

# Automated Refinement of Dependability Analysis through Monitoring in Dynamically Connected Systems

Antonia Bertolino

Antonello Calabrò

Felicita Di Giandomenico\*

Marco Martinucci

Paolo Masci

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"  
Consiglio Nazionale delle Ricerche  
Pisa, Italy

{ antonia.bertolino, calabro, digiandomenico, martinucci, masci }@isti.cnr.it

## ABSTRACT

Model-based analysis is a well-established method to assess the dependability of a system before deployment. It is well known that, in highly dynamic contexts, the accuracy of the analysis results can be limited because unpredictable phenomena may affect the system during its operation. In such contexts, the analysis typically needs to be refined with data obtained from real system executions. In this paper we tackle the issue of refining model-based dependability analysis in automated systems through monitoring. Specifically, we report on our preliminary results on the development of a system that exploits the synergic use of an automated approach for model-based dependability analysis and a flexible monitoring architecture.

## Keywords

Model-based analysis, Monitoring, On-line refinement.

## 1. INTRODUCTION

Modern software applications are more and more 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 in the unpredictable open-world settings [7]. In such partially unknown and evolving context, dependability analysis [6] calls for on-line support to enhance the accuracy of preliminary estimates performed at design-time.

Analysis at the early stage of a development process is of paramount importance to achieve the required functional and non-functional properties. Nevertheless, the incomplete a priori knowledge about the operating system and environment unavoidably undermines the accuracy of the considered elements and, hence, of the analysis results.

\*Contact author

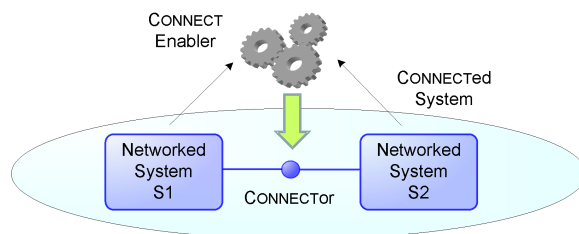


Figure 1: Overview of the CONNECT architecture

In this paper, we tackle the issue of refining model-based dependability analysis in automated systems through monitoring. We focus on the synergic use of an automated approach to perform model-based dependability analysis with a run-time monitoring of real system executions.

We present the work we are carrying out in the context of the European Project CONNECT [5], which considers dynamic environments populated by heterogeneous networked systems willing to communicate.

In CONNECT, communication is seamlessly supported by CONNECT Enablers, which are automated components that make possible the interoperation between heterogeneous systems by synthesising and deploying, at run-time, mediating software bridges, called CONNECTORS (see Figure 1).

Specifically, the CONNECT architecture includes five main Enablers: *Discovery*, which discovers mutually interested Networked Systems (NSs), and retrieves information on the interfaces of NSs; *Learning*, which possibly completes the specifications of the NSs through a learning procedure; *Synthesis*, which performs the dynamic synthesis of mediating CONNECTORS to enable interoperation among NSs willing to interact; *Dependability*, which uses a model-based analysis to support Synthesis in the definition of dependable CONNECTORS; *Monitoring*, which continuously monitors the deployed CONNECTOR to update the CONNECTOR specification used by the other Enablers with run-time data. In this work, we report on the interplay between *Dependability* and *Monitoring*.

The rest of the paper is structured as follows. Section 2 and Section 3 briefly describe the functionalities of the Depend-

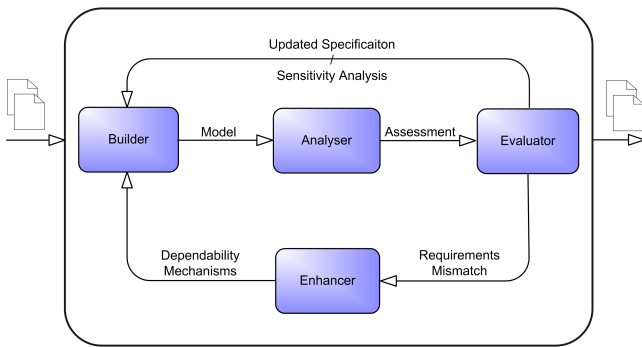


Figure 2: Dependability Enabler Architecture

ability and Monitoring Enablers. Section 4 discusses the interactions between Dependability and Monitoring, putting emphasis on how monitoring is triggered and how feed-backs are used in the dependability analysis. Section 5 shows an example of application, and Section 6 concludes the paper.

## 2. DEPENDABILITY ENABLER

The Dependability Enabler is in charge of performing model-based dependability analysis to assess, before deployment, if the synthesised CONNECTOR is able to satisfy the dependability requirements requested by the NSs.

The Dependability Enabler is logically split into four main functional modules (see Figure 2): *Builder*, *Analyser*, *Evaluator* and *Enhancer*. The Builder module derives the dependability model of the CONNECTED system from the specification provided by Synthesis. The Analyser module uses the generated dependability model to perform a quantitative assessment of the non-functional requirements reported by the Discovery Enabler. The Evaluator module checks the analysis results to determine if the non-functional requirements are met. If the requirements are not satisfied, the Evaluator activates the Enhancer module to determine possible solutions to improve the dependability level of the CONNECTED System. If the requirements are satisfied, the Evaluator reports to Synthesis that the CONNECTOR can be successfully deployed, and reports to the Monitoring Enabler the aspects that must be observed for the CONNECTOR that is going to be deployed, e.g., transition durations and probability of transitions failure.

We realised a prototype implementation, denominated DEA<sup>1</sup>, of the Dependability Enabler based on the Möbius [3] analysis tool; further details on the architecture and the implementation of this Enabler can be found in [12].

## 3. MONITORING ENABLER

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

The Monitoring Enabler used for this contribution, denominated GLIMPSE<sup>2</sup>, has a highly flexible and lightweight ar-

<sup>1</sup>DEpendability Analyser, <http://dcl.isti.cnr.it/dea>

<sup>2</sup>Generic fLexIble Monitoring based on a Publish-Subscribe infrastructure, <http://labse.isti.cnr.it/tools/glimpse>

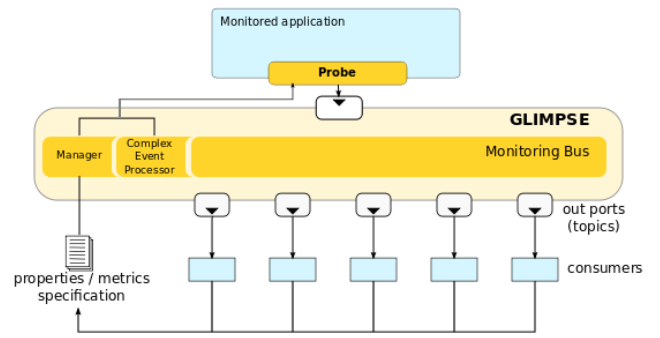


Figure 3: Monitoring Enabler Architecture

chitecture [8] decoupling high-level event specification from the underlying observation and analysis mechanisms, thus yielding the greatest generality and facility of use.

The Monitoring Enabler is driven by the Manager component (see Figure 3). The collection of raw data coming from the observed components is provided by Probes, which may be realised either by injecting code into an existing software, or by using proxies that intercept events to be analysed and send them to the Monitoring Bus. In this context, the Monitoring Enabler injects the Probes into the CONNECTOR to observe transitions. The communication backbone of the Monitoring Enabler is the Monitoring Bus, to which all the information (events, requests, responses) are sent on by: Probes, CONNECT Enablers, Complex Event Processor and by any other services joining GLIMPSE.

The Complex Event Processor, instructed with the information provided by the Dependability Enabler, is implemented using JBoss Drools Fusion [2], a rule engine based on Charles Forgy’s Rete Algorithm [10], able to perform the analysis of the events streaming on the Monitoring Bus. The latter is implemented using ServiceMix4 [4] and ActiveMQ [1] technologies.

## 4. ENABLERS INTERACTIONS

The interactions between Dependability and Monitoring Enablers start when the Dependability Enabler determines that the synthesised CONNECTOR satisfies the required dependability level. Specifically, after the analysis phase, Dependability instructs Monitoring on the CONNECTOR and NSs aspects (among those used in the dependability analysis) that must be observed at run-time. The Monitoring Enabler, upon receiving the request, properly manages the probes embedded in the CONNECTOR. Once the CONNECTOR is deployed, data derived from real executions are sent to Dependability. The Dependability 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, Dependability 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.

With more details, Dependability and Monitoring interact by using a Publish/Subscribe protocol. The interaction pat-

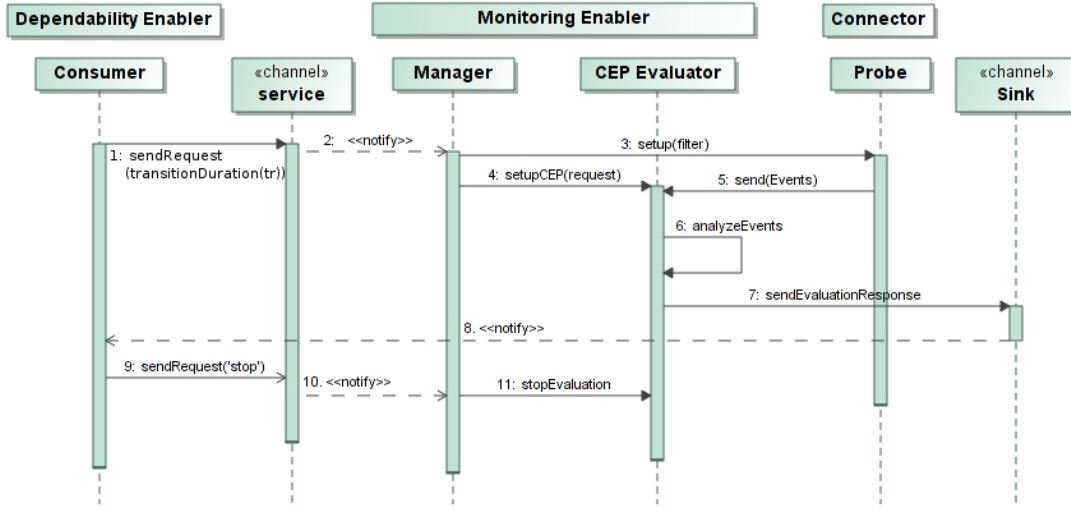


Figure 4: Sequence diagram of the basic interaction pattern between Dependability and Monitoring

Function	Returned value
$transitionDuration(tr)$	time to complete transition $tr$
$stateDuration(s)$	time elapsed in state $s$
$\#(tr, t1, t2)$	number of times transition $tr$ fires in time frame $[t1, t2]$

Table 1: Examples of predefined functions

tern is shown as a sequence diagram in Figure 4, where we intentionally left out system start-up operations. Whenever Monitoring receives a request message on the service channel, a new channel dedicated to the requesting Enabler is set up to communicate monitored values. Monitoring sends response messages to Dependability as soon as the aspect of interest is available. The two Enablers exchange JMS messages whose payload is expressed in XML language. Each payload contains, among other fields, a `connectorID` field, which uniquely identifies the `CONNECTOR` specification. The value of `connectorID` is provided by the Synthesis Enabler along with the `CONNECTOR` specification.

Each *request message* received by the Monitoring Enabler contains the following fields: (i) `aspect`, that specifies the aspect to be monitored; this field contains the identifier of a function, which is taken from an ontology shared between the Enablers (see Table 1 for some samples of functions), and the function parameters; (ii) an optional field `timeFrame`, that specifies the monitoring time frame; if this field is not included, the monitoring activity continues as long as the Dependability Enabler does not send a request to stop the observations.

Each *response message* sent by the Monitoring Enabler contains a `instanceID` field, which uniquely determines a `CONNECTOR` instance. The information on the `CONNECTOR` instance is provided to the Monitoring Enabler by the Probe whenever the `CONNECTOR` is activated, and is useful to perform instance-based statistical analysis.

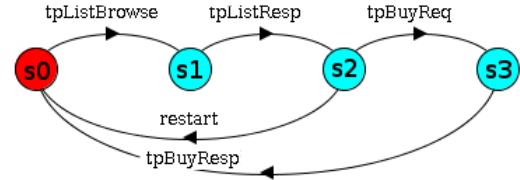


Figure 5: LTS of the Consumer

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <request connectorID="xyz">
3   <aspect id = "transitionDuration"
4     action = "start" >
5     <transition startState = "s0"
6       label = "tpListBrowse"
7       endState = "s1" />
8   <transition startState = "s2"
9     label = "tpBuyReq"
10    endState = "s3" />
11 </aspect>
12 </request>

```

Listing 1: Sample request from Dependability

## 5. APPLICATION EXAMPLE

In the following example, we focus on the Enablers interactions only, leaving out the actions taken by Dependability once obtained the values from Monitoring. To show a basic interaction between Dependability and Monitoring Enablers, we refer to the Distributed Market Scenario [9]. The scenario considers a distributed market, where consumers execute a discovery protocol to gather information on the products sold by merchants. It is assumed that consumers and merchants use different communication protocols. With reference to recent works on synthesis of mediating `CONNECTORS` [14] and automata discovery/learning [13], the specification of the `CONNECTED` system is given with Labelled Transition Systems (LTSs) [11]. Specifically, consumers use the protocol represented by the LTS shown in Figure 5.

---

```

1 rule "transitionDuration"
2   salience 0
3   dialect "java"
4   when
5     $aEvent : SimpleEvent( this.data == "s0"
6       , this.getSourceID == "xyz");
7     $bEvent : SimpleEvent( this after
8       $aEvent, this.getSourceID == $aEvent
9       .getSourceID);
10  then
11    sendResponse(EvaluateTimeStamps($aEvent.
12      getTimestamp(), $bEvent.getTimestamp
13      ()))));
14    retract($aEvent);
15    retract($bEvent);
16 end

```

---

**Listing 2: Sample rule for checking a duration policy**

---

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <response connectorID = "xyz"
3   instanceID = "abc" >
4   <aspect id = "transitionDuration" >
5     <transition startState = "s0"
6       label = "tpListBrowse"
7       endState = "s1" />
8     <double>1.0</double>
9   </transition>
10 </aspect>
11 </response>

```

---

**Listing 3: Sample response from Monitoring**

We consider the case in which the parameters under monitoring are the duration of the transitions executed by the NS requesting the communication, because the CONNECTOR specification contains time-outs to limit the duration of the wait periods for the CONNECTOR. Hence, with reference to Figure 5, the parameters to be monitored are the consumer transitions `tpListBrowse` and `tpBuyReq`. The request messages sent by Dependability to Monitoring are shown in Listing 1. The predefined function `transitionDuration(tr)` is mapped into an XML `aspect` element with `id` attribute equal to `transitionDuration`. This operation generates a Drools rule, shown in Listing 2, that will be executed until the Dependability Enabler sends a stop request. An example of response message sent by the Monitoring Enabler is shown in Listing 3.

## 6. CONCLUSIONS

This paper has shown work-in-progress on the synergic use of stochastic model-based dependability analysis, conducted at system design-time, with run-time monitoring to improve the accuracy of estimates of dependability properties. The reference framework is that of the European project CONNECT, which investigates solutions to interoperability in heterogeneous, dynamic environment. The basic interplay between Dependability and Monitoring Enablers has been discussed, highlighting the benefits of the monitoring feed-backs on the model-based analysis. A simple application example has been also presented, which shows the operative steps of the Enablers interactions.

Future work includes: (i) to finalise the definition of the

statistical analysis on observed data, (ii) to set-up an appropriate data model to store the monitored data, (iii) to extend the monitoring approach in order to check dependability properties and to validate the correctness of the specifications of both CONNECTOR and NSs. All these points are in our current and future research agenda.

## ACKNOWLEDGEMENTS

This work is partially supported by the EU FP7 Project CONNECT (FP7-231167).

## 7. REFERENCES

- [1] Activemq: A complete message broker. <http://activemq.apache.org>.
- [2] Drools fusion: Complex event processor. <http://www.jboss.org/drools/drools-fusion.html>.
- [3] Möbius tool. <http://www.mobius.illinois.edu/>.
- [4] Servicemix: an open source esb. <http://servicemix.apache.org/home.html>.
- [5] CONNECT: Emergent CONNECTORS for Eternal Software Intensive Networked Systems. <http://connect-forever.eu/>, 2009–2013.
- [6] A. Avizienis, J.C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1:11–33, 2004.
- [7] L. Baresi, E. Di Nitto, and C. Ghezzi. Toward open-world software: Issue and challenges. *Computer*, 39:36–43, 2006.
- [8] A. Bertolino, A. Calabrò, F. Lonetti, and A. Sabetta. GLIMPSE: A generic and flexible monitoring infrastructure. Technical report, ISTI-CNR, 2010. 2010-TR-024.
- [9] F. Di Giandomenico, M. Kwiatkowska, M. Martinucci, P. Masci, and H. Qu. Dependability analysis and verification for CONNECTED systems. In *ISOLA2010*, volume 6416 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 2010.
- [10] C. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligences*, 19(1):17–37, 1982.
- [11] R. M. Keller. Formal verification of parallel programs. *Communications of the ACM*, 19:371–384, July 1976.
- [12] P. Masci, M. Martinucci, and F. Di Giandomenico. Towards automated dependability analysis of dynamically CONNECTED systems. In *ISADS2011*, 2011. To appear.
- [13] H. Raffelt, B. Steffen, and T. Berg. Learnlib: a library for automata learning and experimentation. In *FMICS’05*, pages 62–71, New York, NY, USA, 2005. ACM.
- [14] R. Spalazzese and P. Inverardi. Mediating CONNECTOR Patterns for Components Interoperability. In *ECSA2010*, LNCS, pages 335–343. Springer, 2010.