 0.1416 for the parameter under observation. The value assumed during pre-deployment dependability analysis was 0.05, a clearly divergent value calling for a new evaluation of the coverage measure.

Number of Guards	Occurences 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 1: Elaboration of data from Monitor to update the failure probability parameter

Figure 10 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 surprising, at increasing the failure probability, the coverage decreases. The coverage value obtained through the pre-deployment analysis is 0.9, fully satisfying the requirements that means the requirements are satisfied. But the coverage value after updating the failure probability parameter is 0.73, no more a satisfactory value. The CONNECTor needs to be enhanced; therefore DEPER informs the Synthesis enabler about the analysis results and appropriate actions are taken by Synthesis (typically, a new CONNECTor is synthesised).

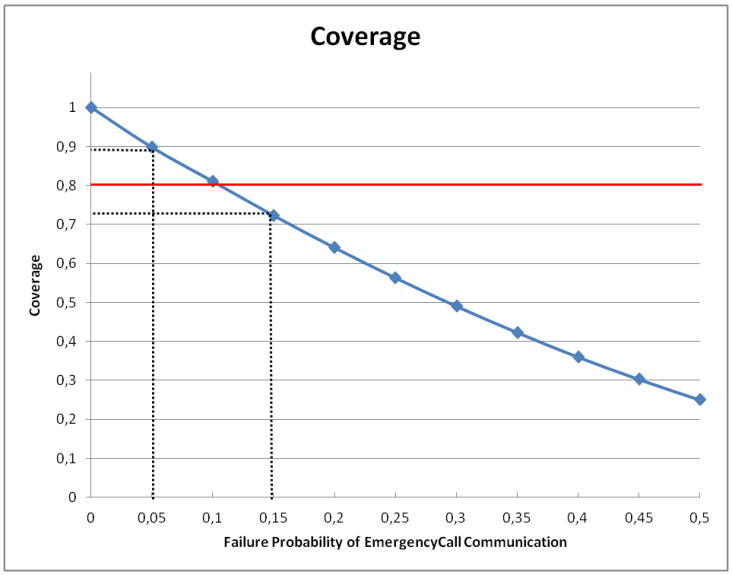

Figure 10: Trend of Coverage as a function of failure probability of the EmergencyCall channel

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 8. 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 2 summarizes the mean values for the time duration of the two transitions (in time units). Through the *probability plotting paper* method [**?**], 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 2: Timing values from Monitor

Figure 11 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.

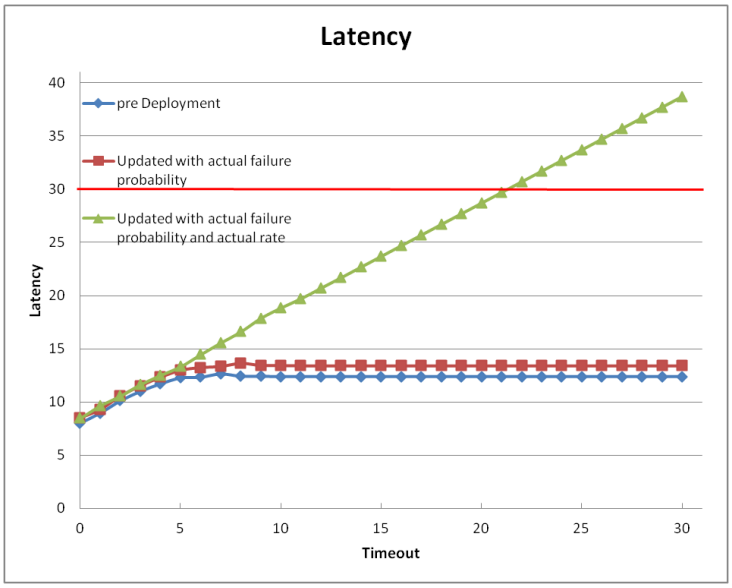

Figure 11: Trend of Latency as a function of Timeout

Similarly, Figure 12 shows the trend of *coverage* (on the *y* axis) at increasing values of *Timeout* (on the *x* axis). As in the previous figure, 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.

7. Related work

This paper addresses the integration between stochastic model-based analysis of dependability and performance and event-based monitoring, in order to meet the needs of adaptive analysis in dynamic and evolving contexts.

Stochastic model-based approaches for quantitative analysis of non-functional properties have been largely developed along the last decades and documented in

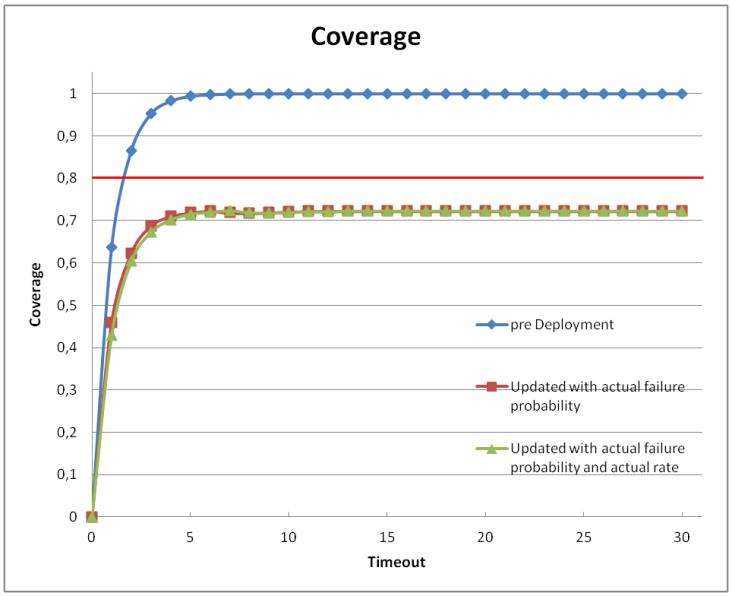

Figure 12: Trend of Coverage as a function of Timeout

a huge literary production on this topic. The already cited papers [20, 5] provide a survey of the most popular ones. The choice of the most appropriate type of model to employ depends upon the complexity of the system under analysis, the specific aspects to be studied, the attributes to be evaluated, the accuracy required, and the resources available for the study. The prototype implementation of our DEPER enabler is based on Stochastic Activity Networks (SANs) [23], a variant of the Stochastic Petri Nets class.

With regard to monitoring, various approaches have been recently proposed. Similarly to GLIMPSE, also [21] presents an extended event-based middleware with complex event processing capabilities on distributed systems, adopting a publish/subscribe infrastructure, but it is mainly focused on the definition of a complex-event specification language. The aim of GLIMPSE is to give a more general and flexible monitoring infrastructure for achieving a better interpretability with many possible heterogeneous systems.

Another monitoring architecture for distributed systems management is presented in [14]. Differently from GLIMPSE, this architecture employs a hierarchical and layered event filtering approach. The goal of the authors is to improve monitoring scalability and performance for large-scale distributed systems, minimizing the monitoring intrusiveness.

Many works focus on the definition of expressive complex event specification languages. Among them, GEM [15] is a generalized and interpreted event monitoring language. It is rule-based (similar to other event-condition-action approaches) and also provides a tree-bases detection algorithm taking into account communication delay. Also the Snoop language [7] 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 [11]. It has a simple syntax and a semantics based on a first order temporal logic. 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. In our prototype we used Drools because ServiceMix offers it as business rule engine.

The focus of our approach is in the combined usage of pre-deployment model-based analysis and run-time observations via monitoring. Preliminary studies that attempt combining off-line with on-line analysis have already appeared in the literature. A major area on which such approaches have been based is that of autonomic computing. Among such studies, in [19], an approach is proposed for autonomic systems, which combines analytic availability models and monitoring. The analytic model provides the behavioural abstraction of components/subsystems and of their interconnections and dependencies, while statistical inference is applied on the data from real time monitoring of those components and subsystems, to assess parameter values of the system availability model. Through on-line monitoring and estimation of system availability, adaptive on-line control of system availability can then be obtained. In [22], an approach is proposed to carry out run-time reliability estimation, based on a preliminary modelling phase followed by a refinement phase, where real operational data are used to overcome potential errors due to model simplifications. The model is based on Discrete Time Markov Chain, and a prototype version of the monitoring system has been implemented, that is initially trained with the reference model and the preliminary reliability estimation, and then uses operational data to compute the on-line reliability level. Our approach aims at proposing a general and powerful evaluation framework, tailored to a variety of dependability and performance metrics, to meet a wide spectrum of system requirements and adaptation needs.

Another point of strength of our approach is the ability to automate the analysis process along the system lifetime, from the design phase to the operational one, based on transformation rules. Research on definition and development of transformation-based verification and validation environments are being pursued since several years. Providing automatic/automated transformations methods

from system specification languages to modelling languages amenable to perform dependability analysis has been recognized as an important support for improving the quality of systems. In addition, it favours the application of verification and validation techniques at industry level, where these methods are not widely used primarily due to the high level of abstractness of the mathematical modelling and analysis techniques. To provide some examples, the Viatra tool [10] automatically checks consistency, completeness, and dependability requirements of systems designed using the Unified Modeling Language. The Genet tool [6], based on the theory of regions [12], allows the derivation of a general Petri net from a state-based representation of a system. Our work addresses the transformation from the LTS formalism, as system specification language, to SAN, as dependability modelling language. Since there are some steps in common with the Genet tool and related theory, we partially reused available results from this previous study in our prototype implementation.

8. Final discussion and conclusions

In this paper, we have tackled the challenge of dependability and performance analysis in dynamic and evolving systems, whose peculiarities make traditional methods largely inadequate. Our proposal to cope with the issues raised in the addressed context resorts to integrate pre-deployment stochastic model-based analysis with run-time monitoring, to achieve adaptive dependability assessment through re-calibration and enhancement of the dependability and performance prediction along time. The aim of this two-phase analysis framework is twofold. On one side, the stochastic model-based analysis performed at pre-deployment time provides a primary support to the realization of the "rightest" system, given the partial knowledge about the involved subsystems and environment conditions. However, the resulting unavoidable potential inaccuracy on the prediction so obtained could lead to more or less severe consequences if awareness about it is never acquired. This is the point were monitoring provides its contribution, by observing events which help to enhance the previously performed analysis. In fact, through monitoring properly selected events during execution and collecting them along multiple executions we can identify changes that require to be accounted for by a new iteration of the model-based analysis and possible. ❒[4]

[4] Felicita: *se ci sono i valori di range non c'e'bisogno di fare una nuova analisi, ma solo dare un feedback per una nuova sintesi*

Although not novel in its basic principles, this pre-deployment and run-time integrated framework is proposed as a general, automated approach to fulfill the dependability and performance assessment needs in dynamic and evolving contexts.Therefore, the work done so far constitutes an important step towards the definition of an automated and adaptive process to provide dependability and performance analysis accounting for modern and future application needs. Although prototypes of DEPER, GLIMPSE and their integration are already available, additional effort is needed to fully embed in them the many potentialities offered by the approach. Especially, techniques would be desirable to balance between time to produce results and their accuracy. For instance, in the automatic generation of the dependability and performance model from the specification of the CONNECTed system, techniques could be sought able to optimise the model on the basis of the specific metrics that needs to be assessment. Also, compositional solution methods for the dependability and performance model would be highly attractive, possibly reusing partly solved model, e.g., when the synthesised CONNECTor is derived as specialisation of an already existing CONNECTor that has already been analysed, or when already analysed dependability mechanisms are introduced in the dependability model.
In view of better exploiting the functionalities of the GLIMPSE monitoring infrastructure, able to observe more complex events as aggregation of elementary ones, GLIMPSE could be instrumented to evaluate final properties of interest, like coverage. Then, the comparison with pre-deployment assessment made through DEPER would become a powerful cross-validation operation, reinforcing the confidence in the design-time forecast, or revealing inadequacy of the assumed model parameter (we exclude potential deficiencies in the set-up of the model itself since the dependability and performance models are automatically derived from the LTS specifications).
Finally, we would like to underline that, despite we developed the approach assuming the context of the EU project CONNECT to make the exposition more concrete, the approach is general and applicable to other contexts sharing the characteristics of evolution and partially known specifications.

References

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

[4] A. Bertolino, A. Calabrò, F. Di Giandomenico, and N. Nostro. Dependability and performance assessment of dynamic connected systems. In M. Bernardo and V. Issarny, editors, *Formal Methods for Eternal Networked Software Systems*, volume 6659 of *LNCS*, pages 350 – 392. Springer, 2011.

[5] A. Bondavalli, S. Chiaradonna, and F. Di Giandomenico. Model-based evaluation as a support to the design of dependable systems. In Hassan B. Diab and Albert Y. Zomaya, editors, *Dependable Computing Systems: Paradigms, Performance Issues, and Applications*, pages 57–86. Wiley, 2005.

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

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

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

[9] ReSIST Consortium. Eu project resist: Resilience for survivability in ist. deliverable d13: From resilience-building to resilience-scaling technologies: Directions. Technical report, 2007. http://www.resist-noe.org/.

[10] Gyorgy Csertan, Gabor Huszerl, Istvan Majzik, Zsigmond Pap, Andras Pataricza, Daniel Varro, and Dniel Varr. Viatra - visual automated transformations for formal verification and validation of uml models. In *17th IEEE International Conference on Automated Software Engineering (ASE'02)*, pages 267–270, 2002.

[11] Gianpaolo Cugola and Alessandro Margara. TESLA: a formally defined event specification language. In *Proceedings of DEBS*, pages 50–61, 2010.

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

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

[14] Ehab Al-Shaer Hussein, Hussein Abdel-wahab, and Kurt Maly. HiFi: A New Monitoring Architecture for Distributed Systems Management. In *Proceedings of ICDCS*, pages 171–178, 1999.

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

[16] Antinisca Di Marco, Claudio Pompilio, Antonia Bertolino, Antonello Calabrò, Francesca Lonetti, and Antonino Sabetta. Yet another meta-model to specify non-functional properties. In Domenico Bianculli, Sam Guinea, Andreas Metzger, and Andrea Polini, editors, *QASBA*, ACM International Conference Proceeding Series, pages 9–16. ACM, 2011.

[17] 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*, pages 139 – 146. IEEE, Kobe, Japan, June 2011.

[18] P. Masci, N. Nostro, and F. Di Giandomenico. On enabling dependability assurance in heterogeneous networks through automated model-based analysis. In *Proc. Third International Workshop, SERENE 2011*, volume 6968 of *LNCS*, pages 78 – 92. Sringer, Geneva, Switzerland, September 2011.

[19] Kesari Mishra and Kishor S. Trivedi. Model based approach for autonomic availability management. In *ISAS 2006*, volume 4328 of *LNCS*, pages 1–16. Lecture Notes in Computer Science, 2006.

[20] David M. Nicol, William H. Sanders, and Kishor S. Trivedi. Model-based evaluation: from dependability to security. *IEEE Transactions on Dependable and Secure Computing*, 1:48–65, January-March 2004.

[21] P.R. Pietzuch, B. Shand, and J. Bacon. Composite event detection as a generic middleware extension. *Network, IEEE*, 18(1):44 – 55, jan. 2004.

[22] K. S. Trivedi R. Pietrantuono, S. Russo. Online monitoring of software system reliability. In *Proc. EDCC '10 - 2010 European Dependable Computing Conference*, pages 209–218. IEEE Computer Society, 2010.

[23] W. H. Sanders and L. M. Malhis. Dependability evaluation using composed SAN-based reward models. *Journal of Parallel and Distributed Computing*, 15(3):238–254, 1992.

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