On enabling dependability assurance in heterogeneous networks through automated model-based analysis

Paolo Masci, Nicola Nostro, Felicita Di Giandomenico

On enabling dependability assurance in heterogeneous networks through automated model-based analysis

Paolo Masci[1]*, Nicola Nostro[2], and Felicita Di Giandomenico[2]

[1] Queen Mary University of London, UK
paolo.masci@eecs.qmul.ac.uk

[2] ISTI-CNR, Pisa, Italy,
{nicola.nostro,felicita.digiandomenico}@isti.cnr.it

Abstract. We present the specification of a basic library of dependability mechanisms that can be used within automated approaches for synthesising dependable CONNECTors in heterogeneous networks. The library builds on classical dependability patterns, such as majority voting and retry, and uses the concept of overlay networks for triggering the synthesis of specific dependability mechanisms in the CONNECTor from high-level specifications. We translated such dependability mechanisms into SAN models with the aim to evaluate, through model-based analysis, which dependability mechanisms should be embedded in the synthesised CONNECTor for ensuring a given dependability level between networked systems willing to be connected. A case study is also presented to show the application of selected library mechanisms. This work is carried out in the context of CONNECT, a European FET project which is investigating the possibility of enabling long-lasting inter-operation among networked systems by synthesising mediating CONNECTors at run-time.

1 Introduction and background

Interoperability in future networks will be characterised by seamless and continuous communication among heterogeneous networked systems. Over time, networked systems may change mode of operation, e.g., because of hardware/software updates or new application contexts. As a consequence, network infrastructures ought to provide appropriate means for supporting interoperability among evolving networked systems.

In the European FET project CONNECT [1], interoperability issues of evolving networks are tackled by synthesising dependable CONNECTors at run-time. To do this, the network infrastructure defined in CONNECT embeds five logical units, denominated *enablers*, that seamlessly collaborate for ensuring continuous and long-lasting inter-operation among networked systems. In CONNECT, the *discovery enabler* discovers the functionality of networked systems and applications and retrieves information on the interfaces they use for inter-operating

* Corresponding author.

with others. The *learning enabler* completes such a knowledge on the interaction behaviour of networked systems by applying learning algorithms, and produces a model of this behaviour in the form of a labelled transition system (LTS). The *synthesis enabler* dynamically synthesises a software mediator using code generation techniques (from the independent LTS models of each system) that will connect and coordinate the interoperability between heterogeneous systems. In order to fulfil dependability requirements, *synthesis* triggers the *dependability enabler*, which is in charge of analysing the CONNECTor's design before the CONNECTor gets deployed and put in operation. If needed, the dependability enabler drives the synthesis enabler towards possible CONNECTor's enhancement. The *monitoring enabler* continuously monitors the deployed CONNECTors during their execution for updating the other enablers with run-time data.

In this work, we focus on the dependability enabler, which performs a model-based analysis for assessing the dependability level of the synthesised CONNECTors. Specifically, we point our attention on a dependability enabler's module, denominated *enhancer*. Such a module is responsible for guiding the synthesis process towards enhancements of a CONNECTor's design whenever the analysis reveals inadequate dependability levels. In brief, this module is in charge of selecting a combination of dependability mechanisms suitable for enhancing the synthesised CONNECTor so that it complies with given requirements. Then, the *synthesis enabler* embeds the selected dependability mechanisms in the CONNECTor's design and proceeds with its implementation and deployment. The architecture of the dependability enabler and the functionalities of the enhancer have been presented in [13]. The contribution of this paper consists in the definition of a basic library of dependability mechanisms for the enhancer. The library builds on classical dependability patterns (see [18] for a survey), and uses the concept of *overlay networks* for triggering the synthesis of specific dependability mechanisms in the synthesised CONNECTor from high-level specifications. How these dependability mechanisms can then be embedded in the synthesised connectors pertains to the synthesis enabler and is not addressed in this paper.

The paper is organised as follows. In Section 2, the stochastic activity networks [15] (SAN) formalism, a widely used formalism for model-based dependability analysis of complex systems, is briefly illustrated in order to allow the reader to understand the formal specification of the developed dependability mechanisms. In Section 3, we explain the ideas underpinning the library of dependability mechanisms, and we present the specification of such a library with the SAN formalism. In Section 4, we trial our ideas by applying the library to a case study based on a demonstrative scenario based on that presented in [8]. In Section 5, we report on related work and conclude the paper.

2 The SAN Formalism

Stochastic Activity Networks [15, 14, 21] are an extension of the Petri Nets (PN) formalism [17, 16]. SANs are directed graphs with four disjoint sets of nodes: *places*, *input gates*, *output gates*, and *activities*.

Activities replace and extend the *transitions* of the PN formalism. Each SAN activity may be either *instantaneous* or *timed*. Timed activities represent actions with a duration affecting the performance of the modelled system, e.g., message transmission time. The duration of each timed activity is expressed via a *time distribution* function. Any instantaneous or timed activity may have mutually exclusive outcomes, called *cases*, chosen probabilistically according to the *case distribution* of the activity. Cases can be used to model probabilistic behaviours. An activity *completes* when its (possibly instantaneous) execution terminates.

As in PNs, the state of a SAN is defined by its *marking*, i.e., a function that, at each step of the net's evolution, maps the places to non-negative integers (called the *number of tokens* of the place). SANs enable the user to specify any desired enabling condition and firing rule for each activity. This is accomplished by associating an *enabling predicate* and an *input function* to each input gate, and an *output function* to each output gate. The enabling predicate is a Boolean function of the marking of the gate's input places. The input and output functions compute the next marking of the input and output places. If these predicates and functions are not specified for some activity, the standard PN rules are assumed. The evolution of a SAN, starting from a given marking μ, may be described as follows:

1. the instantaneous activities enabled in μ complete in some unspecified order;
2. if no instantaneous activities are enabled in μ, the enabled (timed) activities become *active*;
3. the completion times of each active (timed) activity are computed stochastically, according to the respective time distributions; the activity with the earliest completion time is selected for completion;
4. when an activity (timed or not) completes, one of its cases is selected according to the case distribution, and the next marking μ' is computed by evaluating the input and output functions;
5. if an activity that was active in μ is no longer enabled in μ', it is removed from the set of active activities.

Graphically, places are drawn as circles, input gates as left-pointing triangles, output gates as right-pointing triangles, instantaneous activities as narrow vertical bars, and timed activities as thick vertical bars. Cases are drawn as small circles on the right side of activities.

3 Library of dependability mechanisms

Overlay networks are virtual networks built on top of existing network substrates: nodes in overlay networks represent logical hosts involved in interactions, and links in overlay networks correspond to paths in the network substrate that are traversed by messages during inter-operations. To date, overlay networks have been used for exploiting peculiar characteristics of network substrates at the application level; for instance, Andersen et al [3] exploit highly redundant

networks substrates for defining resilient communication systems as overlay networks (a survey on the use of overlay networks for defining new applications can be found in [12]).

Thanks to the infrastructure provided by CONNECT, here we can use the concept of overlay networks in an alternative way, i.e., rather than using overlay networks for exploiting the characteristics of the network substrate, we use them for *defining* the characteristics of the network substrate that should be synthesised. The basic idea is to view CONNECTors as overlay networks, and to exploit their structure for triggering the generation of specific dependability mechanisms in the network substrate during the synthesis process.

In the following, we describe the models we defined for triggering the generation of typical dependability mechanisms suitable to contrast two typical classes of failure modes that may happen during interactions: *timing failures*, in which networked systems send messages at time instants that do not match an agreed schedule, and *value failures*, in which networked systems send messages containing incorrect information items. For the purpose of this paper, we consider timing failures of type omission, i.e., late messages are always discarded, and value failures that cause a networked system to respond within the correct time interval but with an incorrect value. Each model is specified with the SAN formalism. The models are developed according to three basic rules that allow to simplify the automated procedure for embedding the mechanism in the specification of the synthesised CONNECTor: (i) each model has an initial place, `s0`, whose tokens enable the first activity of the model; (ii) each model has a final place, `s1`, which contains tokens whenever the last activity of the model completes; (iii) the overall number of tokens in `s1` is always less or equal to the number of tokens in `s0`. With the above rules, the behaviour of the model can be seen as an *enhanced activity*, and can be directly used to replace any activity that moves tokens between two places in the specification of the CONNECTor (the basic semantics of an activity is always preserved).

3.1 Retry Mechanism

The retry mechanism consists in re-sending messages that get corrupted or lost during communications, e.g., due to transient failures of communication links. This mechanism is widely adopted in communication protocols, such as TCP/IP [6] 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 synthesis of a retry mechanism can be triggered with the stochastic activity network shown in Figure 1. On the sender side, the mechanism creates a message re-transmission policy for re-sending the message at most N times; on the receiver side, the mechanism creates a policy for avoiding duplicated reception of messages and for sending acknowledgements.

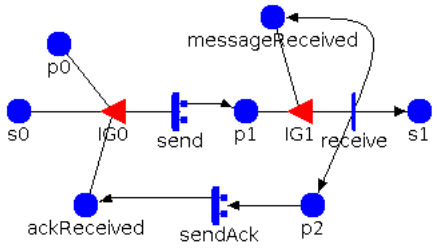


Fig. 1. Retry mechanism

In the model, all places initially contain zero tokens, except `p0`, which contains N tokens, where N is a model parameter representing the maximum number of re-transmissions. Activity `send` is enabled when the conjunction of the following conditions is true: `p0` and `s0` contain at least one token, and `ackReceived` contains zero tokens. When activity `send` completes with success (case 0, with probability $pr0$), a token is removed from `s0` and `p0`, and the marking of `p1` is incremented by one. Activity `receive` is enabled when `p1` contains at least one token and `messageReceived` contains zero tokens. When activity `receive` completes, a token is moved to `s1`, and the marking of `p2` and `messageReceived` is incremented by one. A token in `p2` enables activity `sendAck`, whose aim is to enable the receiving host notify the sender that the message has been successfully received. The sender stops re-transmitting the message as soon as it gets an acknowledgement that the message has been successfully received, or after N attempts.

3.2 Probing Mechanism

The probing mechanism exploits redundant paths and periodic *keep-alive* messages for enabling reliable communication in face of path failures. The basic idea is to continuously collect statistics on the characteristics of the communication channels, and to select the best channel on the basis of such statistics. This mechanism has been used for defining communication services with guaranteed delivery and performance levels, e.g., see Akamai's SureRoute [2] and reliable multi-cast protocols for peer-to-peer networks [23].

The synthesis of a probing mechanism that uses two redundant communication channels can be triggered with the stochastic activity network shown in Figure 2. The mechanism instruments the sender with a periodic channel probing functionality suitable to feed a monitoring system that collects statistics about the reliability level of the communication channels.

In the model, place `mode` is a state variable that indicates the mode of operation of the mechanism, which can be either *probing mode* (`mode` contains zero tokens), i.e., the mechanism tests the characteristics of the communication channels through keep-alive messages, or *normal mode* (`mode` contains one token), i.e.,

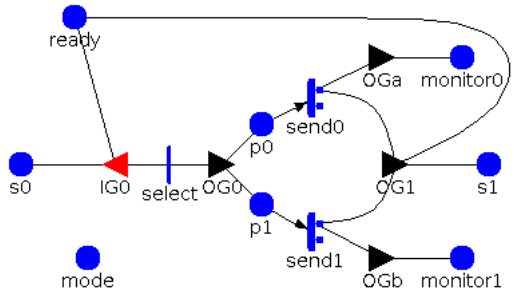


Fig. 2. Probing mechanism

the mechanism selects the best estimated channel for relaying messages. Initially, all places contain zero tokens, except `ready`, which contains one token.

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

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

3.3 Majority Voting Mechanism

Majority voting is a fault-tolerant mechanism that relies on a decentralised voting system for checking the consistency of data. Voters are software systems that constantly check each other's results, and has been widely used for developing resilient systems in the presence of faulty components. In a network, voting systems can be used to compare message replicas transmitted over different channels, see, for instance, the protocol proposed in [24] for time-critical applications in acoustic sensor networks.

The synthesis of a majority voting mechanism that uses three redundant communication channels can be triggered with the stochastic activity network shown in Figure 3. The mechanism replicates the message sent by the transmitting host over three channels. In this case, the mechanism is able to tolerate one faulty channel.

In the model, all places initially contain zero tokens. Activity `multipathRouter` gets enabled when `s0` contains a token. When such an activity completes,

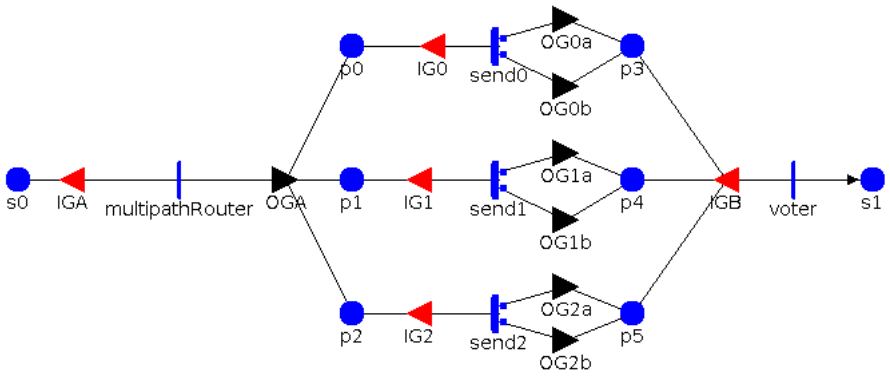


Fig. 3. Majority voting mechanism

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

3.4 Error Correction Mechanism

Error correction deals with the detection of errors and re-construction of the original, error-free data. A widely used approach for enabling hosts to automatically detect and correct errors in received messages is *forward error correction* (FEC). The mechanism requires the sender host to transmit a small amount of additional data along with the message. The mechanism has been used, for instance, in [22] for defining an overlay-based architecture for enhancing the quality of service of best-effort services over the Internet.

The synthesis of an error correction mechanism that uses two redundant communication channels can be triggered with the stochastic activity network shown in Figure 4. One channel is used to send the original message, and the other channel is used to send the error correction (EC) code. The receiver is instrumented with a filtering mechanism that checks and corrects messages.

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

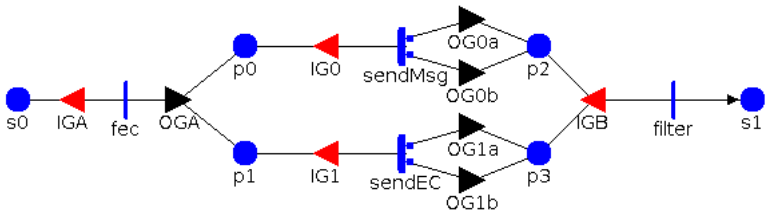


Fig. 4. Error correction mechanism

tokens. When activity `filter` completes, a token is moved into `s1` and all tokens in other places are removed.

3.5 Security Mechanism

A typical way to enforce protection on a host is to decouple the host from the rest of the network. A host can, for instance, be protected from receiving unwanted traffic by creating a ring that selectively filters the incoming traffic. Similarly, the identity of a host can be protected by anonymising the host's messages through a set of intermediary hosts, denominated *proxies*. This mechanism has been used, for instance, in [11] and [4] for protecting hosts from denial-of-service attacks.

The synthesis of a security mechanism over a network with two intermediary hosts can be triggered with the stochastic activity network shown in Figure 5. The mechanism creates an anonymiser service that selects a channel with a certain probability, and forwards the message on such a channel. In the model,

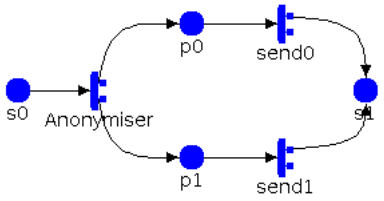


Fig. 5. Security mechanism

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

4 Case study

In this section, we show how the developed library can be used within an automated model-based dependability analysis with the aim to enhance a synthesised CONNECTor. We consider a demonstrative scenario described in [8], where two kinds of heterogeneous devices need to communicate in a reliable and timely manner. For clarity of exposition, in order to fit the purpose of this paper and also make it self-contained, in the following we report a concise and slightly reworded description of the scenario reported in [8], and an informal specification of the protocols used by the two kinds of heterogeneous devices, and of the synthesised CONNECTor. Readers interested in the complete original specification are re-directed to [8].

4.1 Specification

The scenario considers an emergency situation in which policemen need to exchange information with security guards. Each policeman can exchange confidential data with other policemen with a *secured file sharing* protocol. Security guards, on the other hand, exchange information by using another protocol, denominated *emergency call*. The two protocols have the same aim (i.e., enable information exchange) but a mediating CONNECTor is needed in order to enable inter-operation, because they use different message types and different message sequences.

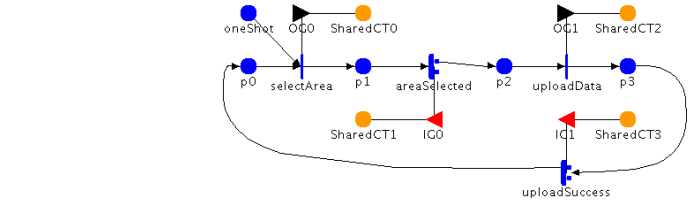
Secured File Sharing. This is a basic peer-to-peer protocol for enabling file sharing. The peer that initiates the communication, denominated *coordinator*, sends a multicast message (`selectArea`) to a selected group of peers. When a peer receives a `selectArea` message, the peer replies to the coordinator with a `areaSelected` message. Upon receiving the `areaSelected` message, the coordinator sends a data file to the peers (`uploadData` message) that, in turn, automatically reply with an `uploadSuccess` message if the data has been successfully received.

Emergency Call. This is a peer-to-peer protocol for sending data files from a control centre to groups of devices. Each group of devices is coordinated by a leader. The protocols is initiated by the control centre, which sends an `eReq` message to a group of devices located in a selected area of interest. The group leader is in charge of replying to the control centre with an `eResp`. Whenever the control centre receives the `eResp` from a group leader, an `emergencyAlert` message is sent to all devices. Each device automatically notifies the control center with an `eACK` message whenever it successfully receives the data.

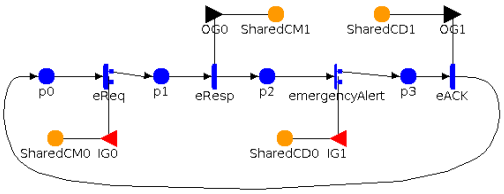
Mediating Connector. A mediating CONNECTor suitable for enabling inter-operation from devices using the secured file sharing protocol to devices using the emergency call protocol performs the following translations: `selectArea` messages are translated into `eReq` messages directed to the leaders of selected

group of devices; `eResp` messages are translated into `areaSelected` messages; `uploadData` messages are translated into multicast `emergencyAlert` messages; `eACK` messages are collected by the CONNECTor and then translated into a single `uploadSuccess` message. A timeout is used to avoid infinite wait in the case of failure of `eACK` messages.

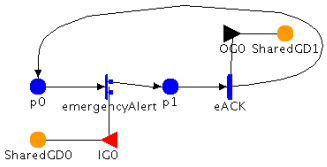
4.2 SAN Models



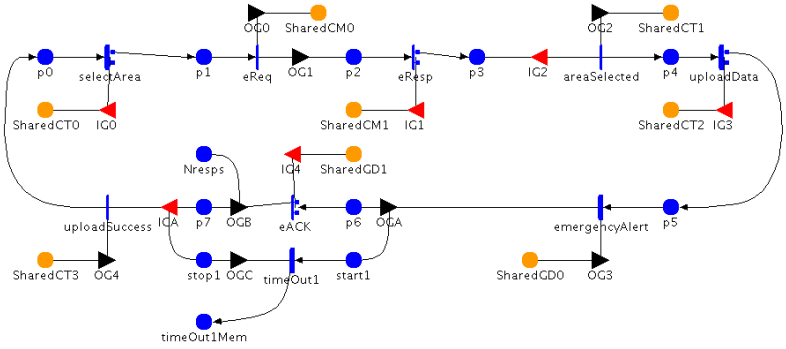
(a) SecuredFileSharing



(b) EmergencyCall, commander



(c) EmergencyCall, guard



(d) CONNECTor

Fig. 6. SAN Models

The SAN models corresponding to the specifications are shown in Figure 6. The CONNECTor uses two different channels to communicate with the two different kinds of devices. Any send and receive action performed by the CONNECTor

is represented by a timed activity. Each activity has two case probabilities: case 0 is associated to the correct behaviour; case 1 is associated to incorrect behaviour. Since the purpose of this case study is not to show new results on a real-world protocol, but to exemplify the utility of the developed library, here we assume that timed activities are all exponentially distributed and with the same rate.

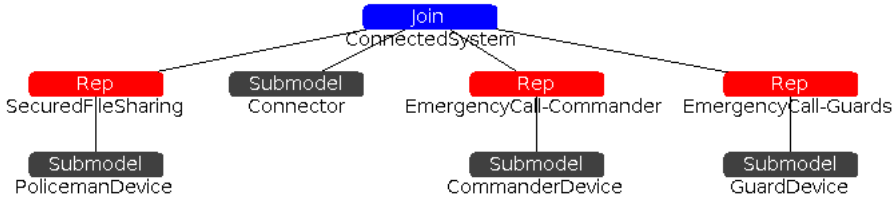


Fig. 7. SAN model of the CONNECTed system

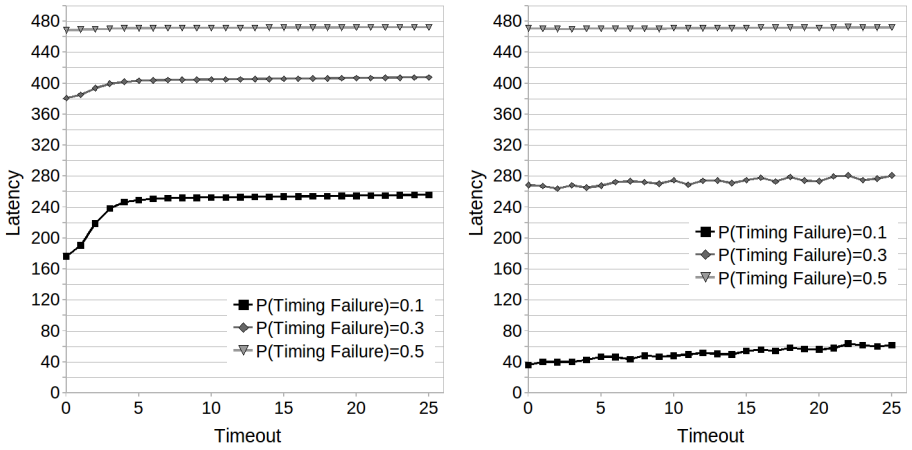
The model of the CONNECTed system, which is shown in Figure 7, is obtained by composing the models through place sharing. In the CONNECTed system, there is a shared place for each pair of activities that represent send/receive actions: send activities add tokens in the shared place, while receive activities remove tokens from the shared place and use the marking of the shared place as enabling condition.

4.3 Analysis

The analysis is performed through Möbius [10], and consists in a measure of latency and a measure of coverage. Latency is measured from when the control centre sends the initial request `selectArea` to when it receives `uploadSuccess`. Coverage is given by the percentage of responses the control centre receives back within a certain time T.

The analysis we describe can be automated with the approach reported in [13]. In order to simplify the exposition, here we consider only failures between the CONNECTor and the guards' devices (which execute the Emergency Call protocol), and we use the probing mechanism to contrast timing failures, and the majority voting mechanism to contrast value failure. Both mechanisms are introduced on the communication channel between the CONNECTor and the guards' devices (which follow the Emergency Call protocol).

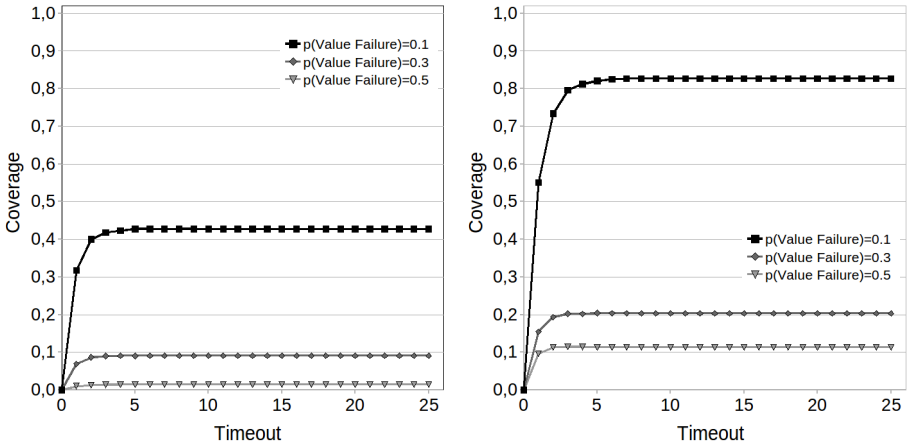
Latency. The first analysis aims to assess the trend of latency for different values of timeout, assuming three different values of timing failure probability between the CONNECTor and the guards. Figure 8(a), shows the value of latency (on the y axis) for the CONNECTed system without dependability mechanisms (the timeout value is reported on the x axis). Figure 8(b), shows the same analysis performed on the model enhanced with the probing mechanism. We can notice that, with the considered system parameters, the mechanism is able to



(a) Basic model

(b) Enhanced model with Probing Mechanism

Fig. 8. Latency assessment in case of timing failure



(a) Basic model

(b) Enhanced model with Majority Voting Mechanism

Fig. 9. Coverage assessment in case of value failure

reduce latency only in two out of three situations. When the timing failure probability is 0.5 the latency has very similar values for both models; in fact, in such a case the failure probability has a too high value and the probing mechanism is a too light means to contrast its effects.

Coverage. The second analysis is performed for three different probabilities of value failures between the CONNECTor and the guards. Figures 9(a) and 9(b) show the analysis results for the basic model and for the model enhanced with the majority voting mechanism. In this case, the mechanism is able to improve coverage in all considered cases. We can notice that, for the probability values considered, the coverage provided by the enhanced model is approximately double compared to basic one.

5 Related work and conclusions

Automated dependability analysis, pursued through transformation-based verification and validation environments, has been the subject of several studies in the last decade. Automatic/automated methods from system specification languages to modelling languages amenable to perform dependability analysis has been recognised as an important support for improving the quality of systems. Moreover, it favours the application of verification and validation techniques at industry level, where these methods have difficulties to be applied on a routine basis, primarily due to the high level of expertise required to deal with mathematical modelling and analysis techniques. Development of an integrated environment to support the early phases of system design, where design tools based on the UML (Unified Modeling Language) are augmented with transformation-based validation and analysis techniques, is presented in [5], among several other works. A Modeling framework allowing the generation of dependability-oriented analytical models from AADL (Architecture Analysis and Design Language) models is described in [19]. Tools have also been developed, supporting the definition of model-based transformations. To provide some examples, the Viatra tool [9] automatically checks consistency, completeness, and dependability requirements of systems designed using the Unified Modeling Language. The Genet tool [7] allows the derivation of a general Petri net from a state-based representation of a system. The ADAPT Tool supports model transformations from AADL Architectural Models to Stochastic Petri Nets [20]. However, from the point of view of enhancing the model-transformation environment with template models of basic fault tolerance mechanisms to allow automated assessment of enhanced, fault tolerant designs, it appears a rather novel research direction. Although studies exist dealing with libraries of fault tolerance mechanisms (e.g. [18]) to assist the design of dependable systems, to the best of the authors' knowledge the attempt to provide template models of dependability mechanisms, to be incorporated in a wider system dependability model to assess their efficacy at system design time, is a new contribution of this paper.

In this paper, five dependability mechanisms have been specified in terms of SAN models, covering basic means to cope with timing and value failures of communication channels in heterogeneous networked systems. They are first encapsulated in the dependability model of the CONNECTor set-up to allow interoperability among networked systems, and managed by the dependability evaluator enabler to analyse their appropriateness to satisfy dependability properties required by the networked systems. Upon positive assessment, they are employed to build advanced CONNECTors design. A case study has been also included, inspired by current research activity ongoing in the context of the EU CONNECT project, to show the practical application of selected dependability mechanisms in presence of failure scenarios.

The work described is a first step in the development of enhanced automated dependability analysis as a support for the synthesis of dependable CONNECTors. After the definition of the individual dependability mechanisms, all the implications related with their systematic usage to replace basic elements of the CONNECTor dependability model (showing unsatisfactory from the dependability or performance point of view), have to be rigorously considered and solved. Of course, also investigations on further dependability mechanisms suitable to the addressed context would be interesting to carry-on. Indeed, these are among the directions we are exploring as future work.

Acknowledgements

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

References

1. CONNECT: Emergent CONNECTors for Eternal Software Intensive Networked Systems. http://connect-forever.eu/, 2009–2013.
2. Akamai Technologies, Inc. Akamai sureroute for failover and performance, 2003.
3. David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Resilient overlay networks. *SIGOPS Oper. Syst. Rev.*, 35:131–145, October 2001.
4. David G. Andersen. Mayday: Distributed Filtering for Internet Services. In *4th Usenix Symposium on Internet Technologies and Systems*, Seattle, WA, March 2003.
5. Andrea Bondavalli, Mario Dal Cin, Diego Latella, István Majzik, Andràs Pataricza, and Giancarlo Savoia. Dependability analysis in the early phases of uml-based system design. *Language*, 16(5):265–275, 2001.
6. R. T. Braden. RFC 1122: Requirements for Internet hosts — communication layers, October 1989.
7. 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.
8. CONNECT Consortium. Deliverable D5.2 – Dependability Assurance *(available soon)*, 2011.

9. Gyorgy Csertan, Gabor Huszerl, Istvan Majzik, Zsigmond Pap, Andras Pataricza, Daniel Varro, and Dániel Varró. 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.
10. D. Daly, D. D. Deavours, J. M. Doyle, P. G. Webster, and W. H. Sanders. Möbius: An extensible tool for performance and dependability modeling. In B. R. Haverkort, H. C. Bohnenkamp, and C. U. Smith, editors, *11th Int. Conf., TOOLS 2000*, volume 1786 of *LNCS*, pages 332–336. Springer Verlag, 2000.
11. Angelos D. Keromytis, Vishal Misra, and Dan Rubenstein. Sos: Secure overlay services. In *In Proceedings of ACM SIGCOMM*, pages 61–72, 2002.
12. Jinu Kurian and Kamil Sarac. A survey on the design, applications, and enhancements of application-layer overlay networks. *ACM Comput. Surv.*, 43:5:1–5:34, December 2010.
13. P. Masci, M. Martinucci, and F. Di Giandomenico. Towards automated dependability analysis of dynamically connected systems. In *Proc 10th Intl. Symposium On Autonomous Decentralised Systems (ISADS2011)*, pages –, 2011.
14. Ali Movaghar. Stochastic activity networks: a new definition and some properties. *Scientia Iranica*, 8(4):303–311, 2001.
15. Ali Movaghar and J.F. Meyer. Performability modelling with stochastic activity networks. In *Proc. of the 1984 Real-Time Systems Symposium*, pages 215–224, 1984.
16. Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.
17. Carl Adam Petri. *Kommunikation mit Automaten*. PhD thesis, Technische Hochschule Darmstadt, 1961.
18. ReSIST Consortium. EU project ReSIST: Resilience for Survivability in IST. Deliverable D33: Resilience-explicit computing. Technical report, 2008. http://www.resist-noe.org/.
19. Ana-Elena Rugina, Karama Kanoun, and Mohamed Kaâniche. A system dependability modeling framework using aadl and gspns. *Architecting dependable systems*, 4615:14–38, 2007.
20. Ana-Elena Rugina, Karama Kanoun, and Mohamed Kaaniche. The adapt tool: From aadl architectural models to stochastic petri nets through model transformation. *Seventh European Dependable Computing Conference*, 0:85–90, 2008.
21. William H. Sanders and John F. Meyer. Stochastic activity networks: formal definitions and concepts. In *Lectures on formal methods and performance analysis: first EEF/Euro summer school on trends in computer science*, pages 315–343. Springer-Verlag New York, Inc., New York, NY, USA, 2002.
22. Lakshminarayanan Subramanian, Ion Stoica, Hari Balakrishnan, and Randy H. Katz. Overqos: offering internet qos using overlays. *SIGCOMM Comput. Commun. Rev.*, 33:11–16, January 2003.
23. Wenjun Zeng, Yingnan Zhu, Haibin Lu, and Xinhua Zhuang. Path-diversity p2p overlay retransmission for reliable ip-multicast. *Multimedia, IEEE Transactions on*, 11(5):960–971, aug. 2009.
24. Zhong Zhou, Zheng Peng, Jun-Hong Cui, and Zhijie Shi. Efficient multipath communication for time-critical applications in underwater acoustic sensor networks. *IEEE/ACM Trans. Netw.*, 19:28–41, February 2011.