Towards Automated Dependability Analysis of Dynamically Connected Systems

Paolo Masci, Marco Martinucci and Felicita Di Giandomenico
Information Science and Technologies Institute
Italian National Research Council - Pisa, Italy
e-mail: {masci, martinucci, digiandomenico}@isti.cnr.it

Abstract

Dynamic environments may include autonomous and decentralised components that pose many challenges from the point of view of interoperability, thus triggering research studies in several directions. One of these challenges is the investigation of the automatic composition of heterogeneous systems willing to communicate, by synthesising at run-time the connectors that allow interoperation. Besides functional properties, synthesised connectors generally need to satisfy also non-functional (dependability-related) properties. This paper investigates the definition of an automated procedure to support the synthesis of dependable connectors.

Index Terms

Stochastic modelling, autonomous systems interoperation, automated dependability analysis.

I. Introduction

Ubiquitous computing refers to a vision of technology where devices are seamlessly integrated in the environment and everyone benefits from their services without even being aware of their presence [1]. In such a vision, complex and pervasive systems are expected to be composed of autonomous and decentralised components that cooperate or require services on the basis of specific needs. Pervasive systems may evolve along time, rely on heterogeneous communication protocols, and dynamically establish communication at run-time. The fast pace at which technology evolves continuously undermines the effectiveness of such systems because ubiquitous systems components need to exchange information, but this ability is generally linked to the level of interoperability of their underlying technologies.

Among the current studies addressing the issue of interoperability of the next generation systems, the European Project CONNECT [2] targets the dynamic synthesis of connectors via which networked systems communicate. The resulting emergent connectors compose and adapt interaction protocols run by the networked systems. The dynamic synthesis of connectors relies on activities related to discovery, learning, and dependability [3] analysis: interaction protocols are, in general, not known a priori, and both functional and non-functional (dependability-related) properties must be taken into account to synthesise a proper connector that enables successful interactions among networked systems.

In this work, we report on the research we are undertaking towards the development of a Dependability unit that implements an automated dependability analysis to support the synthesis of dependable connectors. The Dependability unit employs a stochastic model-based analysis, which shows appropriate to support design decisions during connector synthesis. We describe the logical architecture of a Dependability unit, and a prototype implementation of the modules composing its architecture. In order to exemplify the functionalities of the Dependability unit and the practical utility during the synthesis process, we consider a simple case study from a distributed marketplace scenario. In the considered scenario, the dynamic synthesis of a connector with a given dependability level is required to enable successful interoperation among heterogeneous consumers and merchants.

The paper is organised as follows. Section II summarises the main aspects of our reference framework, which includes the units for Discovery/Learning, Synthesis, Dependability, and Monitoring. The architectural structure of the Dependability unit is introduced in Section III, pointing out both its internal organisation in modules and a possible implementation for each module. The case study on a distributed and heterogeneous marketplace application is presented in Section IV. The final considerations and conclusions are drawn in Section V and VI.

II. REFERENCE FRAMEWORK

Following the approach taken in the CONNECT Project, which is general and well suited for addressing interoperability in the context of complex pervasive systems, we consider a reference framework with four logical units: Discovery/Learning, Synthesis, Dependability and Monitoring.

Discovery/Learning. This unit gathers information on the networked systems. Specifically, the unit discovers mutually interested networked systems, and retrieves information on the specification of the interfaces of the networked systems. The unit assumes that networked systems are *discovery enabled*, i.e., they provide a minimal description of their intent and functionalities. When networked systems do not provide the complete description of their behaviour, the unit completes the specification through a learning procedure (e.g., via model-based testing).

Synthesis. This unit performs the dynamic synthesis of mediating connectors to enable interoperation among networked systems willing to interact. The unit uses the behavioural models built by Discovery/Learning to identify mismatches between the communication protocols employed by the networked systems, generates a connector that resolves the incompatibilities between the communication protocols, and deploys the connector.

Dependability. This unit supports the Synthesis unit in the definition of a connector that enables networked systems to interact with a given dependability level. Specifically, the unit executes a stochastic model-based quantitative analysis to determine if the dependability [3] level of the connected system meets the required dependability level –in the case the requirements are not satisfied, the unit reports possible dependability enhancements to Synthesis.

Monitoring. This unit becomes operational when the connector is deployed. The unit continuously monitors the deployed connector to update the functional and non-functional specification of the connector with run-time data.

In this framework, a networked system broadcasts a *connect request* whenever a new connection to a service is needed. The connect request contains a description of the requested service together with a specification of the required dependability level for the service. When Discovery/Learning detects a connect request, the unit searches the networked systems that can provide the requested service. If such systems are found and operate a communication protocol different from that of the networked system that made the connect request, Discovery/Learning activates Synthesis to generate a suitable connector that enables interoperation. The Synthesis unit, on the basis of the specification of the communication protocols, produces a mediating connector. Before connector deployment, Synthesis activates the Dependability unit to evaluate if the connected system that will be obtained satisfies the non-functional requirements expressed by the networked systems. If the non-functional requirements are satisfied, the connector is deployed; otherwise, Synthesis is supported by the Dependability unit in the definition of possible enhancements that can be applied. Once the connector is deployed, the Monitoring unit continuously updates the functional and non-functional specification of the connector with run-time data

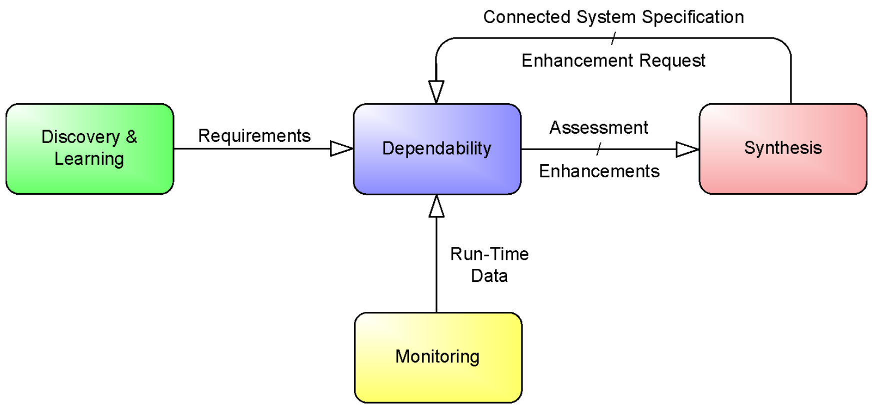


Fig. 1. Input-Output relations between the Dependability unit and the other units.

to allow the other units to take into account dynamic system changes.

In this work, we elaborate on the Dependability unit. A Dependability-centric view of the input-output relations between the Dependability unit and the other units is shown in Figure 1: Discovery/Learning provides the dependability requirements; Synthesis provides the specification of the connected system, and possibly requests a dependability enhancement; Monitoring provides run-time statistical data on the execution of the deployed connector. The dependability assessment and the enhancements determined by the Dependability unit are used by Synthesis.

III. Architecture of the Dependability Unit

The Dependability unit 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 uses the generated dependability model to perform a quantitative assessment of the non-functional requirements reported by Discovery/Learning. The Evaluator checks the analysis results to determine if the non-functional requirements are met. If the requirements are satisfied, the Evaluator reports to Synthesis that the connector can be successfully deployed. If, on the other hand, the requirements are not satisfied, the Evaluator reports a warning message to Synthesis. The Synthesis unit, in turn, may reply with a request to improve the dependability level of the connected system. Upon receiving such enhancement request, the Evaluator module activates a loop in the Dependability unit to determine solutions to improve the dependability level of the connected system, e.g., use an updated specification that takes into account an alternative connector deployment or enhance the connector specification with ad hoc dependability mechanisms.

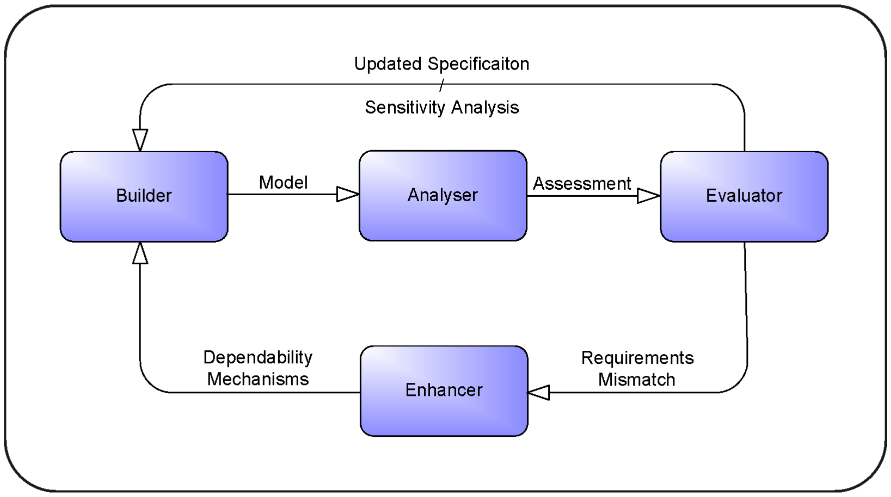


Fig. 2. Architecture of the Dependability unit.

In the following, we provide more details on the functionalities of the modules of the Dependability unit, and report on the prototype implementation we realised.

A. Builder

The Builder module takes in input the specification of the connected system from Synthesis, and the dependability requirements from Discovery/Learning. The specification includes both the nominal behaviour of the connected system, i.e., in fault-free situations, and the exceptional conditions, i.e., the failure modes. The module produces in output a dependability model of the connected system that contains enough details to assess the given dependability requirements.

Specification of the connected system. With reference to recent works on synthesis of mediating connectors [4] and automata discovery/learning [5], the specification of the connected system is given with Labelled Transition Systems (LTSs) annotated with non-functional information necessary to build the dependability model of the connected system. An LTS is an abstract machine that represents the sequence of actions performed by the system. Formally, an LTS is a tuple $(\mathcal{S}, \mathcal{S}_0, \mathcal{L}, \mathcal{T})$, where $\mathcal{S}$ is a set of states, $\mathcal{S}_0 \subseteq \mathcal{S}$ is a set of initial states, $\mathcal{L}$ is a set of labels, and $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{L} \times \mathcal{S}$ is a transition relation. Annotations include, for each labelled transition, the following fields: *time to complete*, *firing probability*, *failure mode*, and *failure probability*.

Dependability requirements. Typical dependability-related requirements are related with availability, reliability, performance and their combination. In our architecture, the dependability requirements provided by the networked systems are translated by Discovery/Learning into *metrics* and *guarantees*. Metrics are arithmetic expressions that describe how to obtain a quantitative assessment of the properties of interest of the connected system. In this context, metrics are expressed in terms of transitions and states of the LTS specification of the networked systems. Guarantees are boolean expressions that are required to be satisfied on the metrics under a set of constraints.

Dependability model. The dependability model of the connected system is specified with a formalism that allows to describe complex systems that have probabilistic behaviour, e.g., stochastic processes.

Implementation. The prototype implementation of the Builder module takes in input the LTS of the connected system described with Finite State Processes (FSP) [6]. LTS annotations are expressed as C++ functions. The dependability model of the system is specified with Stochastic Activity Networks (SANs), a generalisation of Stochastic Petri Nets introduced in [7] and formally defined in [8].

The SAN model is obtained from the LTS model by using the theory of regions [9]. A region identifies a set of states in the LTS such that all transitions with the same label either enter, exit, or never cross the boundary of the region. Each region in the LTS corresponds to a place in the derived SAN model, and each labelled transition in the LTS corresponds to an activity in the SAN model. A similar approach has already been used in other works to translate LTSs into Petri Nets (see, for instance, [10], [11] and [12]). In our approach, in order to have a well-defined probabilistic model, non-deterministic choices among k transitions outgoing from an LTS state are mapped in the SAN model into instantaneous activities with k case probabilities.

The metric is an arithmetic expression that may contain a predefined set of functions (see Table I for some examples). The guarantee is given by a boolean expression on the metric and a set of constraints on the connected system model (e.g., constraints on the time frame of evaluation of the metric and constraints on the behaviour of the networked systems). Statistical operators (e.g., $mean$ and $variance$), comparison and logical operators can be used in the expression. In order to uniquely identify states and transitions of the metric expression that appear in the generated SAN model, renaming is used to constrain the definition of the LTS regions.

Function	Description
$timeFrame(s) : \mathcal{S} \rightarrow \mathbb{R}^+$	returns the interval of time when the system is in state s
$minTimeStamp(tr) : \mathcal{T} \rightarrow \mathbb{R}^+$	returns the first instant of time when transition tr fires
$avgTimeStamp(tr) : \mathcal{T} \rightarrow \mathbb{R}^+$	returns the average instant of time when transition tr fires
$maxTimeStamp(tr) : \mathcal{T} \rightarrow \mathbb{R}^+$	returns the last instant of time when transition tr fires
$\#(tr, t1, t2) : \mathcal{T} \times \mathbb{R}^+ \times \mathbb{R}^+ \rightarrow \mathbb{N}$	returns number of times transition tr fires during the interval $[t1, t2]$
$\#(l, t1, t2) : \mathcal{L} \times \mathbb{R}^+ \times \mathbb{R}^+ \rightarrow \mathbb{N}$	returns number of times transitions with label l fire during the interval $[t1, t2]$

TABLE I

EXAMPLES OF PREDEFINED FUNCTIONS THAT CAN BE USED IN THE METRIC EXPRESSION.

B. Analyser

The Analyser module takes in input the dependability model from the Builder module and the dependability requirements from Discovery/Learning. The module builds a reward model, i.e., a model that enables a quantitative assessment of the metrics of interest, and makes use of a solver engine to obtain a quantitative assessment of the dependability metrics.

Reward model. The reward model is the dependability model extended with reward functions. Reward functions allow to specify properties of interest: they return a value depending on the system state, and can be evaluated either at an instant of time or accumulated over a time frame.

Solver. The solver evaluates the reward functions defined in the reward model. The evaluation can be performed either with analytical approaches or with simulation. Analytical approaches are based on state space analysis, and they are applicable when the model does not face state space explosion and when timing information follow deterministic or exponential distributions. Simulation, on the other hand, is always applicable (unless extremely rare events have to be evaluated), and is expected to be particularly useful when a tradeoff between accuracy and time to perform the assessment is needed.

Implementation. The prototype implementation of the Analyser is based on Möbius [13], a popular software tool that provides a comprehensive framework for model-based evaluation of dependability and performance aspects of systems. The framework supports, among others, the SAN formalism.

In Möbius, each reward function is a C++ function that returns a value depending on the marking of the SAN. There are two kinds of reward functions: *rate rewards* and *impulse rewards*. Rate rewards are used to implement time-based reward functions. Impulse rewards are used to implement action-based reward functions, i.e., they are associated with specific activities and can be evaluated only when the associated activity completes.

The reward functions are automatically derived from the metrics expression as follows: the metric is mapped into its syntax tree to decompose the metric into a combination of basic functions; the basic functions are translated into C++ functions by using a predefined repository of function templates. For instance, with reference to the functions shown in Table I, a rate reward template is used to translate $timeFrame(s)$, while an impulse reward template is used to translate $\#(tr, t1, t2)$. The content of the repository of function templates is not shown in this work, but some details will be provided in the case study shown in Section IV.

The Solver evaluates the reward functions via simulation. The quantitative assessment of the metric is obtained from the assessment of the reward functions by merging the results according to the arithmetic operations specified in the syntax tree of the metric expression.

C. Evaluator

The Evaluator reports to Synthesis if the connected system satisfies the dependability requirements provided by Discovery/Learning. In the case of requirements mismatch, the Evaluator sends a warning message to Synthesis, and may receive back a request to evaluate if enhancements can be applied to improve the dependability level of the connected system.

Requirements mismatch. If the requirements are not satisfied, the Evaluator may receive a request to explore one of the following three directions for improvements:

1. Update the specification of the connector to take into account an alternative connector deployment (e.g., a deployment that uses a communication channel with lower failure rate). Upon receiving this request, the Evaluator triggers a new analysis that considers the updated specification of the connector.
2. Enhance the specification of the connector by including dependability mechanisms (e.g., a message retransmission technique). Upon receiving this request, the Evaluator triggers a sensitivity analysis whose objective is to understand which failure modes of the connected system have highest impact on the dependability measure. Such failure modes will be the first ones to be considered for devising appropriate counter-measures capable of limiting their effects. To this end, the Evaluator builds a

sensitivity analysis campaign to instruct the Builder module on the creation of dependability model variants, each of which considers a specific subset of failure modes, among those foreseen. Whenever a variant is generated, the Analyser module performs the assessment of the metrics on the generated model. The Evaluator collects the analysis results and, after all variants have been analysed, produces a ranking of the failure modes. This ranking is used by the Evaluator to iteratively activate the Enhancer module as long as one of the following conditions is not met: the guarantees are satisfied, or the Enhancer signals that all possible dependability mechanisms have been explored.

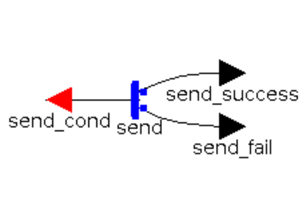
3. Apply a combination of the previously mentioned enhancements.

Implementation. The Evaluator compares the analysis results against the guarantees specified in the requirements, and reports a message containing a boolean value to Synthesis. The Evaluator may receive from Synthesis the following three types of enhancement requests:

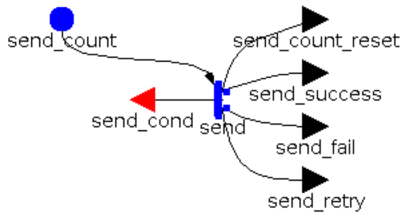
- *alternative deployment*: upon receiving this request, the Evaluator triggers a new analysis that considers the updated annotated LTS specification of the connector contained in the request.
- *dependability mechanism*: upon receiving this request, the Evaluator triggers a sensitivity analysis that considers the impact of one failure mode at a time. Specifically, for each failure mode, a variant of the SAN models is generated such that the considered failure mode is the only one considered in the model. The ranking of the failure modes is obtained by assessing the metrics on the generated variants: the higher the impact on the guarantees dissatisfaction, the higher the ranking of the failure mode.
- *combined enhancement*: a request that allows a combined use of the previously mentioned enhancements. The combined enhancement request contains the annotated LTS specification of the connector for the alternative deployment, and an indication that the Evaluator module should also enable the evaluation of dependability mechanisms to improve the connector. A field in the request is used to specify if the enhancements should be evaluated individually and/or in combination. When evaluated individually, an evaluation ordering is also provided.

D. Enhancer

The Enhancer is activated by the Evaluator when the guarantees are not satisfied and Synthesis makes a request to enhance the connector with dependability mechanisms. The Enhancer is instructed by the Evaluator module on the requirements mismatch and the failure mode that needs to be tackled. Specifically, the Enhancer performs the following actions: (i) selects a dependability mechanism that can be employed,



(a) Original model.



(b) Enhanced model with message retransmissions.

Fig. 3. Example of dependability mechanism and application rule.

among those available, to contrast the failure mode indicated by the Evaluator module; (ii) if all possible applications of dependability mechanisms have already been explored without success, the Enhancer sends a warning message to the Evaluator module; otherwise, the Enhancer instructs the Builder module on the application of the selected dependability mechanism in the connected system model and triggers a new analysis.

Dependability mechanisms. Dependability mechanisms are counter-measures that can be adopted to contrast failure modes. Typically, dependability mechanisms are based on the application of redundancy, e.g., duplication of system channels, or retry of message transmissions over system channels. The dependability mechanism, in this context, will be embedded in the synthesised connector, because networked systems are not under the control of the framework. Nevertheless, the dependability mechanisms embedded in the connector can be employed to improve, to some extent, the dependability level of the networked systems. For example, the reliability level of a transmission performed by a networked system can be improved through timeouts or message retransmissions applied at the connector level.

Implementation. The dependability mechanism suitable to contrast a given failure mode is determined through an ontology of dependability mechanisms, such as that reported in [14]. A detailed discussion on the content of the ontology is out of the scope of this paper.

We developed ad hoc dependability models for a set of relevant dependability mechanisms, and a set of rules to automate the application of the mechanisms in the SAN model of the connected system. The ad hoc models can be parametric; for instance, a retransmission mechanism is parametric with respect to the maximum number of allowed retransmissions. As an example of mechanism and application rule, in Figure 3 we graphically show a retransmission mechanism and how to modify the original model in order to apply message retransmissions to a send operation. Specifically, the original model contains a timed activity `send` that models the send operation, an input gate `send_cond` that specifies the enabling condition of the activity, and two output gates, `send_success` and `send_fail`, that specify the output functions in the case of correct and faulty behaviour. The enhanced model is obtained from the original model by adding the following elements: a place `send_count`, with initial marking the maximum allowed number of retransmissions; an output gate `send_count_reset`, which resets the marking of `send_count` to its initial value when the `send` succeeds; an output gate `send_retry`, which reactivates `send` as long as `send_count` contains tokens, and resets the marking of `send_count` after performing all retransmission attempts.

IV. Case Study

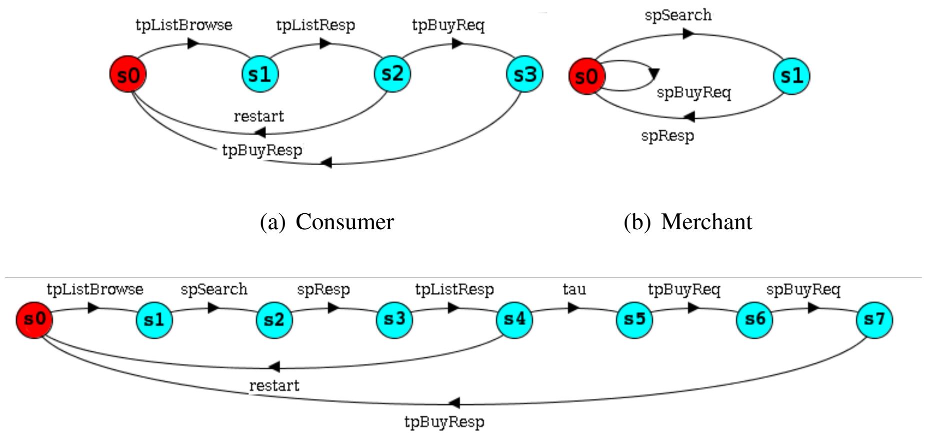
In this section, we show in detail the operations performed by the modules of the Dependability unit to support the synthesis of a dependable connector in a simple case study based on a distributed marketplace with heterogeneous networked systems. In the considered scenario, consumers need to interoperate with merchants to purchase products. Consumers and merchants have heterogeneous devices. In the general case, each merchant and each consumer may use a different protocol. Without loss of generality, in this case study we assume that all merchants have the same protocol $P1$, and that all consumers have the same protocol $P2$ ($P2 \neq P1$).

A. Connected System Specification

The LTS of the connected system is obtained as the parallel composition of the LTSs of consumers, connectors and merchants, which are defined hereafter.

The consumer follows a tuple space protocol, e.g., LIME [15], in which communication is obtained by reading and writing tuples onto a shared space. The consumer starts the interaction by writing a browse request (`tpListBrowse`) in the tuple space to check the availability of a certain product. After reading the response (`tpListResp`) from the tuple space, the consumer either restarts the interaction (`restart`), or writes a purchase request (`tpBuyReq`) and then reads the confirmation of successful transaction (`tpBuyResp`).

The merchant follows a message passing protocol, e.g., SOAP [16]. In the protocol, the consumer starts the interaction with a discovery phase (`spSearch`). After receiving all responses (`spResp`) from the available merchants, the consumer either directly sends a purchase request to a merchant (`spBuyReq`), or starts a new discovery phase.



(a) Consumer (b) Merchant

(c) Connector for one consumer and one merchant

Fig. 4. LTS specifications.

In the connected system, the number of consumers ranges over $[1, 100]$ and the number of merchants ranges over $\{1, 3, 5\}$.

The synthesised connector is generated according to the algorithm reported in [4]. Specifically, a `tpListBrowse` from a consumer is translated into a multicast `spSearch` directed to all merchants; the `spResp`s from the available merchants are collected and translated into a single `tpListResp`; a `tpBuyReq` from a consumer is translated into an `spBuyReq` followed by a `tpBuyResp`. A connector enables the interaction between one consumer and many merchants. The LTSs of consumer, merchant and connector are shown in Figure 4. The complete FSP specification is reported in the Appendix.

Annotations. All transitions are exponentially distributed, and have a rate equal to one message per second. The probability that the consumer restarts the communication is $P_{restart}$. The failure mode of a transition associated with any send (`spSearch`, `tpListResp`, `spBuyReq` and `tpBuyResp`) and any receive (`tpListBrowse`, `spResp`, `tpBuyReq`) action performed by the connector is message omission. The omission probability of the transition is linked to the number of devices that use the channel. Consumers and merchants use different channels, and there is an exponential relation between the omission probability and the number of devices that use the channel. The omission probability for consumers results in the interval $[0.7, 0.85]$, while for merchants results in the interval $[0.92, 0.96]$.

B. Dependability Requirements

The metric is *Service Discovery Ratio* (SDR), which gives the mean probability for a consumer to successfully complete a purchase transaction with a merchant in a given time frame. The metric is specified in terms of networked system transitions as follows: $SDR(t1,t2) = \dfrac{\#((\mathtt{S3},\mathtt{tpBuyResp},\mathtt{S0}),t1,t2)}{\#((\mathtt{S0},\mathtt{tpListBrowse},\mathtt{S1}),t1,t2)}$.

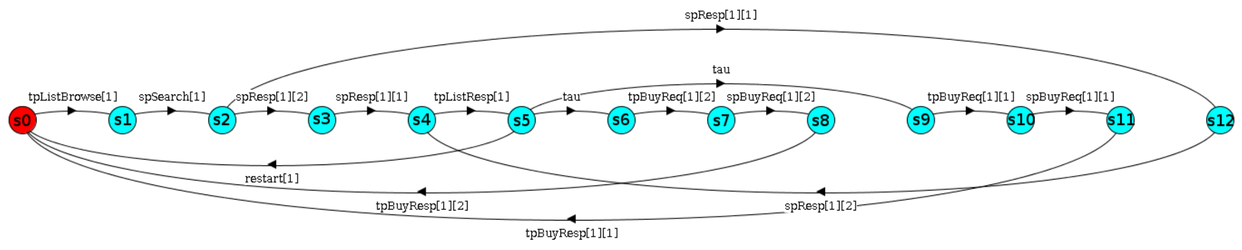
The guarantee on the metric is $mean(SDR(0,200)) \geq 0.7$. The constraint are: the consumer performs only one, fault-free, `tpListBrowse` request during the interval $[0,200]$, and a product is purchased after the consumer has successfully checked the availability of a merchant. These constraints can be expressed by the conjunction of the following conditions: the `tpListBrowse` request completes exactly once during the interval $[0,200]$, i.e., $\#((\mathtt{S0},\mathtt{tpListBrowse},\mathtt{S1}),0,200) = 1$, such request is correctly received by the connector, i.e., $P(fail(\mathtt{S0},\mathtt{tpListBrowse},\mathtt{S1}) \mid fire(\mathtt{S0},\mathtt{tpListBrowse},\mathtt{S1})) = 0$, and the consumer does not restart the transaction, i.e., $P(fire(\mathtt{S2},\mathtt{restart},\mathtt{S0})) = 0$.

C. Automated Dependability Analysis

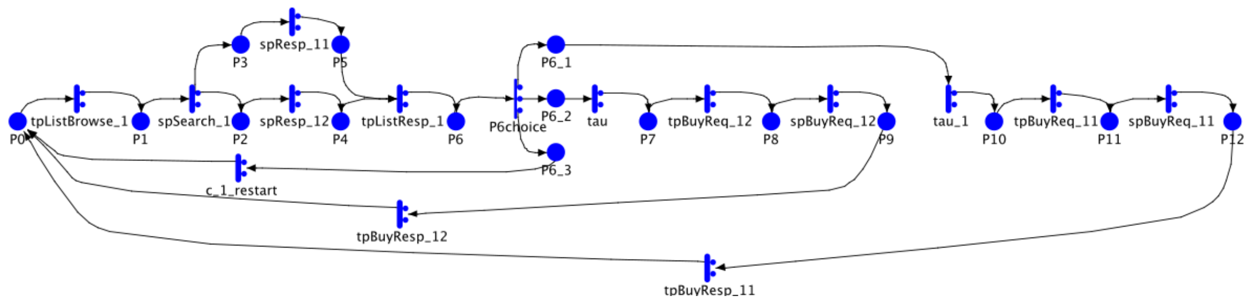
Dependability Model. The SAN model automatically generated by the Builder module has a timed activity for each labelled transition in the LTS. All activities have two case probabilities: case 1 is associated to the correct behaviour, while case 2 models the omission failure specified in the annotations. In order to show a human-readable model, in Figure 5(a) we show the LTS model in the case of one consumer and two merchants, and in Figure 5(b) we show the SAN model obtained from such LTS.

Reward Model. The syntax tree of the SDR metric expression has two branches: the quotient operator is the root of the tree, and the dividend and divisor are the two branches of the tree. The functions of the two branches are translated into impulse reward functions associated to the timed activities `tpBuyResp_11`, `tpBuyResp_12`, and `tpListBrowse_1`. The reward functions return 1 every time such transitions fire. The impulse rewards are evaluated during the time frame $[0,200]$, as specified in the guarantee.

Dependability Assessment. The generated reward model is evaluated via simulation. Connected systems with different number of consumers (c) and merchants (m) are considered, according to the provided specification ($c \in [10,100]$ and $m \in \{1,3,5\}$). In Figure 6(a), we show the threshold specified in the requirements (70%) and the trend of SDR (on the y axis) for different number of consumers (on the x axis) and different number of merchants (one curve for each possible number of merchants). We can notice that SDR monotonically decreases as the number of consumers gets larger. This is reasonable because, according to the specification of the channel used by the connector, higher number of consumers



(a) LTS model of the connected system.



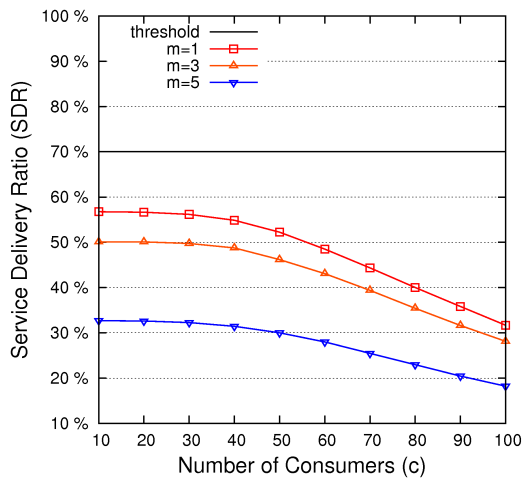
(b) SAN model of the connected system.

Fig. 5. Connected system models with one consumer, one connector and two merchants.

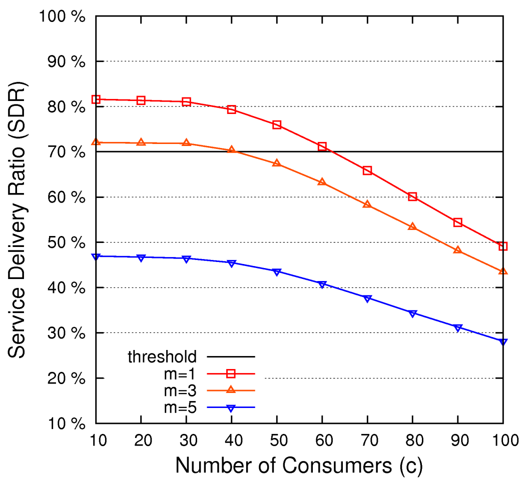
leads to higher message loss. The same applies when comparing the curves with different number of merchants: if we fix the number of consumers, SDR is lower when the connected system contains a higher number of merchants. This is due to the fact that the connector must wait for the responses from all merchants before replying to the consumer, and the probability that at least one response gets lost increases when the system contains a higher number of merchants.

Evaluation of Results. The guarantee ($SDR \geq 0.7$) is not satisfied for the admissible range of consumers and merchants. The Evaluator reports a warning message to Synthesis; here, we assume that Synthesis replies with a *combined improvement* request with a new deployment type and a dependability mechanism. The improvements must be evaluated individually and in combination. The specification associated with the new deployment reports that the failure mode on the consumers' channel is still linked to the number of devices that use the channel through an exponential relation, but that the omission probability results in the range $[0.81, 0.96]$. The first improvement that must be evaluated is a new deployment type.

Dependability Enhancement. The analysis results for the alternative deployment are shown in Figure 6(b). The guarantee is still not met for connected systems with high number of consumers and merchants.

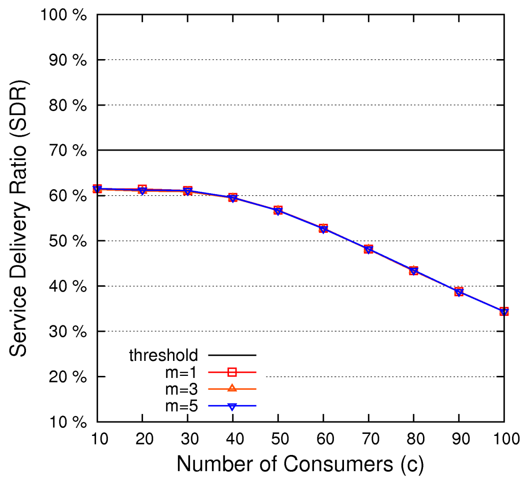


(a) Synthesised connector.

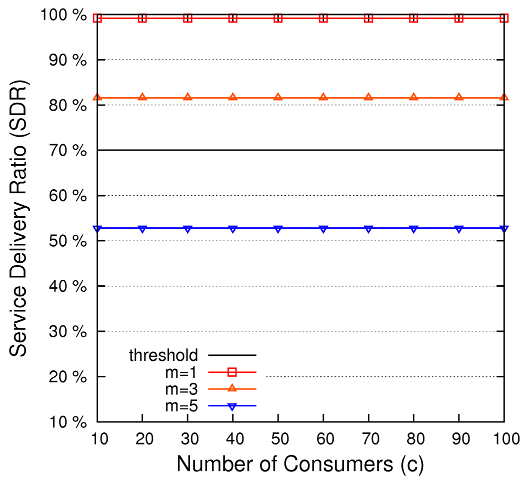


(b) Alternative connector deployment.

Fig. 6. Metric assessment.



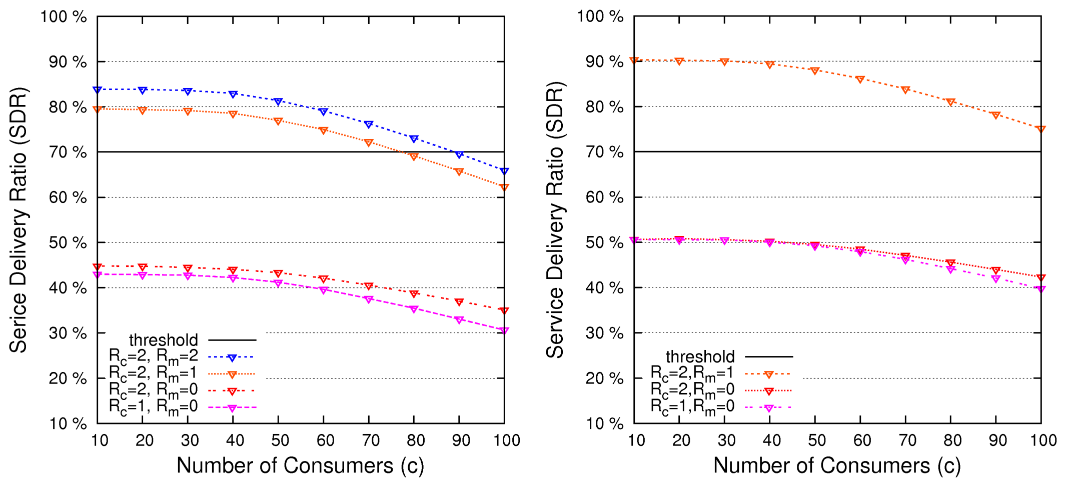
(a) Failures only on consumers' channel.



(b) Failures only on merchants' channel.

Fig. 7. Sensitivity analysis.

The Evaluator, hence, triggers a sensitivity analysis for the possible failure modes. The impact of each failure mode is evaluated separately. The results of the analysis when the omission failure on the consumers' channel is the only one considered are shown in Figure 7(a); the results when the failure on the merchants' channel is the only one considered are shown in Figure 7(b). The Evaluator gives the highest rank to the failure mode on the consumers' channel because the number of times the guarantee is dissatisfied is higher when the failure on the consumers' channel is enabled.



(a) Original deployment and retransmissions.

(b) Alternative deployment and retransmissions.

Fig. 8. Metric assessment when using dependability mechanisms.

The Enhancer, according to the ontology of dependability mechanisms, chooses a message retransmission technique to contrast the omission failures. We assume that the ontology reports that the maximum allowed number of retransmissions is 2. Figure 8(a) shows the analysis results for a connected system when the connector is enhanced with message retransmissions. The figure reports the value of SDR in case of 5 merchants, which is the most critical scenario. In the figure, R_c (R_m) represent the number of retransmissions performed on the consumers' (merchants') channel. The Enhancer tries to apply first the retransmission mechanism to the transitions performed on the consumers' channel. Nevertheless, the guarantee is not satisfied even when the maximum allowed number of retransmissions is used. Hence, the Enhancer applies the retransmission technique also to the transitions performed on the merchants' channel. Although SDR gets further improved, the guarantee is still not satisfied when the connected system contains high number of merchants.

The Evaluator, then, triggers an analysis of the connected system in the case of combined use of the two improvements. Figure 8(b) shows the analysis results for a connected system with 5 merchants (the most critical scenario): the first combination found that allows to satisfy the guarantee uses the alternative connector deployment and extends the connector specification with two retransmissions on the consumers' channel ($R_c = 2$), and one retransmission on the merchants' channel ($R_m = 1$). This information is reported to Synthesis in order to enable the deployment of an enhanced connector that satisfies the given dependability requirements.

V. Final Discussion

The architectural structure and implementation described in this paper constitute an important step towards the definition of an automated procedure to provide dependability analysis as a support the synthesis of dependable connectors. There are several aspects that still need to be investigated to fully reach the ambitious goal of automated dependability analysis, and a discussion on the main points that need to be addressed is proposed in the following.

Possible constraints on the time allowed for the dependability analysis to complete have not been accounted for so far. At the moment, the envisaged applications of the framework assume that networked systems networked systems do not suspend their activity while waiting the connector to be ready. Indeed, at the moment, timing constraints are considered only on the operations performed after connector deployment. Performing dependability analysis in a complete on-line setting is a very challenging problem and techniques are need to balance between time to produce results and their accuracy. For instance, in the automatic generation of the dependability model from the specification of the connected system, a technique needs to be developed to optimise the dependability model on the basis of the specific metrics that needs to be assessment.

Compositional solution methods for the dependability model would be desirable, possibly reusing partly solved model, e.g., when the synthesised connector is derived as specialisation of an already existing connector that has already been analysed, or when already analysed dependability mechanisms are introduced in the dependability model. Indeed, although the addressed context is dynamic and evolving, we assume that the pace at which evolution occurs is in general considerably slower that the requested rate of an already available connector. Hence, the networked systems are expected to remain stable for a significant portion of their lifetime and to request services for which the same synthesised connector can be reused to satisfy interoperability.

The ontology to support enhancement of the connector model with dependability mechanisms coping with failure patterns needs to be extended over time with new mechanisms, to enhance the handling of failure modes or to contrast new failure modes. We should also take into account that the identification of the dependability mechanisms may depend on the connector deployment, i.e., the connector may share resources with some of the bridged networked system, or the connector may have its own resources separate from those of the networked systems.

VI. Conclusions

This paper has presented an approach for the realisation of a completely automated dependability analysis in heterogeneous and dynamic environments. The presented research activity is undertaken in the context of the framework proposed in the European project CONNECT, which is general and well suited for addressing interoperability in the context of complex pervasive systems. A model-based approach has been adopted, which can be successfully employed since the early stages of the connector synthesis. The logical architecture of a Dependability unit that supports the synthesis of dependable connectors has been presented together with a prototype implementation. A simple case study has been considered to show how the approach works and the utility of the Dependability unit in guiding the connector synthesis.

Acknowledgements

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

References

[1] M. Weiser, "The computer for the twenty-first century," *Scientific American*, pp. 94–104, September 1991.

[2] "EU FP7 Project CONNECT (FP7–231167)," 2009–2013, http://connect-forever.eu/.

[3] 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*, vol. 1, pp. 11–33, 2004.

[4] R. Spalazzese and P. Inverardi, "Mediating connector patterns for components interoperability," in *Proceedings of the European Conference on Software Architecture (ECSA2010) - To Appear*, 2010.

[5] H. Raffelt, B. Steffen, and T. Berg, "Learnlib: a library for automata learning and experimentation," in *FMICS '05: the 10th intl. workshop on Formal methods for industrial critical systems*. New York, NY, USA: ACM, 2005, pp. 62–71.

[6] J. Magee and J. Kramer, *Concurrency: state models & Java programs*. New York, NY, USA: John Wiley & Sons, 2006.

[7] A. Movaghar and J. F. Meyer, "Performability modelling with stochastic activity networks," in *1984 Real-Time Systems Symposium*. Austin, TX: IEEE Computer Society Press, December 1984, pp. 215–224.

[8] W. H. Sanders and J. F. Meyer, "Stochastic activity networks: formal definitions and concepts," pp. 315–343, 2002.

[9] A. Ehrenfeucht and G. Rozenberg, "Partial (set) 2-structures. Part I: basic notions and the representation problem," *Acta Inf.*, vol. 27, no. 4, pp. 315–342, 1990.

[10] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev, "Deriving petri nets from finite transition systems," *IEEE Trans. Comput.*, vol. 47, no. 8, pp. 859–882, 1998.

[11] U. Buy and G. Singal, "Toward efficient algorithms for generating compact petri nets from labeled transition systems," in *COMPSAC '02*. Washington, DC, USA: IEEE Computer Society, 2002, pp. 717–722.

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

[13] "Mobius tool," http://www.mobius.illinois.edu/.

[14] "ReSIST: Resilience for Survivability in IST. Deliverable D33: Resilience-explicit computing." Tech. Rep., 2008.

[15] A. L. Murphy, G. P. Picco, and G.-C. Roman, “Lime: A coordination model and middleware supporting mobility of hosts and agents,” *ACM Trans. Softw. Eng. Methodol.*, vol. 15, no. 3, pp. 279–328, 2006.

[16] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, and H. F. Nielsen, “Soap version 1.2 part 2: Adjuncts,” W3C Recommendation, June 2003. [Online]. Available: http://www.w3.org/TR/soap12-part2/

Appendix

```
/* FSP specification of the distributed marketplace scenario */
const ConsumerN = 100
const MerchantN = 5
range CN = 1..ConsumerN
range MN = 1..MerchantN

CONSUMER = ( tpListBrowse -> tpListResp ->
             (restart -> CONSUMER | tpBuyReq -> tpBuyResp -> CONSUMER) ).

MERCHANT = ( spSearch -> spResp -> MERCHANT | spBuyReq -> MERCHANT).

MainCONNECTOR = ( tpListBrowse -> spSearch -> tpListResp ->
                  ( restart -> MainCONNECTOR | select -> tpBuyResp -> MainCONNECTOR )).
SubCONNECTORa = ( spSearch -> spResp -> tpListResp -> SubCONNECTORa ).
SubCONNECTORb = ( select -> tpBuyReq -> spBuyReq -> tpBuyResp -> SubCONNECTORb ).

||CONNECTOR = ( con:MainCONNECTOR /{ tpListBrowse/con.tpListBrowse, spSearch/con.spSearch,
                                     tpListResp/con.tpListResp, restart/con.restart,
                                     forall[i:MN] {select[i]/con.select},
                                     forall[i:MN] {tpBuyResp[i]/con.tpBuyResp} }
                || sa[MN]:SubCONNECTORa /{ forall[i:MN] {spSearch/sa[i].spSearch},
                                           forall[i:MN] {spResp[i]/sa[i].spResp},
                                           forall[i:MN] {tpListResp/sa[i].tpListResp} }
                || sb[MN]:SubCONNECTORb /{ forall[i:MN] {select[i]/sb[i].select},
                                           forall[i:MN] {tpBuyReq[i]/sb[i].tpBuyReq},
                                           forall[i:MN] {spBuyReq[i]/sb[i].spBuyReq},
                                           forall[i:MN] {tpBuyResp[i]/sb[i].tpBuyResp} } )\ {select}.

||SYS = ( c[CN]:CONSUMER /{ forall[i:CN] {tpListBrowse[i]/c[i].tpListBrowse},
                            forall[i:CN] {tpListResp[i]/c[i].tpListResp},
                            forall[i:CN] {restart[i]/c[i].restart},
                            forall[i:CN] { forall[j:MN] {tpBuyReq[i][j]/c[i].tpBuyReq} },
                            forall[i:CN] { forall[j:MN] {tpBuyResp[i][j]/c[i].tpBuyResp} } }
         || m[MN]:MERCHANT /{ forall[i:MN] {forall[j:CN] {spSearch[j]/m[i].spSearch} },
                              forall[i:MN] {forall[j:CN] {spResp[j][i]/m[i].spResp} },
                              forall[i:MN] {forall[j:CN] {spBuyReq[j][i]/m[i].spBuyReq} } }
         || con[CN]:CONNECTOR /{ forall[i:CN] {tpListBrowse[i]/con[i].tpListBrowse},
                                 forall[i:CN] {spSearch[i]/con[i].spSearch},
                                 forall[i:CN] {tpListResp[i]/con[i].tpListResp},
                                 forall[i:CN] {restart[i]/con[i].restart},
                                 forall[i:CN] { forall[j:MN] {tpBuyReq[i][j]/con[i].tpBuyReq[j]} },
                                 forall[i:CN] { forall[j:MN] {spBuyReq[i][j]/con[i].spBuyReq[j]} },
                                 forall[i:CN] { forall[j:MN] {tpBuyResp[i][j]/con[i].tpBuyResp[j]} },
                                 forall[i:CN] { forall[j:MN] {spResp[i][j]/con[i].spResp[j]} } } ).
```