On-the-Fly Dependable Mediation between Heterogeneous Networked Systems

Antonia Bertolino, Antonello Calabrò, Felicita Di Giandomenico, Nicola Nostro, Paola Inverardi, Romina Spalazzese

On-the-Fly Dependable Mediation between Heterogeneous Networked Systems

Antonia Bertolino[1], Antonello Calabrò[1], Felicita Di Giandomenico[1], Nicola Nostro[1], Paola Inverardi[2], and Romina Spalazzese[2]

[1] Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", CNR, Pisa, Italy
[2] Department of Computer Science, University of L'Aquila, Coppito (AQ), Italy
firstname.lastname@isti.cnr.it,
{paola.inverardi,romina.spalazzese}@univaq.it

Abstract. The development of next generation Future Internet systems must be capable to address complexity, heterogeneity, interdependency and, especially, evolution of loosely connected networked systems. The European project CONNECT addresses the challenging and ambitious topic of ensuring eternally functioning distributed and heterogeneous systems through on-the-fly synthesis of the CONNECTors through which they communicate. In this paper we focus on the CONNECT enablers that dynamically derive such connectors ensuring the required non-functional requirements via a framework to analyse and assess dependability and performance properties. We illustrate the adaptive approach under development integrating synthesis of CONNECTors, stochastic model-based analysis performed at design time and run-time monitoring. The proposed framework is illustrated on a case study.

1 Introduction

We live in the Future Internet (FI) era, which is characterized by unprecedented levels of connectivity and evolution. Software systems are increasingly pervasive, dynamic and heterogeneous, and many -even critical- aspects of modern society rely on their continuous availability and seamless interoperability. Ensuring the successful dynamic composition among heterogenous, independently developed Networked Systems (NSs) raises the need of novel computing paradigms, such as the revolutionary approach to *on-the-fly connection* pursued within the European FP7 Future and Emerging Technology Project CONNECT.

CONNECT follows the ambitious goal of enabling *seamless* and *dependable* interoperability among NSs in spite of technology diversity and evolution. The key idea is to compose systems by generating on-the-fly the interoperability solution necessary to assure the connection among the heterogenous NSs both at application and at middleware level. The synthesized solution is called a CONNECTor or also *mediating connector* or *mediator* for short; the system obtained from the composition of the NSs through the CONNECTor is said the CONNECTed System.

Automatically synthesized CONNECTors are concrete emergent entities that mediate the NSs discrepancies, i.e., they translate and coordinate mismatching interaction protocols, actions, and/or data models, allowing applications to interact effectively.

M. José Escalona, J. Cordeiro, and B. Shishkov (Eds.): ICSOFT 2011, CCIS 303, pp. 20–37, 2012.

A synthesized CONNECTor, if it exists, provides by construction a correct solution to functional interoperability among the NSs.

This is however not sufficient: effective interoperability also requires that such on-the-fly CONNECTed systems provide the required non-functional properties and continue to do so even in presence of evolution. The CONNECTors are not a priori guaranteed to provide the desired non-functional properties for the CONNECTed system, thus a suitable and adaptive assessment framework is required. In this paper, we focus on the problem of ensuring the non-functional properties for CONNECTed systems.

Concerning dependability and performance properties, several challenges arise. Off-line, or pre-deployment, assessment can help to take appropriate design decisions by providing a priori feedback about how the system is expected to operate. Nevertheless, the unavoidable high chance of inaccurate/unknown model parameters might result in inadequate analysis results. Moreover, the many possible variations occurring during the system lifetime would require to foresee and analyze all the possible scenarios which could take place at run-time (e.g., to be stored in a look-up table from which to retrieve the correct analysis upon a scenario's occurrence). On the other hand, resorting to processing the measurements collected in real operation at a later stage, e.g. in periodic reviews, may be inadequate, since by the time the observations are processed the operational environment may have changed. Furthermore, in the CONNECT context the above problems are exacerbated because, as said, components are dynamically assembled to satisfy an emergent user goal. In this scenario the only part of the system under control is the synthesized CONNECTor, whereas for the NSs only declarative or learned on-the-fly knowledge can be assumed.

To contribute to overcome the above issues, we have developed an approach which tries to combine the benefits of both pre-deployment and processing of data obtained from real executions. The proposed assessment framework combines stochastic model-based analysis [1] with continuous on-line assessment of non-functional properties through a lightweight flexible monitoring infrastructure, and applies such approach to the on-the-fly CONNECT system into a continuous loop.

In the following we initially provide the context for our approach by introducing the CONNECT architecture (Section 2). Then we introduce the case study that is used to demonstrate the applicability of the integrated analysis framework (Section 3). Mediator synthesis (Section 4), pre-deployment analysis (Section 5) and the run-time monitor (Section 6) are briefly presented, and hence their synergic usage (Section 7), through which adaptive assessment is pursued. Finally we overview related work (Section 8) and draw conclusions (Section 9).

2 The 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. As said in the introduction, the ambition of the project is to have eternally functioning systems within a dynamically evolving context.

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

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

Discovery Enabler: discovers the NSs, catches their requests for communication and initiates the CONNECT process. We tend to make the minimum possible assumptions on the information (called the *affordance*) that NSs must provide;

Learning Enabler: we use active learning algorithms to dynamically determine the interaction behaviour of a NS and produce 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. More details on this enabler are given in Section 4;

Deployment Enabler: deploys and manages the synthesized CONNECTors;

Monitor Enabler: collects raw information about the CONNECTors behaviour, filters and passes them to the enablers who requested them. The CONNECT monitoring infrastructure is further described in Section 6;

DEPER Enabler: this is the enabler assessing dependability and performance properties and is described in detail in Section 5;

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. For reasons of space, we do not deal with this enabler in this paper.

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)).

In this paper we provide a snapshot of the functioning of CONNECT over the case study introduced in the following section. For space limitation, we focus on the interaction among Synthesis, Dependability&Performance and Monitor, which are highlighted by tick borders in Figure 1. We show first how a dependable CONNECTor is deployed (pre-deployment analysis), and then how, via the feedback obtained through run-time monitoring of the CONNECTor behaviour, CONNECTor adaptation is triggered and managed. In particular, we devise a process for the CONNECTor creation that is supported by powerful infrastructures made available by CONNECT itself. Once the Discovery Enabler discovers new devices, the CONNECT supporting infrastructure starts the computation of a CONNECTor on the fly –if possible. Then, when the intent to communicate is manifested, the CONNECTor -if it exists- is partially computed and is hence concretized. Note that the CONNECTor could not exist because NSs are not compatible and then do not have a way to communicate.

3 Case Study

In this section, we present our running example for presenting how synthesis, analysis and monitoring work in integrated way.

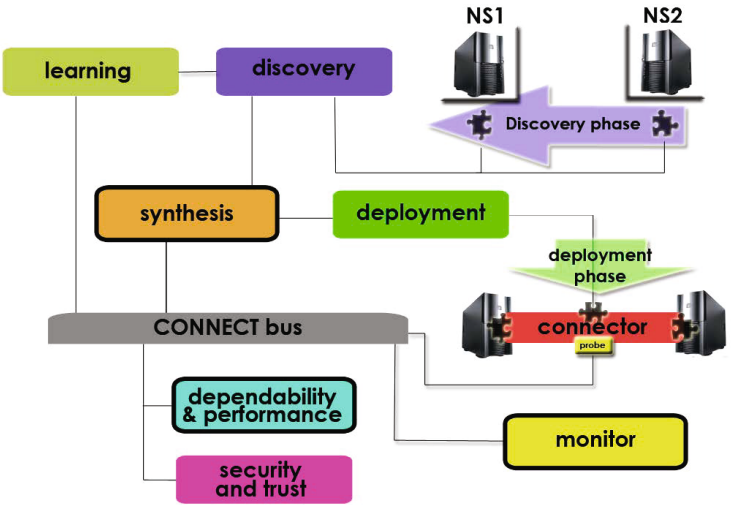


Fig. 1. The CONNECT architecture

3.1 Terrorist Alert Scenario

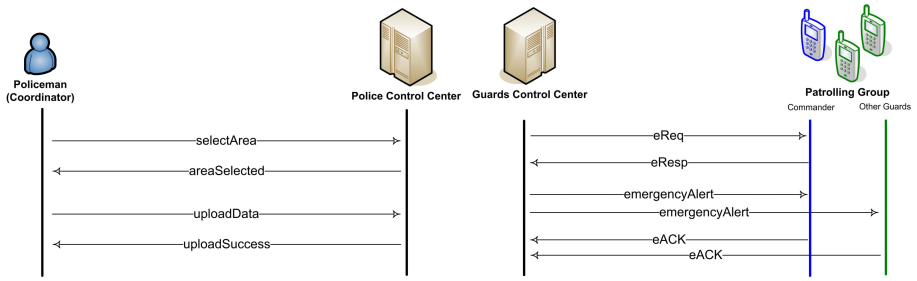
We consider the CONNECT Terrorist Alert scenario [2], depicting the critical situation that during a show in the stadium, the stadium control center spots one suspect terrorist moving around. This emergency situation makes it necessary to exchange information between policemen and security guards patrolling the surroundings of the stadium equipped with heterogeneous applications.

Each policeman can exchange confidential data with other policemen with a *SecuredFileSharing* application. Security guards, on the other hand, exchange information by using another application, denominated *EmergencyCall*. The two applications have the same aim (i.e., enable information exchange), but use different protocols as we describe in the following.

SecuredFileSharing

- The peer that initiates the communication denominated *coordinator* (the Policemen of our example) sends a broadcast message (`selectArea`) to selected peers operating in a specified area of interest (the Police control center of our example).
- 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 or the coordinator can receive an exception.

An example of message flow between a coordinator, i.e., a Policeman, and a Police control center is depicted in Figure 2(a) while the application behaviour of another Policeman is shown in Figure 3. It is worth to notice that, for readability, in the figure



(a) *SecuredFileSharing* Application (b) *EmergencyCall* Application

Fig. 2. Sequence Diagrams of the applications

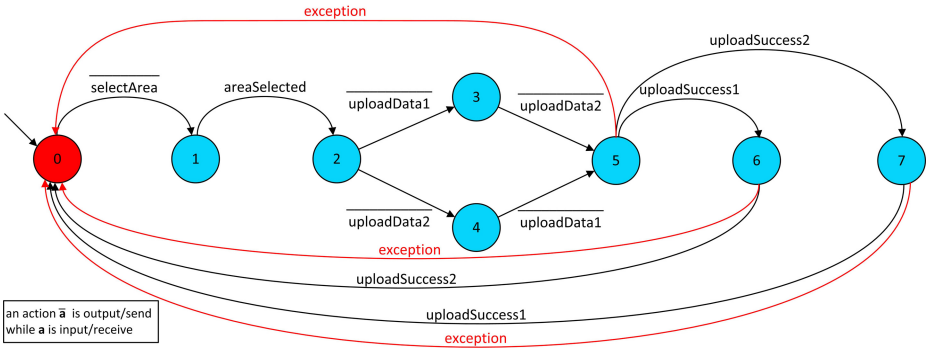


Fig. 3. LTS of the *Policeman* application

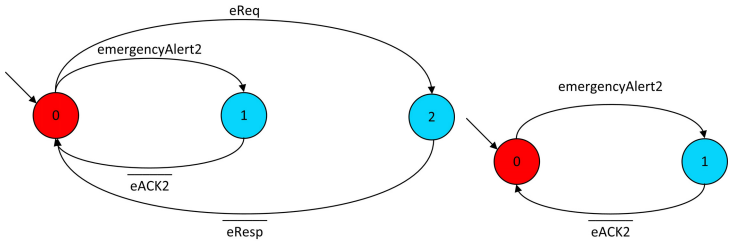
the LTS of the Policeman describes the sending of the broadcast alert to two selected peers while in the experiments we conducted, we used 11 selected peers.

The affordance of the Policeman application includes also the following *non-functional requirement*: she/he should receive the 65% of acknowledgements for the alerts sent within 30 time units, otherwise a failure is reported.

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.

The message flow among the Guard control Center, the Commander and the Other Guards is depicted in Figure 2(b). Figure 4(a) shows the LTS of the Commander, and the LTS of the Other Guards is shown in Figure 4(b).



(a) Commander. (b) Other guards.

Fig. 4. LTSs of the *EmergencyCall* application

3.2 CONNECT in the Case Study

With reference to CONNECT architecture (see Figure 1), the two NSs which need to communicate but are not a priori compatible, are the devices implementing the described applications *SecuredFileSharing* and *EmergencyCall*. Hence, to allow a Policeman and the Guards, operating in the zone where the suspect terrorist has escaped, to communicate, CONNECT proposes to automatically synthesize on-the-fly a CONNECTor that can mediate between the two different communication protocols. Such CONNECTor should be able to support the exchange of information, while also fulfilling possible non-functional requirements.

4 Automated Mediator Synthesis

Our focus is on the interoperability between heterogeneous protocols. By *interoperability* we mean the ability of protocols to *correctly communicate and coordinate* i.e., to *correctly synchronize*. In other words, two systems successfully interoperate if they correctly exchange *compatible conversations* or *compatible traces*. By *heterogeneous protocols* we mean that, although in principle they could interact since they have compatible (i.e., complementary) functionalities, protocols can be characterized by discrepancies that may undermine their ability to seamlessly interoperate (i.e., communicate and coordinate). Discrepancies include incompatible interaction protocols and/or different interfaces meaning different actions and/or data models. Examples of heterogeneous application protocols are the Policeman and Commander of the Patrolling Group of the case study.
In order to enable interoperability among heterogeneous protocols, we devised a theory for the automated synthesis of CONNECTors [3, 4], also called mediating connectors or mediators for short. Figure 5 provides an overview of our methodology.

Our approach takes as input the descriptions of two NSs and in particular their behavioral protocols, described as Labeled Transition Systems (LTSs), together with their ontological information conceptualizing their actions through an application domain ontology. By referring to the case study, the synthesis takes as input the LTS of the Policeman, the LTS of a Commander of the Patrolling Group and of its Guards and their ontologies and follows a process made up by three phases or steps as described in the following.

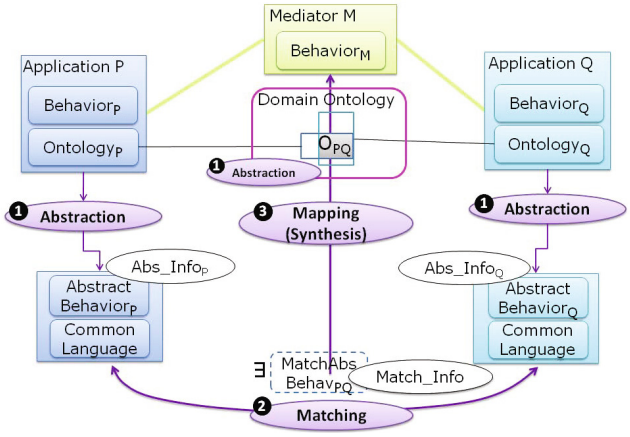


Fig. 5. An overview of the mediator synthesis approach

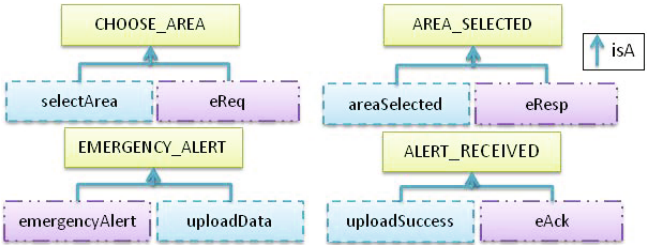


Fig. 6. Domain Ontology (upper case elements) where the ontologies of both protocols have been mapped (Dots and dashes boxes - Ontology of the Guards Control Center; dashed boxes - Ontology of the Policeman)

1. *Abstraction* that makes models comparable and, if possible, reduces their size making it easier and faster to reason on them. Reasoning on ontologies, a common language for both protocols is identified on the domain ontology and used to relabel them. The common language of our case study is illustrated by the elements with upper case names in Figure 6.
2. *Matching* that checks the NSs compatibility identifying possible mismatches. Compatible protocols can potentially interoperate despite they show some differences. That is, communication and coordination between such protocols is possible in principle since they are semantically equivalent and complementary, but cannot be achieved seamlessly because of *heterogeneity*: *mismatches* and/or *third parties conversations*. Examples of mismatches are: protocol languages have (i) different granularity, or (ii) different alphabets; protocols behavior have different sequences of actions with data (i.e., traces) because of (a.1) the order in which actions and data are performed by a protocol is different from the order in which the other protocol performs the complementary actions with data. Protocols behavior may have different sequences of actions also because of (a.2) interleaved actions related to *third parties conversations* i.e., with other systems, the environment. In some cases, as

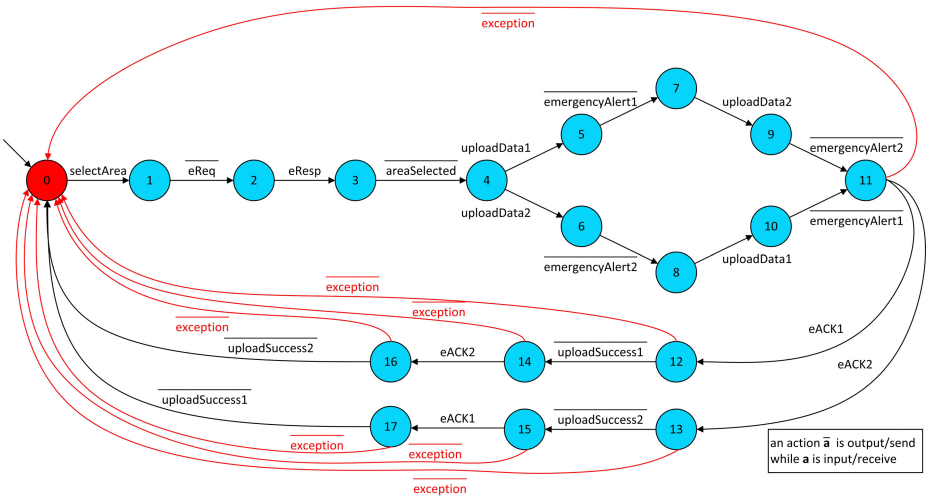


Fig. 7. Synthesised CONNECTor for the case study

for example (i), (ii) and (a.1), it is necessary to properly perform a manipulation of the two languages. In the case (a.2) it is necessary to abstract the third parties conversations that are not relevant to the communication. Referring to the case study, the heterogeneities we identify are what we call signature mismatches [5, 6], i.e., the two protocols use different names for semantically equivalent concepts.
Synchronization between protocols, thus, can be achieved under mediation i.e., through a mediator that while managing such mismatches, allows protocols to effectively exchange compatible traces (sequences of actions).

3. *Mapping* or *Synthesis* that produces a mediator that mediates the found mismatches so enabling the NSs to communicate. Figure 7 illustrates the synthesized CONNECTor between the Policeman and the Commander of the Patrolling Group applications of our case study.
   - 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 to all the Guards (Commander included).
   - Each `eACK` message automatically sent by the Guards' devices that correctly receive the `emergencyAlert` message are translated into a `uploadSuccess` message for the Policeman.

Note that still for figure readability, the illustrated CONNECTor is between one Policeman, a Commander of the Patrolling Group and one Guard.

A **mediator** is then a protocol that allows communication and coordination among compatible protocols by mediating their differences. It serves as the locus where

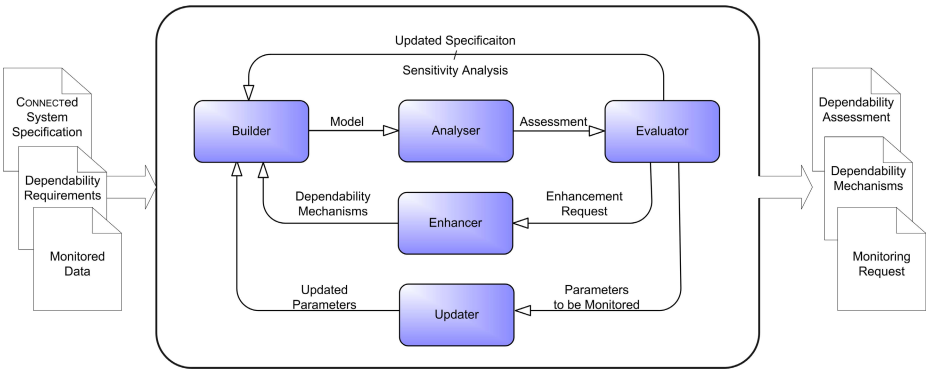


Fig. 8. Architecture of the Dependability&Performance (DEPER) Enabler

semantically equivalent and complementary actions are correctly synchronized thus enabling (a mediated) interoperability among protocols.

In summary, by reasoning about the protocols discrepancies, our theory automatically identifies and synthesizes an *emerging mediator* that solves them thus allowing protocols to interoperate. Automatically synthesized CONNECTors are concrete emergent entities that translate and coordinate mismatching interaction protocols, actions, and/or data models, letting applications interact effectively

5 Pre-deployment Analysis to Support CONNECTor Synthesis

Pre-deployment assessment is a crucial activity to drive the system design towards a realization compliant with the required quality levels. 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. As briefly outlined in the Introduction, the pre-deployment assessment in CONNECT is performed by the Dependability and Performance enabler, introduced in Figure 1 and shortly referred to as DEPER, which exploits State-Based Stochastic modelling and analysis, embedded in an automated process. Figure 8 illustrates the architecture of DEPER that, together with the prototype implementation based on the Möbius evaluation framework, is documented in [7, 8].

Very briefly, the activities of modules Builder, Analyser and Evaluator, triggered in sequence, perform the cycle of the pre-deployment analysis: from the specification of the CONNECTed system (both functional and non-functional) and of the metrics to be analysed, to checking whether the analysis results match with the metrics level, as requested by the NSs.

Evaluator informs Synthesis about the outcome of the check. 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, thus calling the intervention of the Enhancer module. In the other case, 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 the events to be observed on-line by the Monitor enabler.

The attempts to improve dependability and performance of the CONNECTor consist in extending the model with a dependability mechanism, selected from a library of already defined ones (see [9]), until either a successful mechanism is found, or all the mechanisms are exhausted.

The Updater module provides adaptation of the off-line analysis performed at pre-deployment time to cope with changes in, or inaccurate estimates of, model parameters, through interactions with the Monitor enabler (e.g., because of limited knowledge of the NSs characteristics acquired by Learning/Discovery enablers). It receives from Monitor a continuous flow of data for 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 resulting into a significant deviation of the performed analysis, a new analysis is triggered.

5.1 Pre-deployment Analysis in the Terrorist Alert Scenario

Taking as a reference the above described scenario, we focus in the following on the basic interactions between Synthesis and DEPER enablers, performed to exchange requests for dependability analysis of a pre-deployed CONNECTor.

The interaction starts when Synthesis sends a JMS message to DEPER. The message contains the specification of the LTSs of the CONNECTor and of the NSs involved.

Figure 9 depicts the dependability and performance model of the synthesized CONNECTor built by DEPER at design time, using the SAN formalism [10]. The model is obtained through automatic transformation from the LTS specification of the NSs, that is the *SecuredFileSharing* and *EmergencyCall* in the considered scenario. The measure assessed in the evaluation is the *coverage*. Coverage represents a dependability indicator and is given by the percentage of responses the control center receives back from the guards within a certain time T.

Once the message has been received and the SAN models have been built, DEPER starts the coverage analysis through the Möbius tool [11]. The performed analysis is shown in Figure 12 of Section 7 and allows to state that the CONNECTor fully satisfies the coverage requirement with a Timeout greater than 2 time units. Hence, a response is sent, through a JMS message, from DEPER to Synthesis to communicate that the CONNECTor can be dependably deployed.

After the deployment of the CONNECTor a sensitivity analysis from DEPER on the impact of model parameters on the assessment of the selected measure revealed that critical parameters to keep under observation on-line via the Monitor enabler, for the coverage measure, are occurrences of transitions `eACK`. 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,

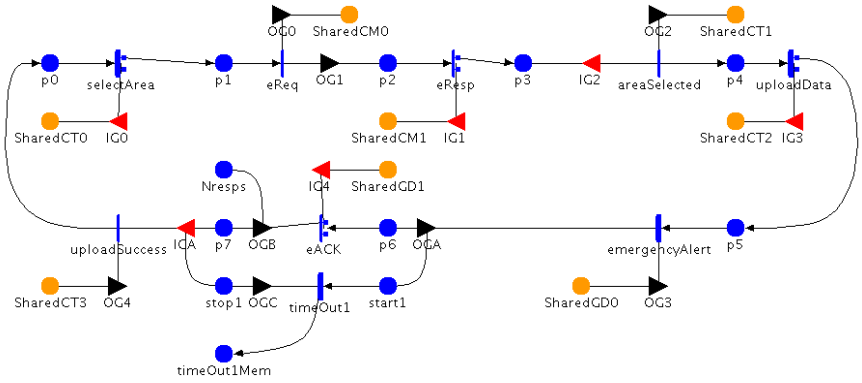


Fig. 9. SAN model of the CONNECTor

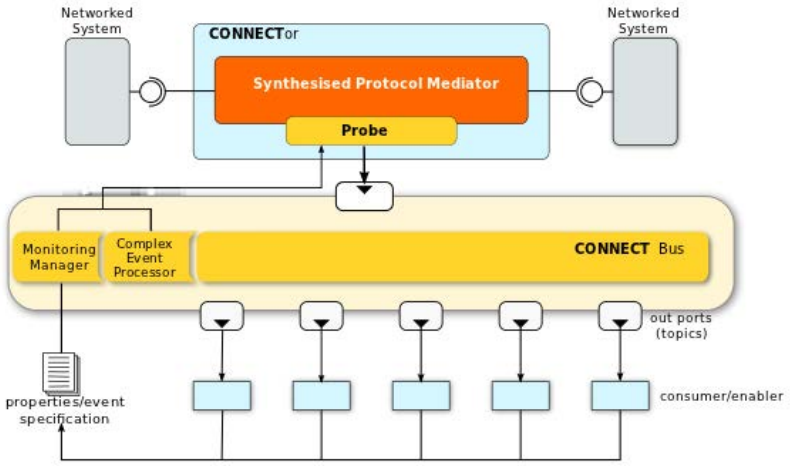


Fig. 10. GLIMPSE architecture

a new analysis round needs to be triggered to understand whether the dependability and performance requirements are still met by the CONNECTed system.

6 Events Observation through Monitoring

Timely and effective run-time adaptation can only be ensured by continuously observing the interactions among the NSs. To this purpose, in CONNECT we have developed a modular, flexible and lightweight monitoring infrastructure, called GLIMPSE[2]. GLIMPSE infrastructure, shown in Figure 10, is totally generic and can be easily applied to different contexts. To provide a better communication decoupling, we adopted a publish-subscribe communication paradigm.

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

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 the evaluation of a non-functional property or interested to receive notification of occurrences of events/exceptions that may occurs into the CONNECTor.

The gathered information is provided in form of events. An event is an atomic description, a smaller part of a larger and more 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 specific object, and sent through the CONNECT bus. A Complex Event Processor (CEP) analyzes the atomic events to infer complex events matching the consumer requests, by a rule engine. In the current GLIMPSE implementation, we adopt the Drools Fusion rule language [12] that is open source and can be fully embedded in the realized Java architecture.

Finally, the Manager module is in charge to manage all the communication between the consumers and the CEP.

7 Continuous Run-Time Adaptation

After having performed the pre-deployment analysis phase and the deployment of the CONNECTor, we focus here on its adaptive assessment via the interaction among Synthesis, DEPER and the Monitor enabler. 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, run-time monitoring is exploited to re-calibrate and enhance the dependability and performance prediction along time. As already pointed out in Section 5, the continuous run-time adaptation of the pre-deployment performed analysis is in charge to the Updater module.

DEPER and Monitor interact by using a Publish/Subscribe protocol. The interaction starts when DEPER sends a JMS message whose payload contains an XML object rule generated using ComplexEventRule classes [7].

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, tries to infer one or more of the patterns to which the DEPER enabler is subscribed.

Upon occurrence of a relevant event the DEPER enabler is notified: the latter, 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.

As an example, we consider the steps to refine the accuracy of the *failure probability* of the communication channel between the *EmergencyCall* application and the CONNECTor. We accumulated data generated from several executions of the CONNECTor, in scenario's configurations with a fixed number of 11 guards, which allowed to refine the value of the failure probability from 0.02, assumed during pre-deployment dependability analysis, to 0.1.

Upon updating the parameter, a new dependability analysis is performed in order to verify if the updated CONNECTor still satisfies the requirement. Unfortunately, the analysis shows that coverage measure does not meet the requirement. The CONNECTor needs to be enhanced. Based on the library of dependability mechanism [9], the *retry mechanism* has been automatically selected and implemented in the already developed CONNECTor model, in order to enhance the CONNECTed system and satisfy the requirement.

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 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, the sender assumes that the communication was not successful, and re-transmits the message.

The SAN model of the retry mechanism is shown in Figure 11. On the sender side, the mechanism creates a message re-transmission policy for re-sending the message at most $N = 3$ times; on the receiver side, the mechanism creates a policy for avoiding duplicated reception of messages and for sending acknowledgements. 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. A detailed description of the mechanism model can be found in [9].

Figure 12 shows the trend of the *coverage* (on the *y* axis) at increasing values of Timeout (on the x axis). Also, the threshold coverage line as specified in the requirement (set to the value 0.65) is reported. 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 coverage have been updated with actual values from the run-time observations; and (iii) the results of the analysis after both the parameters influencing coverage have been updated and a retry mechanism have been implemented to enhance the CONNECTed system. It is worth noting that coverage value obtained through the pre-deployment analysis is fully satisfying the requirement with timeout value greater than 3 (time units). While the coverage value after updating the failure probability parameter never satisfies the requirement, although the value of timeout increases, which means that the estimation of coverage at pre-deployment time was too optimistic. Finally, we note that the analysis performed considering the actual value of failure probability and including the enhanced mechanism provides results on coverage that satisfies the coverage requirement.

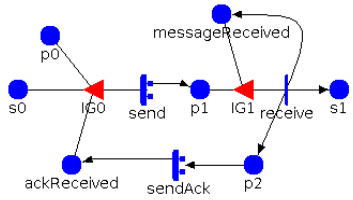


Fig. 11. Retry mechanism

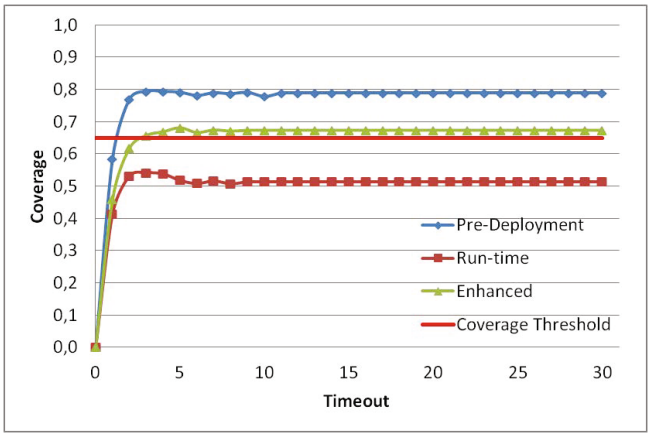


Fig. 12. Trend of Coverage as a function of Timeout

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).

8 Related Work

The automated synthesis of application-layer CONNECTors relates to a wide number of works in the literature within different research areas.

The *Ubiquitous Computing* (UbiComp) proposed by Weiser [13] has a key principle that is to make the computer able to vanish in the background to increase their use making it in an efficient and invisible manner to users. Our CONNECTors fit perfectly the ubiquitous vision where each NS maintains its own characteristics, being able to communicate and cooperate with the others without having any prior knowledge of them thanks to the support provided by the mediators that masks divergencies making them appear homogeneous.

Interoperability and *mediation* have been investigated in several contexts, among which protocol conversion [14–16], integration of heterogeneous data sources [17], software architecture [18], architectural patterns [19], design patterns [20], patterns of connectors [21, 22], Web services [23–27], and algebra to solve mismatches [28] to mention a few.

A lot of work has also been devoted to *connectors* like for example [29–33] to mention few. A work strictly related to the mediators is the seminal work by Yellin and Strom [34]. With respect to our synthesis approach, this work prevents to deal with ordering mismatches and different granularity of the languages (see [5, 6] for a detailed mismatches description). Other works related to our but posing the focus on different problems are [35] and [36].

Stochastic model-based approaches for quantitative analysis of non-functional properties have been largely developed along the last decades and documented in a huge literary production on this topic. The already cited papers [1, 37] 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) [10], a variant of the Stochastic Petri Nets class.

With regard to monitoring, various approaches have been recently proposed. Similarly to GLIMPSE, also [38] 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 [39]. 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.

A prominent part of our framework 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 [40], 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. In [41], 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. 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.

9 Conclusions

We have introduced the ambitious vision of the CONNECT project for an eternally and dependably CONNECTed world. Of the complex CONNECT architecture under development, we have focused here on the Synthesis enabler, which derives on-the-fly a

mediator enabling the functional interoperation among heterogenous NSs; the Dependability&Performance enabler, which applies Stochastic model-based analysis for assessing the desired non-functional properties; and the Monitor, which observes the run-time CONNECTor behaviour. We have discussed on a case study their integrated usage to allow for adaptive analysis accounting for possible inaccurate information or potential evolution of the involved NSs. We refer to a library of adaptation patterns that DEPER suggests to Synthesis to enhance the CONNECTor and make it compliant with the expected non-functional properties. At present, Synthesis uses such suggestion to synthesize a new CONNECTor that can satisfy the non-functional requirements. In future, we will investigate approaches for on-the-fly adaptation of the CONNECTor, where possible.

For reasons of space, we could not cover other important enablers in the CONNECT architecture. Further information can be obtained from the project web site.

Acknowledgements. This work has been partially supported by the European Project CONNECT Grant Agreement No.231167.

References

1. Bondavalli, A., Chiaradonna, S., Giandomenico, F.D.: Model-based evaluation as a support to the design of dependable systems. In: Diab, H.B., Zomaya, A.Y. (eds.) Dependable Computing Systems: Paradigms, Performance Issues, and Applications, pp. 57–86. Wiley (2005)
2. CONNECT Consortium: Deliverable 6.1 – Experiment scenarios, prototypes and report – Iteration 1 (2011)
3. Inverardi, P., Issarny, V., Spalazzese, R.: A Theory of Mediators for Eternal Connectors. In: Margaria, T., Steffen, B. (eds.) ISoLA 2010, Part II. LNCS, vol. 6416, pp. 236–250. Springer, Heidelberg (2010)
4. Spalazzese, R., Inverardi, P., Issarny, V.: Towards a formalization of mediating connectors for on the fly interoperability. In: Proceedings of the WICSA/ECSA 2009, pp. 345–348 (2009)
5. Spalazzese, R., Inverardi, P.: Mediating Connector Patterns for Components Interoperability. In: Babar, M.A., Gorton, I. (eds.) ECSA 2010. LNCS, vol. 6285, pp. 335–343. Springer, Heidelberg (2010)
6. Spalazzese, R., Inverardi, P.: Components interoperability through mediating connector pattern. In: WCSI 2010, arXiv:1010.2337. EPTCS, vol. 37, pp. 27–41 (2010)
7. Bertolino, A., Calabró, A., Di Giandomenico, F., Nostro, N.: Dependability and Performance Assessment of Dynamic CONNECTed Systems. In: Bernardo, M., Issarny, V. (eds.) SFM 2011. LNCS, vol. 6659, pp. 350–392. Springer, Heidelberg (2011)
8. Masci, P., Martinucci, M., Di Giandomenico, F.: Towards automated dependability analysis of dynamically connected systems. In: Proc. IEEE International Symposium on Autonomous Decentralized Systems, Kobe, Japan, pp. 139–146. IEEE (June 2011)
9. Masci, P., Nostro, N., Di Giandomenico, F.: On Enabling Dependability Assurance in Heterogeneous Networks through Automated Model-Based Analysis. In: Troubitsyna, E.A. (ed.) SERENE 2011. LNCS, vol. 6968, pp. 78–92. Springer, Heidelberg (2011)
10. Sanders, W.H., Malhis, L.M.: Dependability evaluation using composed SAN-based reward models. Journal of Parallel and Distributed Computing 15, 238–254 (1992)
11. Daly, D., Deavours, D.D., Doyle, J.M., Webster, P.G., Sanders, W.H.: Möbius: An Extensible Tool for Performance and Dependability Modeling. In: Haverkort, B.R., Bohnenkamp, H.C., Smith, C.U. (eds.) TOOLS 2000. LNCS, vol. 1786, pp. 332–336. Springer, Heidelberg (2000)

12. Drools fusion: Complex event processor, http://www.jboss.org/drools/drools-fusion.html
13. Weiser, M.: Hot Topics: Ubiquitous Computing. IEEE Computer (1993)
14. Calvert, K.L., Lam, S.S.: Formal methods for protocol conversion. IEEE Journal on Selected Areas in Communications 8, 127–142 (1990)
15. Lam, S.S.: Correction to "protocol conversion". IEEE Trans. Software Eng. 14, 1376 (1988)
16. Okumura, K.: A formal protocol conversion method. In: SIGCOMM, pp. 30–37 (1986)
17. Wiederhold, G.: Mediators in the architecture of future information systems. IEEE Computer 25, 38–49 (1992)
18. Garlan, D., Shaw, M.: An introduction to software architecture. Technical Report CMU-CS-94-166, Carnegie Mellon University (1994)
19. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture. A System of Patterns, vol. 1. Wiley, Chichester (1996)
20. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Resusable Object-Oriented Software. Addison-Wesley Professional (1995)
21. Wermelinger, M., Fiadeiro, J.L.: Connectors for mobile programs. IEEE Trans. Softw. Eng. 24, 331–341 (1998)
22. Spitznagel, B.: Compositional Transformation of Software Connectors. PhD thesis, Carnegie Mellon University (2004)
23. Motahari Nezhad, H.R., Xu, G.Y., Benatallah, B.: Protocol-aware matching of web service interfaces for adapter development. In: Proceedings of the 19th International Conference on World Wide Web, WWW 2010, pp. 731–740. ACM, New York (2010)
24. Cimpian, E., Mocan, A.: WSMX Process Mediation Based on Choreographies. In: Bussler, C.J., Haller, A. (eds.) BPM 2005. LNCS, vol. 3812, pp. 130–143. Springer, Heidelberg (2006)
25. Vaculín, R., Sycara, K.: Towards automatic mediation of OWL-S process models. In: IEEE International Conference on Web Services, pp. 1032–1039 (2007)
26. Williams, S.K., Battle, S.A., Cuadrado, J.E.: Protocol Mediation for Adaptation in Semantic Web Services. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 635–649. Springer, Heidelberg (2006)
27. Cavallaro, L., Di Nitto, E., Pradella, M.: An Automatic Approach to Enable Replacement of Conversational Services. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 159–174. Springer, Heidelberg (2009)
28. Dumas, M., Spork, M., Wang, K.: Adapt or Perish: Algebra and Visual Notation for Service Interface Adaptation. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 65–80. Springer, Heidelberg (2006)
29. Spitznagel, B., Garlan, D.: A compositional formalization of connector wrappers. In: ICSE, pp. 374–384 (2003)
30. Fiadeiro, J.L., Lopes, A., Wermelinger, M.: Theory and practice of software architectures. Tutorial at the 16th IEEE Conference on Automated Software Engineering, San Diego, CA, USA, November 26-29 (2001)
31. Lopes, A., Wermelinger, M., Fiadeiro, J.L.: Higher-order architectural connectors. ACM Trans. Softw. Eng. Methodol. 12, 64–104 (2003)
32. Barbosa, M.A., Barbosa, L.S.: Specifying Software Connectors. In: Liu, Z., Araki, K. (eds.) ICTAC 2004. LNCS, vol. 3407, pp. 52–67. Springer, Heidelberg (2005)
33. Bruni, R., Lanese, I., Montanari, U.: A basic algebra of stateless connectors. Theor. Comput. Sci. 366, 98–120 (2006)
34. Yellin, D.M., Strom, R.E.: Protocol specifications and component adaptors. ACM Trans. Program. Lang. Syst. 19 (1997)
35. Tivoli, M., Inverardi, P.: Failure-free coordinators synthesis for component-based architectures. Sci. Comput. Program. 71, 181–212 (2008)

36. Canal, C., Poizat, P., Salaün, G.: Model-based adaptation of behavioral mismatching components. IEEE Trans. Software Eng. 34, 546–563 (2008)
37. Nicol, D.M., Sanders, W.H., Trivedi, K.S.: Model-based evaluation: from dependability to security. IEEE Transactions on Dependable and Secure Computing 1, 48–65 (2004)
38. Pietzuch, P., Shand, B., Bacon, J.: Composite event detection as a generic middleware extension. IEEE Network 18, 44–55 (2004)
39. Hussein, E.A.S., Abdel-wahab, H., Maly, K.: HiFi: A New Monitoring Architecture for Distributed Systems Management. In: Proceedings of ICDCS, pp. 171–178 (1999)
40. Mishra, K., Trivedi, K.S.: Model Based Approach for Autonomic Availability Management. In: Penkler, D., Reitenspiess, M., Tam, F. (eds.) ISAS 2006. LNCS, vol. 4328, pp. 1–16. Springer, Heidelberg (2006)
41. Pietrantuono, R., Russo, S., Trivedi, K.S.: Online monitoring of software system reliability. In: Proc. EDCC 2010 - 2010 European Dependable Computing Conference, pp. 209–218. IEEE Computer Society (2010)