Authors: Loris Dal Lago, Orlando Ferrante, Roberto Passerone and Alberto Ferrari

Title: Dependability Assessment of SOA-Based CPS With Contracts and Model-Based Fault Injection

This paper appears in: IEEE Transactions on Industrial Informatics
vol. 14, no. 1, pp. 360-369, Jan. 2018.

DOI: 10.1109/TII.2017.2689337.

Dependability Assessment of SOA-based CPS with Contracts and Model-Based Fault Injection

Loris Dal Lago, Orlando Ferrante, Roberto Passerone, *Member, IEEE,* and Alberto Ferrari

***Abstract*—Engineering complex distributed systems is challenging. Recent solutions for the development of Cyber-Physical Systems (CPS) in industry tend to rely on architectural designs based on Service Orientation (SOA), where the constituent components are deployed according to their service behavior and are to be understood as loosely-coupled and mostly independent. In this paper, we develop a workflow that combines contract-based and CPS model-based specifications with service orientation, and analyze the resulting model using fault injection to assess the dependability of the systems. Compositionality principles based on the contract specification help us make the analysis practical. The presented techniques are evaluated on two case studies.**

***Index Terms*—SOA, dependability, cyber-physical, contract-based, model-based, fault injection.**

I. Introduction

Assessing the dependability of large-scale distributed Cyber-Physical Systems (CPSs) is a difficult task that involves understanding the systems dynamics both in terms of functionality and of network interaction and communication. The study of dependability can be interpreted as the identification of the possible manners under which the system can break down. This is usually expressed in terms of failures, i.e., as paths leading the system to a violation of a given desired property. For industrial practice, it is useful to have these paths laid down using Fault Trees (FT) and Failure Mode and Effect Analysis (FMEA) tables. Yet, elaborating these by hand is costly in terms of both time and budget and often prone to human error, especially as the system scale grows, the architecture gets more distributed, and diverse engineering teams work on different system features.

One natural solution is to adapt techniques from the domain of requirement verification to requirement robustness checking, as most commonly achieved using Model-Based Fault Injection, or Model Extension [1]. When a CPS is deployed over a network, the analysis requires that individual components be distributed in a modular way, precisely defining their role in the architecture and their failure possibilities. This is commonly achieved through the Service Oriented Architectures (SOA) paradigm [2], [3]. This, however, normally lacks a fully parallel recognition at the modeling level, where the fault modalities of the single components are not available on an individual basis and where hardly ever are there explicitly stated dependability dependencies liable to the network.

In this paper, we address the issue of dependability by integrating methods from the SOA paradigm, CPS techniques, formal verification and model based software engineering practice. In particular, we adopt a contract-based approach [4] to specify properties offered by single services when participating in a compatible environment, and use Model-Based Fault Injection [1] to take care of the dependability aspects. Our formal assume-guarantee model leads naturally to modularity. We therefore exploit the principle of compositionality [5] to make the analysis practical and avoid the state explosion problem. We show how the formalism consistently supports non-functional properties, such as timing and availability, and quantify the validity of our techniques by practical implementations on two use cases. Dependability assessment in our context is a rigorous study of resilience to faults, diagnosability and fault event analysis, that span from physical, to functional, to time delays and network availabilities. Resilience refers to the ability of the system to preserve its nominal properties. The safety analysis tool xSAP [6] is used for the assessment of dependability using Model-Based Fault Injection, an approach that has already be proven itself effective on industry scale systems [7]. Our contribution in this sense is showing its application on systems other than digital.

The paper is organized as follows. We first give an account of related work in Section II. Then, Section III discusses modeling dependability for CPS using SOA and contracts on a thermostat example. Dependability analysis is discussed in Section IV. Finally, Section V illustrates an emergency response case study, extends the approach and discusses performance metrics.

II. Related Work

There is a long history of validating systems against dependability, often with techniques based on the intentional injection of faults. One example of dependability analysis is presented by Looker et al. [8], where the fault injection is performed at the network level on the exchanged packets of a finalized system in place. The use-case is a feedback-based thermostat that exchanges messages over a Service-Oriented Architecture. Our reference example is inspired by that work and, besides a great deal of adaptation and rethinking, it has a conceptual resemblance in the fault injection procedure, displaying faults as delays at the network level. The model-based view that we propose, however, takes advantage of the

L. Dal Lago, O. Ferrante and A. Ferrari are with ALES S.r.l., Piazza della Repubblica 68, 00185 Rome, Italy (e-mail: loris.dallago@utrc.utc.com, orlando.ferrante@utrc.utc.com, alberto.ferrari@utrc.utc.com).

R. Passerone is with the Department of Information Engineering and Computer Science, University of Trento, via Sommarive 9, 38123 Trento, Italy (e-mail: roberto.passerone@unitn.it).

DOI: 10.1109/TII.2017.2689337

integration between the formal system and the dependability view to make a comprehensive early awareness of the available system dynamics, before deployment.

Xu et al. propose a set of techniques to verify compliance of services against a service workflow specification [9], [10]. Their model is based on a variant of Petri Nets and requirements are specified using formulas similar to Computation Tree Logic (CTL). Our approach differs, in that we are interested in finding minimal sets of faults that make the system violate the requirements in the context of a CPS. Thacker et al. enrich the traditional Petri Net formalism with continuous dynamics, however the approach is reconcilable to automata for both expressiveness and algorithmic techniques [11]. Rosenkrantz et al. [12] propose a graph-based model for service-oriented networks to quantify the resilience of the system under node and edge failures. The authors develop an algorithm to compute the maximum number of node or edge failures that the network can tolerate. Our objectives are similar, and we employ flags over dedicated extra ports to model availability. However, our model is more general, and is able to deal with more complex properties and the actual service behavior, rather than only with the topology of the system. More recently, Mehnni et al. have introduced SafeSysE, a safety profile and dedicated algorithms for the generation of dependability artifacts from a SysML specification [13]. Our modeling approach is richer, and makes use of model checking tools that guarantee minimality, although at the expense of higher computational complexity. Our use of compositionality and contracts helps to address these problems.

In the present work we use a formalism based on automata, and adapt its inputs to encompass the non-functional aspects relative to the introduction of the network in the system, following a multiple-viewpoint approach [14], founded on model-based fault injection [1]. Model-based fault injection consists in extending the model-abstraction semantics of the system with additional faulty behaviors, in a controlled way, to investigate the reliability of the system in terms of the occurrence of faults that plausibly trigger those behaviors. One example is given by Ezekiel and Lomuscio [15] where modal epistemic logic is used to represent knowledge of cooperation in multi-agent scenarios, combined with modal temporal logics to analyze complex systems in terms of their tolerance to faults. Our approach follows that proposed in [1], using xSAP [6] to study of dependability in terms of tolerance, diagnosability and fault event analysis, with automatic model extension and automatic construction of fault event artifacts such as Fault Trees or FMEA tables. The tool is mature and has already been used on industrial case studies [7]. Among the approaches related to xSAP or its predecessors, the closest to our work are Networked Event-Data Automata, which use model-based fault injection techniques to study the dependability of systems, taking into account the interconnections of automata in a network [16]. In our work we exploit a similar level of formalism to model cyber-physicality of system components, but in addition focus on the service interactions typical of SOA and their compositional nature. Besides the mentioned Petri Nets, alternatives to the use of automata include models based on discrete events [17], [18]. For instance, Vyatkin et al. propose an approach based on the PTIDES model, which is of interest given the distributed nature of services [18]. Timing is modeled by time-stamping messages, which is shown to help the stability of the system using simulation. We prefer the use of automata, which are more easily handled by formal verification tools.

Our dependability analysis considers the functional system level and its dependency to network contingencies in a cohesive, yet separated, way. Derler et al. [19] suggest an approach where different viewpoints are developed independently and with mutual guarantees of correct working in the form of design contracts between, for example, control and software engineers. Our work is along the same lines but contextualizes the cyber-physical modeling over SOA. Regarding the effectual modeling of faults and contingencies in SOA, our work follows the model proposed by Broy et al. [20], extending it with the use of contracts. Farcas et al. [21] also extend that model and use the Service Architecture Definition Language (SADL), which includes primitives to express service unavailability or connection drops, thus accounting for failures. After modeling, the system is verified to check the validity of temporal properties in presence of faults in the architecture, triggered non-deterministically by the SPIN model checker. Unlike our work, SADL can only express the architectural interactions, as Message Sequence Charts (MSCs), but not the functional dynamics of the single components. Also, our dependability assessment is based on the establishment of dependability artifacts, such as Fault Trees and FMEA tables, that are constructed alongside to violations of invariant propositional properties in the model. In addition to that, the support technology that we chose for the artifact construction can handle verification of temporal properties on the extended model with non-deterministic faults.

Gössler et al. [22] propose a formal framework for reasoning about logical causality in contract violation, to establish relations between a fault (violation of a component contract) and a damage (violation of the overall system contract). The analysis is conducted a posteriori on the system execution traces. Their approach is complementary to ours, which can be used during system design to analyze the conditions under which a contract violation may occur. The combination of the two approaches could lead to a seamless analysis framework from design to deployment.

III. Modeling Dependability

In this and the next sections, we discuss the steps that we use in our methodology to make the dependability analysis of SOA-based CPS systematic, so that it can be translated to a usable tool automation process. We propose an inclusive approach that treats the SOA aspect and the cyber-physical aspect at the same level, and talk about Service-Oriented Cyber-Physical Systems (SOCPS) to stress their peer-level modeling. Our approach, shown in Figure 1, begins with a formal specification of the service-oriented architecture of the system, using a combination of the modeling standards SysML and SoaML. This is followed by the specification of properties

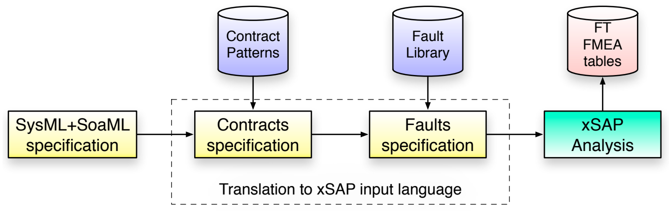

Fig. 1. Overall steps of the methodology

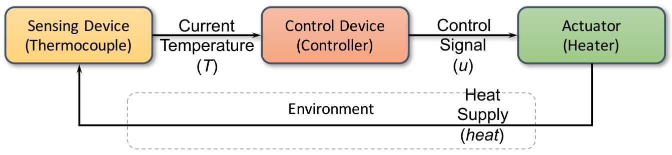

Fig. 2. Thermostat system

as contracts, each composed of an assumption and a guarantee, which are expressed using a pattern-based language to simplify the design entry. Faults are introduced as additional ports of components based on a suitable selection from a library and prepared to be analyzed, carrying information on their presence and degree of impact. The model is then translated into the SMV language and analyzed using the xSAP tool [6], which returns the dependability artifacts of interest, such as Fault Trees (FT) and FMEA tables.

We illustrate our approach by means of an example, a service-oriented thermostat controller that regulates the temperature in a room. The example is adapted from the dependability analysis of Looker et al. [8], including the fault injection based on latencies. The thermostat controller is a feedback system composed of a Thermocouple (the sensor), a Controller (the control device) and a Heater (the actuator), shown in Figure 2. The three components exchange SOA messages of current temperature, control and heat supply. As in the original paper, the thermocouple takes as input the heat supply and internally computes the change in temperature, thus abstracting the provision of heat to the environment and its effect as a SOA communication between actuator and sensor. We let this computation happen once per second, so that the estimate is not exceedingly far from reality. The space embedding the devices is surrounded by walls, where the outside environment is large enough to neglect any temperature variation due to heat transfer. The maximum heat dissipation of the room is a parameter of our model. Interestingly, while the example may look simple, its feedback structure is characteristic and representative of many systems of interest. As we shall see, the feedback and the introduction of fault injection make even this example difficult to analyze.

A. *Specification Language*

Our problem definition language of choice is a combination of the UML standards SysML and SoaML that we hereafter call SysML+SoaML. There exist a wide range of languages that are available to model SOA systems, although many are not prone to the modeling of SOCSPs. WS-BPEL and WS-CDL are XML-based static process execution languages, respectively used for modeling orchestration and choreography in web services. OWL-S and WSMO on the other hand are ontology-based process execution languages whose behavior is dynamic and defined during execution [23]. We are not interested in process execution languages, because we do not need the level of detail that they offer, especially on the implemented service composition. We need a sufficiently abstract language to yield favorably to verification, able to capture both the architecture of the system and its cyber-physical dynamics. SysML+SoaML, which also features the previously advocated separation between the functional structure of the system and its architecture, is a good option for that. Familiarity of system engineers to UML is also a point that is not shared with other abstract languages to model SOA.

Besides the system dynamics, we need to track the non-functional aspects, such as timing issues relative to the network service availabilities. One technique is to extend the expressiveness of the language with additional constraints and annotations [24]. Our approach, instead, is to enrich the models with additional ports, interpreted as functional by the verification and dependability analysis engines, and as non-functional parameters in the interpretation of the system as a whole, as will be described in the next sections.

B. *Contracts for SOCPS*

We employ the framework of contracts [4], [25] for the specification of design requirements on the services of the architecture, in the form of assumptions and guarantees from and to the rest of the system. When put together in the same system, services are required to satisfy a given top-level collaborative goal. In this scenario, the dependability problem studies the circumstances that can prevent the system from reaching the goal upon the violation of one or more of the services guarantees, under validity of the assumptions. The benefits of adopting a contract-based approach are manifold and include design modularity, separation of tasks across working teams and separation of responsibilities. In case of SOA, it is a perfect fit to grasp its natural interface interaction. This will be instrumental to making the analysis of the system effective, as described below in Section IV-A.

Formally, a contract is a pair of properties (A, G) that specify the assumption and the guarantee of the object to which it is attached [4], [25]. Contract implementation makes the semantic of contracts explicit: a component M is said to *implement* a contract C, written $M \models C$, whenever the guarantees hold in the scope of the assumptions.

A contract $C_1 = (A_1, G_1)$ *refines* $C_2 = (A_2, G_2)$, denoted $C_1 \preceq C_2$, whenever the set of implementations of C_1 is subsumed by the set of implementation of C_2. This happens if the assumptions of C_1 are at least as broad as those of C_2 and the guarantees of C_1 are at least as strict. The refinement relation over the class of contracts is a partial order, and forms a distributive lattice with $(A_1, G_1) \wedge (A_2, G_2) = (A_1 \vee A_2, G_1 \wedge G_2)$ the pairwise *conjunction*. Conjunction of contracts is commonly used to combine different viewpoints of the same component [4]. To combine contracts of different systems constituents, we rather need the operator of *parallel*

Id	Requirement	A/G	Comp.
r1	The reference temperature is given in $^\circ$C and ranges in [15, 25]	A	System
r2	The system reaches the reference temperature within 10 minutes and never deviates by more than 1°C thereafter	G	System
r3	The thermocouple can work against power levels no higher than 70 KW	A	Sensor
r4	If the heater supplies energy at 2.1 KW more than the maximum dissipation (7.5 KW), the temperature increases by a rate of 0.06°C per second	G	Sensor
r5	If the heater supplies energy at exactly the maximum dissipation (7.5 KW) or up to 10 W more, the temperature sensed by the thermocouple increases by a rate of no more than 0.0003°C per second	G	Sensor
r6	The heat controller is able to bring the temperature of the room up to steady 14°C - 28°C	A	Controller
r7	The temperature value sent to the controller is assumed never to be below zero, nor higher than 30°C	A	Controller
r8	The output of the heat controller ranges continuously between -1 and 1: negative values signal to lower the temperature level, positive values to increase it and 0 to keep it steady	G	Controller
r9	The heater can supply energy from 0 to 10 KW	G	Heater
r10	For the heater to work properly, input signals must range within [-1, 1]	A	Heater
r11	Positive values on the input signal always imply positive heat supplies by the heater	G	Heater

TABLE I
REQUIREMENTS FOR THE THERMOSTAT SYSTEM TAGGED WITH THE ASSUMPTION/GUARANTEE LABEL AND COMPONENT

System contract	
A	$[15 \leq T_{ref} \leq 25]$
G	$[T' = T_{ref}$ **within** $[600s]]$ **and** $[T' = T_{ref}$ **implies** $[[T_{ref} - 1 \leq T' \leq T_{ref} + 1]$ **always**$]$ **always**
Sensor contract	
A	$[0 \leq heat \leq 70000]$ **and** $[T[0] = 0]$
G	$[heat > 7500 + 2100]$ **implies** $[T' > T + 0.06]$ **within** $[1s]$
G	$[7500 \leq heat \leq 7510]$ **implies** $[T \leq T' \leq T + 3 \cdot 10^{-4}]$ **within** $[1s]$
Control Device contract	
A	$[14 \leq T_{ref} \leq 28]$ **and** $[0 \leq T' \leq 30]$
G	$[T' < T_{ref}]$ **implies** $[0 \leq u \leq 1]$
G	$[T' > T_{ref}]$ **implies** $[-1 \leq u \leq 0]$
Heater contract	
A	$[-1 \leq u \leq 1]$
G	$[u > 0]$ **implies** $[heat > 0]$
G	$[0 \leq heat \leq 10000]$

TABLE II
CONTRACTS FOR THE SYSTEM AND THE SERVICES

composition, denoted $\|$, that allows for the assumptions of one contract to be partially satisfied by what the other contract offers as a guarantee. In formulas [4]:

$$(A_1, G_1) \parallel (A_2, G_2) = ((A_1 \wedge A_2) \vee \overline{(G_1 \wedge G_2)}, G_1 \wedge G_2).$$

It can be shown that under these assumptions parallel composition preserves both refinement and implementations [25]. This property, which we call compositional refinement, guarantees that if a composition of contracts $C_1 \parallel \cdots \parallel C_n$ refines a top-level contract C, then a composition of implementations $M_1 \parallel \cdots \parallel M_n$ implements the top-level contract C as long as each component M_i implements its individual contract C_i.

In addition to refinement and implementation, the theory we adopt supports the notion of contract consistency (existence of an implementation) and contract compatibility (consistency of the assumptions) [4]. Because we are interested in dependability under fault injection, we assume that the system is fully functional in nominal mode. Therefore, the premise is that aspects of consistency and compatibility have been dealt with using appropriate techniques [26], [27].

C. Thermostat specification

The three thermostat components of Figure 2 represent independent services that interact at the interface level. We specify the input/output behavior of the services using contracts.

Table I shows an informal set of requirements for the overall system and its independent services. Each requirement is labeled as Assumption or Guarantee, and is attached to the overall system or a specific component. For instance, the system must receive a reference temperature in the range $[15, 25]^\circ$C (r1), and stabilize in 10 minutes (r2). For the Thermocouple, we require that the heat supply does not get too intense (r3), in which case the sensor is not functional. The Thermocouple ensures a time-proportional rise in the temperature of the room, according to how much the heat supply outdoes the maximum dissipation of the room. Requirements r4 and r5 are particular demands on temperature increase per second. The Control Device has a constant reference temperature threshold as input between 14°C and 28°C (r6), plus the temperature feedback from the Thermocouple is assumed never to be lower than 0°C nor higher than 30°C (r7). The output is given by a control signal that takes values in the continuous interval $[-1, 1]$, specifying whether the temperature should be lowered, kept constant or increased (r8, r10). The Heater can supply energy from 0 to 10 KW (r9). The actual value is internally calculated by the Heater, accounting for the input signal, and then actuated (r11).

The techniques described in this work are to some extent independent of the particular formalism used to express contracts, as long as it supports temporal constraints. Here, we adopt the pattern-based contract framework embodied by the Block-based Contract Language BCL [28]. The reason underlying our choice is the simplicity of a pattern-based approach combined with the Simulink front-end that the language makes available, which could integrate property proving and simulation in the early phases of development. In BCL, contracts are expressed using patterns and expressions over the system variables. Patterns are built in layers to express invariants in terms of events and their time and logical relations. In particular, the keyword **implies** introduces a logical implication, while **within** introduces a timing constraint. The keyword **always** specifies that a timing constraint must be satisfied at every instant. The primed version of a variable is used to denote its value in the following evaluation step, using a synchronous underlying model.

Table II shows the BCL contract specification, obtained as the combination of individual requirements. Without pointing out it explicitly in the formal specification, we assume that the reference temperature is kept constant throughout evolution. Also, we require the initial temperature of the room and the environment temperature to be 0°C.

The natural requisite for correctness of the system is that the contract attached to the three component services compose up to refine the overall system contract. The problem is that, according to requirement r2, the system (top-level) contract is expressed using a range of 600 seconds (10 minutes), whereas the single components are defined by 1 second increments (e.g., requirement r4). Our approach to address this difference is to unfold the composite contract specification by constructing a contract of incremental granularity up to reaching the needed 600 s. Following this direction, we start from the definition of the contract $Thermostat_0$ as the parallel composition of the three service contracts at $t = 0$, when the temperature is $0°$C, according to contractual specification. The composite contract will have the reference temperature T_{ref} as the input variable and the temperature after 1 second ($T[1]$) as the output. Similarly, we can devise one $Thermostat_k$ contract for each k value up to 600, each having the reference temperature T_{ref} and the temperature $T[k]$ as input, and the temperature $T[k+1]$ as the output. Taking the composition of all of these contracts gives

$$Thermostat = \parallel \{Thermostat_k\}_{k\in[0\ldots599]}$$

that takes the constant reference temperature T_{ref} as the input and has the temperature after 600 seconds $T[600]$ as the output. This new contract can then be shown to refine the top-level contract of the overall system by boolean algebraic manipulations of the involved formulas. Once this step of checking the compositional refinement is done, the single services of the architecture can be implemented individually without further connections to the others.

D. Service Implementation

We specify each service implementation as a set of equational laws that can be shown to satisfy the corresponding service contracts. The Thermocouple is formally implemented by acting on the internal energy of the system, accounting for dissipations using the differential equation:

$$\frac{\mathrm{d}Q_{out}}{\mathrm{d}t} = \alpha_{cond} \cdot A \cdot (T - T_a). \qquad (1)$$

Here, Q_{out} measures the heat natural dissipation in homogeneous conditions, $\alpha_{cond} = [5, 20]$ W/(m^2·K) is the heat-transfer coefficient of the wall, $A = [50, 500]$ m^2 is the surface of the room exposed to the outside, T is the temperature of the room and T_a is the ambient temperature facing the wall on the outside, set to $0°$C constant, and assume no other dissipations.

The heat supply from the actuator contributes positively to the internal energy of the room, whereas the dissipation is wasted energy. By inverting the heat equation $\Delta Q = C_s \cdot m \cdot (T' - T)$ we obtain the new temperature of the room, T', under a heat derivative supply of $heat$ for Δt seconds, as

$$T' = \frac{\Delta Q_{room}}{C_s \cdot m} + T = \frac{\Delta t \cdot heat - \Delta Q_{out}}{C_s \cdot m} + T \qquad (2)$$

where $C_s = [1000, 1015]$ J/(Kg·K) is the (isobaric) specific heat of air and $m = [35, 400]$ Kg is the mass of air in the room. The value Δt is set to 1 s in the functional implementation.

The system actuator is implemented by a 10 KW Heater module that increases power based on the control signal u, according to the following rule:

$$heat = \min(heat + 200 \cdot u, 10000).$$

A PID controller implements the control for the system with parameters $K_p = 0.02$, $K_i = 0$ and $K_d = 1.5$, tuned by simulation on different values of the reference temperature and guaranteeing the satisfaction of the controller contract.

IV. Assessing Dependability

Faults in our systems of interest can happen in relation to the functional aspects of the system or to the delivered Quality-of-Service. Functional issues can be studied using standard model extension techniques [1] or, faithful to our contractual view, as contract violation [29]. The situation is different if we want to analyze Quality-of-Service requirements, i.e., non-functional properties. Those mainly relate to time properties of communication (e.g., latencies, jitter, round-trip time) and are paramount for the correct serving of distributed Cyber-Physical Systems. In the case under analysis, delays in communications can stream out-of-date messages in the network, making the internal partial representation of the system of each participant inconsistent, possibly leading the system itself to a crash, and with poor diagnostic information.

An effective way to inject faults as latencies, or other quantities, is thus essential for the system dependability assessment. Our approach is to reduce it to a functional problem, introducing for each input and output variable of each participant an associated fictional port carrying the non-functional values, in a similar style of [14]. If these ports are related to the timing viewpoint, we call them *latency ports*. We attach one latency port for each service input signal (*input latency port*) and one latency port for each service output signal (*internal latency port*). Values for the internal latency ports are produced by the service, and specify the amount of time needed for the corresponding signal to be emitted once all dependent inputs are available. On the other side, input latency ports express the time difference from the last arrival of their relative input signal to the considered current arrival event. For instance, the Thermocouple has an additional input port ilp_{heat} (for input latency port) and an output port int_T (for internal latency port), bound to communicate the time that the message needs to reach the receiver from the moment it is created.

Latency ports are allowed to interact exclusively with a Network participant component, that we introduce anew (see Figure 3). The role of the Network agent is to forward latency port values from one component to another, based on network interconnections, integrating those values with delays coming from commonplace communication. Furthermore, in order to keep coherence in the model, we let all functional signals pass through the Network, so that services never directly communicate with each other in any way.

An abstract SysML+SoaML picture related to our use case example is represented in Figure 3, where input latency ports are represented in green and internal latency ports in red. For instance, the Thermocouple needs 1 second between one

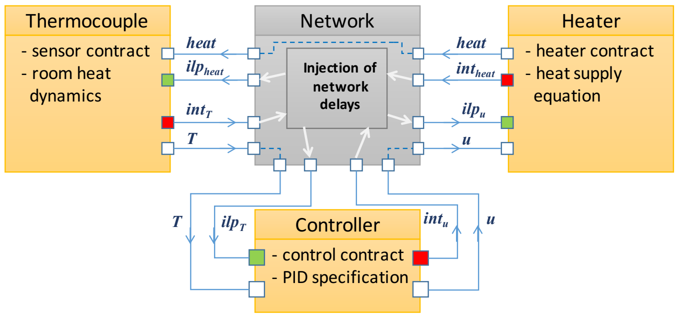

Fig. 3. Thermostat model including the network

```
MODULE Heater(u, LP_u)
VAR
  heat : real;

DEFINE
  max_heat := 10000;
  der_heat := heat + 200 * u;

DEFINE
  comp_time := 0.0;
  ILP := ((comp_time >= LP_u) ? comp_time - LP_u : 0.0);

ASSIGN
  init(heat) := 0; -- start from no heat supply
  next(heat) := (der_heat < max_heat) ? der_heat : max_heat;
```

Fig. 4. nuXmv code for the Heater

sensing and the next, therefore it has on its temperature internal latency port int_T the time needed to reach the 1 second, conditional to the time that already has elapsed since the sensing previous to the current one. In formula, the value on int_T will be given by $int_T = 1s - ilp_{heat}$.

The network decides, following a schema independent from anything attached to the network, how to feed the single participant latency ports. In our specific example, let S stand for Sensor, C for Controller and H for Heater, and $t_{X\to Y}$ for the communication time. Assuming a communication nominal latency of 0.01 s between any two participants and negligible internal latency for the Controller and the Heater (i.e., $int_u = 0$ and $int_{heat} = 0$), the network will pass on the Controller ilp_T port the value (in the nominal case):

$$\begin{aligned} ilp_T &= t_{C\to H} + t_{H\to S} + int_T + t_{S\to C} \\ &= 0.01s + 0.01s + 0.97s + 0.01s = 1.00s \end{aligned}$$

This machinery gives the network the prerogative to decide the communication times with *input latency ports* and leaves to components to decide their own running times, using *internal latency ports*. Since all the network-dependent non-functionality of the architecture has been relegated to functional ports, the dependability assessment can proceed for the non-functional side just in the same way as traditional techniques would do for the functional side.

The nominal version of the thermostat model can be shown to satisfy the top-level thermostat contract. The latency ports only end up being short negligible delays for the system, in nominal mode. Now, injections can happen on latency ports as system delays. For example we can inject faults on internal latency ports. Those would mimic accidental computation delays. We can inject faults on input latency ports. These would mimic network-related delays. After injection, the Thermocouple will use a new value for Δt in its computations, this time acquired from the network through the tainted latency ports. For example, if the *heat* message was so delayed to hit the Thermocouple after 1 second, then the Thermocouple would return a value sensed exactly after that time, and the whole system safety would be at stake, because the contract satisfactions of the single sub-components would no longer be guaranteed. We would like, eventually, to study the system under these sorts of degraded conditions.

A. Fault injection

The tool used for the dependability analysis in the present study is xSAP [6]. xSAP is a safety analysis platform devised to carry out model-based fault injection and dependability artifact construction over digital systems. It is built on top of the tool nuXmv [30] and inherits all of its model checking features. Functionally, xSAP takes a nuXmv specification of a correctly behaved system in the SMV language, an adverse top-level event nominally prevented by the system in normal conditions and a structured specification of the ways faults may trigger therein, providing, as a result, dependability artifacts such as Fault Trees and FMEA tables. The computation may use four different engines: exhaustive BDD model checking (`bdd`), incremental SAT-based (`bmc`) and SMT-based (`msat`) bounded model checking (BMC), and IC3 (`ic3`). We will employ the most convenient for each case.

To use xSAP, we need to translate our specification into nuXmv. Feasibility of translation from UML-like languages to verification engines has already been established in the literature [31]. For our specific work, we did not use any form of automation for this, but directly implemented the three services of the thermostat into nuXmv modules defined by the equations of Section III-D. We then introduced the network participant and latency ports as per the previous section. Figure 4 shows a (simplified) fragment of the code for the Heater, including the command `u` with its latency port `LP_u` and the output `heat` with the internal latency `ILP`. To be analyzed, the code must be further discretized (not shown). The complete source code, including the steps for discretization for the entire system is available in our technical report [32].

The first phase for dependability assessment is to inject faults in the model. This is done in xSAP by dedicating specialized *hook variables* to the occurrence of faults and starting nominal or faulty models correspondingly. Importantly, hooks for the timing view are all put on the network component and the functional view remains intact as it were before the fault injection. In the case at study, we will assume that the faults are transient and consist in a gradual increase in the network times of communication between participants, from 1 to 2^2, 2^5 and 2^{10} hundredth of a second (centiseconds).

In order to specify the Top-Level property around which the dependability analysis is established, we introduce a simple monitor whose task is to supervise the system and, in this particular case, trigger a timeout event if 600 seconds have elapsed without the reference temperature of the room being reached. We can ask xSAP to inject the model with faults and construct the fault tree for the property timeout. The aim is to select all minimal sets of hook variables (the so called *cut sets*) that can lead the Top-Level Event to occur.

The discretized model consists of a finite state machine with approximately 90 bits of state variables. By running xSAP out of the box, the analysis never terminates, no matter which engine is used. This is because, as in contract verification, the single modules have the time granularity of one second, whereas the top-level contract has coarser requirements, expressed over 600 seconds of time. Injections on such a system are performed at every feedback loop of evolution, thereby blowing up the search space exponentially at every loop. More specifically, the hook variables activate extra state machines that model the faults, contributing additional bits (from 6 to 9 in our case) to the state space for every iteration, quickly becoming the dominant factor. We nonetheless retrieved a partial result bounding the computation using `bmc` to the first 100 seconds only. We were able to get a fault tree in 9 minutes for $T_{ref} = 1°\mathrm{C}$, showing that a delay of 2^{10} centiseconds leads to the violation of the contract. For comparison, increasing the `bmc` limit to 110 seconds, the execution time increases to 16 minutes. Beyond this, computation becomes impractical.

Fortunately, by exploiting compositionality in refinement checking, we can avoid extending the analysis up to 600 seconds and focus instead on the single components, on the timing viewpoint. In fact, under a correct contract compositional refinement, the only way to break the top-level contract is by breaking one of the component contracts. Notice that the converse is not true, therefore the approach that we propose is conservative. We therefore developed a model for the Thermocouple, the only component specifying timing restrictions on its contract, with the precise intention of studying its timing view with respect to its contract. The point now will not be to simulate the entire system evolution, but rather check that every possible implementation deriving from the Thermocouple satisfies the contract of the Thermocouple. Compositional refinement checking (see Section III-B) then guarantees that the top-level goal is satisfied.

The application of the analysis to the Thermocouple takes practically no time. We therefore enriched the model with all delay injections on the input latency port of the Thermocouple service, ranging from 2^2 to 2^{10} centiseconds. Figure 5 shows the resulting fault tree, which demonstrates that the system is safe as long as the heat message reaches the $heat$ port of the Thermocouple in less than or equal to 2^7 centiseconds. This is consistent with the partial result obtained using `bmc`, and perfectly in line with the model: up to a delay of 2^7 centiseconds - which is our approximation of 1 second in base 2 logarithmic scale - the system behaves as if no delay was there, i.e., it is resilient. However, as delays go from 2^7 to 2^8, so that the system experiences a delay of 2^7 beyond the Thermocouple computation time, the system starts to malfunction. The effect of delays on the other services is null because they have no functional dependency on time, which

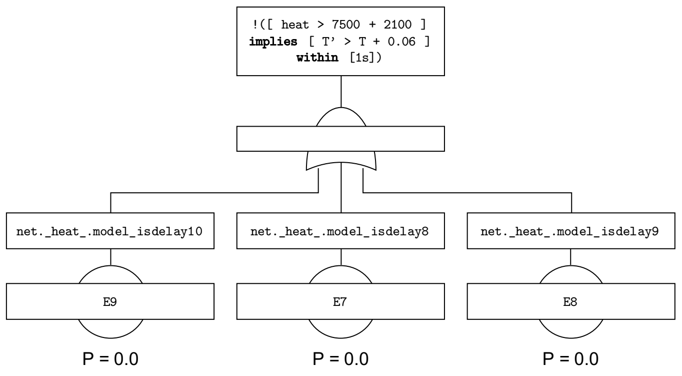
!([heat > 7500 + 2100]
implies [T' > T + 0.06]
within [1s])
net._heat_.model_isdelay10
net._heat_.model_isdelay8
net._heat_.model_isdelay9
E9
E7
E8
P = 0.0
P = 0.0
P = 0.0

Fig. 5. Fault tree for delay injections over the Thermocouple latency ports

implies their fault trees are empty (and thus not shown).

Our interpretation suggests that the system can be read very easily and its resiliency to faults determined. This is yet another evidence that the approach that we used is valuable to our aims. In particular, the output of xSAP tells us how possible combinations of unfavorable events contribute to the loss of the nominal functionality, and therefore to the violation of the system properties. This result is useful also during the design process: different implementations of a control system can be shown to be more or less resilient to unfavorable events (for instance to the likelihood and duration of network delays), and therefore help in architectural decisions.

V. Availability and Duality

In a similar fashion to latency ports, it is possible to supplement the service components with ports concerning service availability. These are boolean ports whose nominal value is positive, before injections, and are responsible for enabling the functional activity of the components. We illustrate this on a simplified version of the CAE emergency response system [33], representative of a number of case studies aimed at the organization of safety plans for the recovery from alarming situations. We use this example to model the evolutionary development of SOA-based systems in terms of participants arbitrarily leaving the playboard.

The scenario is that of an urban area, a *District*, that relies on a *Fire Station* to prevent unexpected fire explosions to burn over. The Fire Station has a number of *Fire Fighting Cars* available, 5 in our specific case, that it can send to mitigate the fire. We assume that one car is always enough to mitigate one fire explosion, and that cars are dispatched in order of their index, from 1 to 5.

The SOA structure is that of the 7 participants (the District, the Station and the 5 Cars) communicating with each other according to the agreed modes. Upon fire, the District sends a help request to the Fire Station. The Fire Station immediately gathers its available resources and sends a signal to one of its Cars to reach the place. The selected Car moves on the district, extinguishes the fire and sends an acknowledgment to the district. Then it goes back to the Fire Station sending a message of mission accomplished. This makes the Fire Station acknowledge that the car is back operative once it gets to the

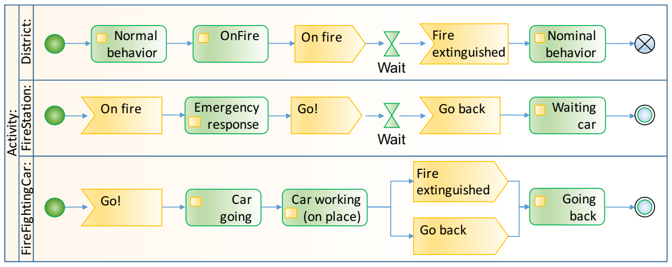

Fig. 6. Emergency response system activity diagram

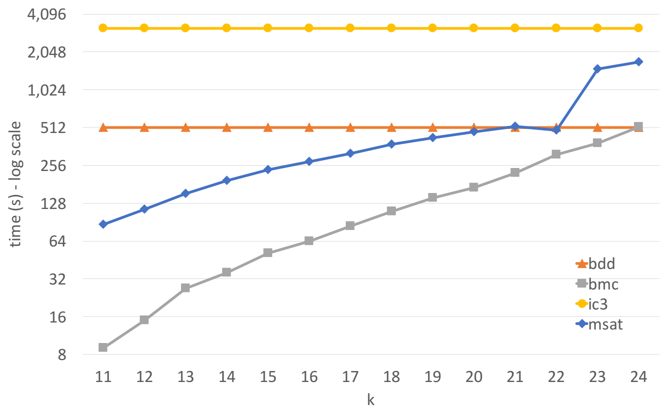

Fig. 7. Performance of xSAP on the CAE model using different engines

station. We assume that a car needs to go back to the station whenever it quenches one fireburst (e.g., to refuel, alternate firemen). The activity diagram of the CAE (with one car) is presented in Figure 6.

A. *CAE injection*

As opposed to the thermostat, which assumed everything was fixed in the topology, here there are parts of the system, either connections or components, whose presence is not to be taken for granted. The fault modalities that this system is subject to are sudden, not necessarily permanent, disappearances of services (boolean unavailability). As for the thermostat, we first create our nominal model, inject faults and see the results. Unlike the thermostat, here we do not need a specific network agent for the interactions because we do not need to attach non-functional behaviors such as timing. The introduction of the network or a similar conception is anyhow possible and would have the advantage of having all fault modalities gathered together in one entity.

We derive the nominal SMV model of the CAE from the SOA specification, assuming no faults and that everything goes smoothly according to plans. Using nuXmv, we verify that the system is always able to eventually quench all fires, with a positive result. As possible instances of system faults, cars can for instance disappear, simulating their unavailability. Car unavailability is modeled as the Fire Station failing to connect with the car to tell it to go. Another fault is the case of a car moving but never reaching the destination. Finally, we model the case of a car losing connection with the Fire Station once the fire is quenched. We represent the faults as extensions on the Car module and on the Fire Station module: unavailabilities are modeled as connections stuck at inactive.

Since Top-Level Events can only be expressed by invariant formulae in xSAP, we will need some workaround to guarantee that fires will always be extinguished, eventually, which is expressed in temporal logic. To do that we implement a monitor, which operates as a supervisor on the district. The purpose of the monitor is to let the system know when a fire is not extinguished in a fixed amount of time, by triggering a timeout variable set to 10 time units. This will provide a bounded guarantee and is expressed in our Top-Level Event specification as non-occurrence of the timeout event.

The xSAP analysis finds 10 minimal cut sets. The first 8 indicate that the failure of any one of the cars labeled in $\{1, 2, 3, 4\}$ can lead to a failure of the desirable property. For the fifth car this is not enough: the first car has to be non operative also, otherwise it could supply for Car 5 once back to the Fire Station. Two cut sets identify this scenario. Other failure scenarios are omitted, since the fault tree construction automatically excludes those subsumed by the minimal ones.

As indicated in Section IV-A, there are four engines available to xSAP for the Fault Tree construction, namely `bdd`, `bmc`, `msat` and `ic3`. The CAE example is simple enough (the state is composed of 38 bits) to let us highlight some aspects regarding performance without incurring into intricacies and model complexities as we had for the thermostat use case example. Figure 7 shows the different performance of xSAP using different engines, as a function of the BMC bound k. Here, the timeout in the monitor is set to 10 time units. The `bdd` and `ic3` methods do not depend on the bound, thus their performance is constant. For this model, the fastest algorithm up to $k = 24$ is SAT-based `bmc`, while `ic3` is the slowest.

After the bound $k = 11$ we expect fault trees to be all equivalent, because the monitor is defined using a timeout of 10, which entails executions that all monotonically subsume the first. Notice that if we did not have domain knowledge about the system, we could not claim satisfaction for any k at all. This delineates a trade-off between the use of bounded model checking and complete methods. For the CAE example, if less than $k = 23$ steps were not enough to hold confidence in the model, then it would be better to use the plain `bdd` procedure, because it would take less time.

Unfortunately, the cut-off value where choosing one method is preferable to the other cannot be known in advance, thus the trade-off can exploit little or no black-box guidance. Consider for example that increasing the timeout from 10 to 12 increases the computation time from 8 to 53 minutes for `bdd`, whereas the `bmc` computation time is still lower than 14 minutes even for a timeout of 23 (setting $k = 24$). Similar to what is usual in model checking, a BMC procedure can be used in the earlier phases of development to find Fault Trees in a very fast way and think about completeness later for self-assurance.

B. *Exploiting duality*

When constructing the fault tree, it is instinctive to set the Top-Level Event as an unwanted hazard. Dually, we can think of the Top-Level Event as a desirable state, rather than a bad one. The extension of the model are then performed by introducing welcoming positive events, and the fault tree represents desirable configurations. We apply this procedure to the CAE example. The question that we would like to answer is whether we can find a non-faulty configuration of cars that can manage a maximum of 5 fires in less than 10 time units. To answer this question, we need to lay out a nominal empty system of one District, one Fire Station and no cars. The injections in this case are not faults, but cars. The fault tree construction generates all possible minimal cut sets leading the system to the satisfaction of the desired formula. We call the dual fault tree a *suggestion tree*.

Because xSAP is allowed to pick any arbitrary value for its variables, for the result to be minimal, the tool will prefer those configurations with no fires ever bursting out in the scene. In this case, suggestion trees are of no use. We therefore need to enforce some sort of fairness by asking that in the final state — corresponding to the property violation — the district experiences all the possible fire explosions and that the monitor reaches a time count equal to the timeout. The construction finds 4 minimal cut sets. In particular, we need any two cars among those in $\{2, 3, 4, 5\}$, arbitrarily, based on when fires burst plus Car 1. Car 1 is special and needs to be there in all minimal configurations: since failures are not admitted, all cars in $\{2, 3, 4, 5\}$ are instructed to `go` only if Car 1 is not at the Fire Station.

The suggestion tree could be found using BMC methods with bound $k = 11$. It took 3.173 s using the SAT-based `bmc` procedure and 25.812 s using `msat`. Then we tried to feed the problem to the `bdd` and `ic3` engines. Not without surprise, xSAP was able to find smaller cut sets using those procedures, respectively in 4.668 s and 38.069 s. The reason for this new, different suggestion tree lays in the completeness of the `bdd` and `ic3` methods: xSAP can look at the whole state space and explore more options to make the existential paths minimal. A path could be generated such that the single car Car 1 could make its way back and forth from the FireStation to extinguish all fires independently, in 24 steps. The same fault tree could then be found using BMC-based techniques with a bound of $k = 24$. This time, however, it took much more time: around 16 s for the `bmc` procedure and more than 300 s for `msat`.

VI. Conclusions

In this paper we proposed a contract-based approach for SOCPS modeling, that combines service orientation and cyber-physicality to track both the dynamics of systems and their SOA-related non-functional aspects, such as timings and availability. Our methodology is founded on the enhancement of the service implementations with additional ports of interaction, whose value is either determined by a network participant or by the single services themselves, internally. We have shown how this is compatible with the decomposition of contracts in viewpoints and how traditional techniques for the construction of dependability artifacts can exploit this decomposition and be used for diversifying assessments.

We advised the utilization of SysML+SoaML as a problem description language, but our methodology is independent from the actual language of choice. For model-based fault injection we used xSAP, which is currently state-of-the-art in the field and has great potential beyond the illustrative examples used in this work. Moreover, it comes with the model-checking algorithms of nuXmv that can be applied on the nominal model extended with faults, supporting verification/preservation of temporal formulae before and after the model extension. Our future work includes automating the translation into SMV, combining the existing technologies into a comprehensive flow. As part of the process, we plan to extend the network with asynchronous communication and a more detailed model of faults, that may be found in SOA-specific interactions between participants in the network, and integrate suggestion trees to support the choice of components and parameters in dynamic system reconfiguration.

References

[1] M. Bozzano and A. Villafiorita, “The FSAP/NuSMV-SA safety analysis platform,” *International Journal on Software Tools for Technology Transfer*, vol. 9, no. 1, pp. 5–24, 2007.

[2] P. Vrba, V. Mařík, P. Siano, P. Leitão, G. Zhabelova, V. Vyatkin, and T. Strasser, “A review of agent and service-oriented concepts applied to intelligent energy systems,” *IEEE Trans. on Indus. Inform.*, vol. 10, no. 3, pp. 1890–1903, Aug 2014.

[3] W. Dai, V. Vyatkin, J. H. Christensen, and V. N. Dubinin, “Bridging service-oriented architecture and IEC 61499 for flexibility and interoperability,” *IEEE Trans. on Indus. Inform.*, vol. 11, no. 3, June 2015.

[4] A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone, “Taming Dr. Frankenstein: Contract-based design for cyber-physical systems,” *European Journal of Control*, vol. 18, no. 3, pp. 217–238, 2012.

[5] S. Tripakis, “Compositionality in the science of system design,” *Proceedings of the IEEE*, vol. 104, no. 5, pp. 960–972, May 2016.

[6] B. Bittner, M. Bozzano, R. Cavada, A. Cimatti, M. Gario, A. Griggio, C. Mattarei, A. Micheli, and G. Zampedri, “The xSAP safety analysis platform,” in *Proceedings of the 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Eindhoven, The Netherlands, April 2-8 2016.

[7] M. Bozzano, A. Cimatti, A. Fernandes Pires, D. Jones, G. Kimberly, T. Petri, R. Robinson, and S. Tonetta, “Formal design and safety analysis of AIR6110 wheel brake system,” in *Proceedings of the 27^{th} International Conference on Computer Aided Verification*, San Francisco, CA, July 18-24 2015.

[8] N. Looker, M. Munro, and J. Xu, “A comparison of network level fault injection with code insertion,” in *Proceedings of the 27^{th} International Computer Software and Applications Conference*, July 2005.

[9] W. Viriyasitavat, L. D. Xu, and W. Viriyasitavat, “Compliance checking for requirement-oriented service workflow interoperations,” *IEEE Trans. on Indus. Inform.*, vol. 10, no. 2, pp. 1469–1477, May 2014.

[10] L. D. Xu and W. Viriyasitavat, “A novel architecture for requirement-oriented participation decision in service workflows,” *IEEE Trans. on Indus. Inform.*, vol. 10, no. 2, pp. 1478–1485, May 2014.

[11] S. Little, D. Walter, C. Myers, R. Thacker, S. Batchu, and T. Yoneda, “Verification of analog/mixed-signal circuits using labeled hybrid Petri nets,” *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 30, no. 4, 2011.

[12] D. J. Rosenkrantz, S. Goel, S. S. Ravi, and J. Gangolly, “Resilience metrics for service-oriented networks: A service allocation approach,” *IEEE Trans. on Serv. Comput.*, vol. 2, no. 3, pp. 183–196, July 2009.

[13] F. Mhenni, N. Nguyen, and J. Y. Choley, “SafeSysE: A safety analysis integration in systems engineering approach,” *IEEE Systems Journal*, vol. PP, no. 99, pp. 1–12, 2016.

[14] A. Benveniste, B. Caillaud, and R. Passerone, “Multi-viewpoint state machines for rich component models,” in *Model-Based Design for Embedded Systems*. CRC Press, November 2009, ch. 15, p. 487.

[15] J. Ezekiel and A. Lomuscio, “Combining fault injection and model checking to verify fault tolerance in multi-agent systems,” in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, ser. AAMAS ’09, Budapest, Hungary, 2009, pp. 113–120.

[16] M. Bozzano, A. Cimatti, J.-P. Katoen, V. Y. Nguyen, T. Noll, and M. Roveri, “Safety, dependability and performance analysis of extended AADL models,” *The Computer Journal*, vol. 54, no. 5, May 2011.

[17] A. Davare, D. Densmore, L. Guo, R. Passerone, A. L. Sangiovanni-Vincentelli, A. Simalatsar, and Q. Zhu, "METROII: A design environment for cyber-physical systems," *ACM Transactions on Embedded Computing Systems*, vol. 12, no. 1s, pp. 49:1–49:31, March 2013.

[18] V. Vyatkin, C. Pang, and S. Tripakis, "Towards cyber-physical agnosticism by enhancing IEC 61499 with PTIDES model of computations," in *Proceedings of the Annual Conference of the IEEE Industrial Electronics Society*, Yokohama, November 2015.

[19] P. Derler, E. A. Lee, S. Tripakis, and M. Törngren, "Cyber-physical system design contracts," in *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*, 2013, pp. 109–118.

[20] M. Broy, I. H. Krüger, and M. Meisinger, "A formal model of services," *ACM Trans. Softw. Eng. Methodol.*, vol. 16, no. 1, February 2007.

[21] C. Farcas, E. Farcas, I. H. Krueger, and M. Menarini, "Addressing the integration challenge for avionics and automotive systems — from components to rich services," *Proceedings of the IEEE*, vol. 98, no. 4, pp. 562–583, April 2010.

[22] G. Gössler, D. Le Métayer, and J.-B. Raclet, "Causality analysis in contract violation," in *Proceedings of the First International Conference on Runtime Verification*, ser. RV'10, St. Julians, Malta, 2010.

[23] J. Puttonen, A. Lobov, and J. L. M. Lastra, "Semantics-based composition of factory automation processes encapsulated by web services," *IEEE Trans. on Indus. Inform.*, vol. 9, no. 4, pp. 2349–2359, Nov 2013.

[24] D. Cancila, R. Passerone, T. Vardanega, and M. Panunzio, "Toward correctness in the specification and handling of non-functional attributes of high-integrity real-time embedded systems," *IEEE Trans. on Indus. Inform.*, vol. 6, no. 2, pp. 181–194, May 2010.

[25] S. Bauer, A. David, R. Hennicker, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski, "Moving from specifications to contracts in component-based design," in *Fundamental Approaches to Software Engineering*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7212, pp. 43–58.

[26] M. Comerio, H.-L. Truong, F. De Paoli, and S. Dustdar, "Evaluating contract compatibility for service composition in the SeCO2 framework," in *Proceedings of the 7^{th} International Joint Conference on Service-Oriented Computing*, Stockholm, Sweden, November 24-27 2009.

[27] T. T. H. Le, R. Passerone, U. Fahrenberg, and A. Legay, "Contract-based requirement modularization via synthesis of correct decompositions," *ACM Transactions on Embedded Computing Systems*, vol. 15, no. 2, pp. 33:1–33:26, 2016.

[28] O. Ferrante, R. Passerone, A. Ferrari, L. Mangeruca, and C. Sofronis, "BCL: a compositional contract language for embedded systems," in *Proceedings of the 19^{th} International Conference on Emerging Technologies and Factory Automation*, Barcelona, Spain, September 2014.

[29] M. Bozzano, A. Cimatti, C. Mattarei, and S. Tonetta, "Formal safety assessment via contract-based design," in *Proceedings of the 12th International Symposium on Automated Technology for Verification and Analysis*, Sydney, Australia, November 3-7 2014.

[30] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta, "The nuXmv symbolic model checker," in *Computer Aided Verification*, 2014.

[31] R. Eshuis, "Symbolic model checking of UML activity diagrams," *ACM Trans. Softw. Eng. Methodol.*, vol. 15, no. 1, pp. 1–38, Jan. 2006.

[32] L. Dal Lago, O. Ferrante, and R. Passerone, "Dependability assessment of SOA-based cyber-physical systems with contracts and model-based fault injection," Dipartimento di Ingegneria e Scienza dell'Informazione, University of Trento, Technical Report DISI-17-003, February 2017.

[33] A. Arnold, B. Boyer, and A. Legay, "Contracts and behavioral patterns for SoS: The EU IP DANSE approach," in *Proceedings of the 1st Workshop on Advances in Systems of Systems*, ser. EPTCS 133, Rome, Italy, March 16 2013.

Loris Dal Lago received his Master degree summa cum laude in Computer Science from the University of Trento, Italy, in March 2015. He researched as an intern at the FBK foundation in 2012, where he gained confidence with the tool family of NuSMV. He joined ALES S.r.l. in April 2015 as an Engineer, where he has been working on model-based techniques for systems design and on safety certification. He has a background in formal logic, safety and model-based design.

Orlando Ferrante received his PhD degree summa cum laude in Electronic Engineering at "Sapienza" University of Rome, Italy, in February 2012. In April 2008 he joined ALES S.r.l. His research interests are in formal verification techniques for Embedded systems. He was involved in several European Projects (SPEEDS, SPRINT, MBAT, DANSE, MISSION) and DARPA METAII project covering formal requirements analysis. His current position in ALES-United Technologies Research Center is group lead for the area of formal methods.

Roberto Passerone (S'96-M'05) is an Assistant Professor at the Department of Information Engineering and Computer Science at the University of Trento, Italy. He received his MS and PhD degrees in EECS from the University of California, Berkeley, in 1997 and 2004, respectively. Before joining the University of Trento, he was Research Scientist at Cadence Design Systems. Prof. Passerone has published numerous research papers on international conferences and journals in the area of design methods for systems and integrated circuits, formal models and design methodologies for embedded systems, with particular attention to image processing and wireless sensor networks. He was track chair for the Real-Time and Networked Embedded Systems at ETFA from 2008 to 2010, and general and program chair for SIES from 2010 to 2015. He has participated to several European projects on design methodologies, including SPEEDS, SPRINT and DANSE, and was local coordinator for ArtistDesign, COMBEST, and CyPhERS.

Alberto Ferrari received his Dr.Eng. degree summa cum laude in Electrical Engineering from the University of Bologna, Italy. In 1994, he obtained a Ph.D. degree on integrated systems for image and speech recognition from the same university. From 1995 to 1996, he was a visiting scholar at the Electrical Engineering Department of the University of California at Berkeley. In 1998 he joined PARADES, working on design methodology and hardware software architecture for safety critical embedded systems. He taught embedded systems at the "Sapienza" University of Rome and has also been part of the ARTIST II, ArtistDesign and Hycon European Networks of Excellence on embedded system design. In 2008, he co-founded ALES s.r.l., that later becomes part of United Technologies Research Center, and he is currently serving as its General Manager. His main research interests are distributed embedded systems design and verification with focus on formal methods for safety and security.