Project Number 318772

D4.1 Analysis of state of the art in compositional reasoning for functional, safety and security formal verification

Version 1.1
29 January 2014
Final

Partners Only Distribution

Fondazione Bruno Kessler,fortiss,RWTH Aachen University,Universite Joseph Fourier,University of York

Project Partners: Fondazione Bruno Kessler, fortiss, Frequentis, LynuxWorks, The Open Group, RWTH Aachen University, TTTech, Universite Joseph Fourier, University of York

Project Partner Contact Information

Fondazione Bruno Kessler Alessandro Ciamatti Via Sommarive 18 38123 Trento, Italy Tel: +39 0461 314320 Fax: +39 0461 314591 E-mail: cimatti@fbk.eu	**fortiss** Harald Ruess Guerickestrasse 25 80805 Munich, Germany Tel: +49 89 36035 22 0 Fax: +49 89 36035 22 50 E-mail: ruess@fortiss.org
Frequentis Wolfgang Kampichler Innovationsstrasse 1 1100 Vienna, Austria Tel: +43 664 60 850 2775 Fax: +43 1 811 50 77 2775 E-mail: wolfgang.kampichler@frequentis.com	**LynuxWorks** Yuri Bakalov Rue Pierre Curie 38 78210 Saint-Cyr-l'Ecole, France Tel: +33 1 30 85 06 00 Fax: +33 1 30 85 06 06 E-mail: ybakalov@lnxw.com
RWTH Aachen University Joost-Pieter Katoen Ahornstrasse 55 D-52074 Aachen, Germany Tel: +49 241 8021200 Fax: +49 241 8022217 E-mail: katoen@cs.rwth-aachen.de	**The Open Group** Scott Hansen Avenue du Parc de Woluwe 56 1160 Brussels, Belgium Tel: +32 2 675 1136 Fax: +32 2 894 5845 E-mail: s.hansen@opengroup.org
TTTech Wilfried Steiner Schonbrunner Strasse 7 1040 Vienna, Austria Tel: +43 1 5853434 983 Fax: +43 1 585 65 38 5090 E-mail: wilfried.steiner@tttech.com	**Universite Joseph Fourier** Saddek Bensalem Avenue de Vignate 2 38610 Gieres, France Tel: +33 4 56 52 03 71 Fax: +33 4 56 03 44 E-mail: saddek.bensalem@imag.fr
University of York Tim Kelly Deramore Lane York YO10 5GH, United Kingdom Tel: +44 1904 325477 Fax: +44 7976 889 545 E-mail: tim.kelly@cs.york.ac.uk	

Contents

Document Control

Version	**Status**	**Date**
0.1	Document outline	4 March 2013
0.2	First Draft	24 May 2013
0.3	Updated Draft: typos, references, conclusion	15 June 2013
0.4	Copy-edited draft	27 June 2013
1.0	Final for EC delivery	30 June 2013
1.1	Updates requested by experts at EC review	29 January 2014

Executive Summary

This deliverable provides a thorough analysis of the state of the art in: compositional reasoning for the verification of functional properties expressed as temporal formulas; compositional dependability assessment techniques and tools; and compositional verification techniques and tools for the analysis of security properties.

Compositional techniques allow managing the inherent complexity of verification problems. In contrast to monolithical techniques, which rely on full state-space exploration and face the state-explosition problem, compositional techniques exploit *divide and conquer* principles and therefore allow inferring (correctness of) properties at system level from properties on system's components. Nonetheless, verification is inherently difficult and no magic one-for-all verification method exists. Compositionality can be effectively exploited only if specific compositional methods are developed for particular classes of properties and/or particular classes of systems.

The deliverable is organized as follows. Section 1 surveys some of the nowadays challenges and solutions for formal verification. Section 2 provides an overview of the most important families of compositional techniques for verification of functional properties. These include assume-guarantee reasoning, contract-based methods and compositional generation of invariants. Section 3 surveys the existing extensions of these techniques to extra-functional properties, that is, for systems including timed and stochastic behavior. Section 4 presents existing techniques for compositional dependability assessment. Section 5 presents techniques for verification of security properties, with focus on access control and information flow security. Conclusions are provided in section 6.

1 Introduction to Formal Verification

The aim of any design methodology is to derive implementations of systems from a set of initial requirements. Implementations should be correct, that is they should satisfy the requirements. Design methodologies start from a set of requirements that can be formalized as formulas of a logic. Requirements may be either functional or extra-functional. The former specify services that are independent of the implementation platform. The latter depend on the resources of the physical implementation such as memory, power or computing resources. Functional requirements include general properties such as deadlock-freedom, integrity of resources, as well as specific properties regarding the functional features of services. Extra-functional requirements deal with performance, dependability, and other platform-dependent properties. The specification of requirements such as autonomy and adaptivity involves mainly extra-functional properties dealing with resource management by seeking some optimum.

In general, designers follow a design flow to derive systems meeting the initial requirements. The flow ideally consists in getting successively refined models m of the designed system meeting the initial specifications ϕ. We consider that m is a transition system describing the system's behavior in some executable language. Furthermore, ϕ is a logical formula of some logic. We represent by M and Φ the sets of models and the language of formulas, respectively.

The correctness of a design flow can be formalized by using two relations:

- A satisfaction relation $\models$ relating models to formulas: $m \models \phi$ means that the model m meets the requirements specified by the formula ϕ.
- A refinement preorder $\sqsubseteq$ relating models to models: $m \sqsubseteq m'$ denotes that m refines m' and intuitively means that the behavior of model m is contained in the behavior of m' . Examples of such preorders are the various behavioral preorders used in process algebras (e.g., [138, 100, 139])

When the two relations are used in the same design flow, they should be compatible. That is refinement should preserve satisfaction of properties of Φ: $m \sqsubseteq m'$ implies that if $m' \models \phi$ then $m \models \phi$.

Correctness can be achieved either by verification or by design.

Verification methods allow to ensure that either a model m satisfies a property ϕ or that a model m refines a model m'. Verification methods may be either algorithmic or constructive. Algorithmic methods use algorithms or semi-algorithms such as model checking, abstract interpretation or bisimulation-based algorithms. Constructive techniques include axiomatic verification techniques and compositional verification techniques such as assume-guarantee. These need not be decidable.

Design methods allow a correct system to be obtained from initial system specifications. For example, to obtain a model m from a formula ϕ such that $m \models \phi$, or obtain a model m from a model m' such that $m \sqsubseteq m'$. Usually, there are additional constraints in order to avoid trivial models m, e.g. by requiring that m is maximal. Other formulations of design problems may start from mixed specifications. Synthesis methods allow a model m to be obtained from given formula ϕ and model m' such that $m \models \phi$ and $m \sqsubseteq m'$. As for verification methods, design methods may be either algorithmic or constructive. Algorithmic design methods include synthesis from logical specification and controller synthesis. Constructive techniques include algorithms and architectures. For instance a distributed algorithm specifies coordination between processes so as to meet a given global property.

As a rule, design methodologies combine both verification and design techniques. Algorithmic techniques are based on global analysis of system behavior. They are applied to small and medium size systems due to inherent complexity limitations. Constructive techniques are in principle applicable to larger systems. They make use of the structure of systems built by composition of components.

1.1 Requirements specification

Requirements characterize the expected behavior of a system. They can be expressed following two paradigms. State-based requirements specify a system's observable behavior by using transition systems. Property-based requirements use a declarative style. These requirements are usually expressed as sets of formulas in a formalism such as a temporal logic [148, 58]. A combination of the two above paradigms is necessary for enhanced expressiveness, such as in the PSL language [74]. The state-based paradigm is adequate for characterizing causal dependencies between events, e.g., sequences of actions. In contrast, the property-based paradigm is more appropriate for global properties, e.g., liveness and mutual exclusion. For concurrent systems, an important trend is towards semantic variations of state-based formalisms such as Live Sequence Charts [67].

Using temporal logics has certainly been a breakthrough in understanding and formalizing requirements for concurrent systems. Nonetheless, subtle differences in the formulation of common concepts such as liveness and fairness, which depend on the underlying time model (e.g., branching or linear time), show that writing rigorous logic specifications is not trivial. Furthermore, the declarative and dense style in the expression of property-based requirements is not always easy to master and understand.

Requirements must be consistent, correct, and complete.

Consistency means that a requirements specification does not contain contradictory requirements that would result in the requirements being *unsatisfiable*, that is, that there exists no model of the requirements. Consistency is an internal characteristic of a set of requirements.

Correctness, intuitively, is the characteristic of a set of requirements that they accurately describe a solution or a system that will achieve the goals of the endeavor for which the requirements are specified. Correctness relates to the intent of the "customer" of a development or activity, and the suitability of the requirements to satisfy the customer's needs. More rigorously, correctness implies that the requirements state a desired relationship among elements of a real-world domain that may be achieved by a suitable model of a system that interacts with that domain.

Completeness of a set of requirements has several aspects. Intuitively, a set of requirements are complete if there is no important information omitted about the needs to be met. One form of incompleteness relates to inadequately defined entities or concepts used to state the requirements or to describe the domain in which the requirements are to be applied. Another is the result of the occurrences in the requirements of details of needs explicitly left "to be determined" ("TBD"). Such incompleteness is relatively easy to detect, as it involves the closure of the information presented. More difficult, and open-ended, is the determination of the completeness of a set of requirements with respect to the environment, or the real-world domain that provides the context for the requirements and their satisfying models. It is difficult to impossible to determine *a priori* and definitively the completeness of a requirements specification in this sense. Often, only an iterative process of refinement and testing of the requirements in relation to a satisfying model in the context of the real-world domain can provide confidence in the completeness, or adequacy, of the requirements. Absolute completeness, which means that specifications describe the system exactly, has only a theoretical interest and is probably unattainable for non-trivial systems [121].

Existing requirements specification formalisms are appropriate primarily for expressing functional requirements. Nowadays, we still lack rigorous formalisms for extra-functional requirements such as security properties (e.g., privacy), reconfigurability properties (e.g., non-interference of configurable features), and quality of service (e.g., degree of jitter).

1.2 Algorithmic Verification

Algorithmic verification involves two different tasks: (1) building executable system models, and (2) developing scalable algorithms both for checking requirements and for providing diagnostics when requirements are not met. The status for each of these tasks is briefly discussed below.

Building executable models

The successful application of algorithmic verification methods requires techniques for building executable models that faithfully represent a system or an abstraction of it. Faithfulness means that the system to be verified and its model are related through a checkable semantics preserving relation. This will ensure soundness of the model, with the consequence that any property that we can verify for the model will hold for the real system.

To avoid errors in building models and to cope with their complexity, models should be generated automatically from system descriptions. For hardware verification, it is relatively straightforward to generate exact logical finite state models, expressed as systems of Boolean equations, e.g., from RTL[1] descriptions. This probably explains the strong and immediate success of model checking in the area. For software, the problem is more difficult. In contrast to logical hardware models, we need to formally define the semantics of the programming language. This may not be an easy task for languages such as C or Java, as it requires some clarification of concepts and additional assumptions about their semantics. Once the semantics is fixed, tractable models can be extracted from real software through abstraction. This allows us to cope with complexity of data and dynamic features.

Currently, we do not know how to build faithful models for systems consisting of hardware and software at the same level of detail as for pure hardware or software. Ideally, for a system consisting of application software running on a platform, the corresponding model could be obtained as the composition of models for the software and the platform. The main difficulty is in understanding and formalizing the interaction between these two types of models, in particular by taking into account timing aspects and resources such as memory and energy. In addition, this should be done at some adequate level of abstraction, allowing tractable models.

Today, we can specify and verify only high-level timed models with tools such as Uppaal [23] for schedulability analysis. These models take into account hardware timing aspects and some abstraction of the application software. The validation of even relatively simple systems such as a node in

[1]Register Transfer Level

a wireless sensor network is carried out by testing physical prototypes or by ad-hoc simulation. We need theory, methods, and tools for modeling complex heterogeneous systems [21]. Weaknesses in the state-of-the-art are also seen in standards and languages for system modeling. Efforts for extending UML to cover scheduling and resource management issues have failed to provide a rigorous basis for this. At the same time, extensions of hardware description languages to encompass more asynchronous execution models such as SystemC and TLM can be used only for simulation, due to a lack of formal semantic foundations.

Fighting state-space explosion

Today we have fairly efficient verification algorithms and tools [102, 119, 134]. However, all suffer from well-known inherent complexity limitations when applied to large systems. To cope with this complexity, we see two main avenues. The first avenue is to develop new abstraction techniques, in particular for specific semantic domains depending on the data handled by the system and on the properties to be verified. The second avenue is the convergence between model checking and abstract interpretation [62]. These two main algorithmic approaches, which have developed rather independently for almost three decades, have a common foundation: solving fixpoint equations in specific semantic domains.

Initially, model checking focused on the verification of finite state systems such as hardware or complex control-intensive reactive systems such as communication protocols. Later, research on model checking addressed verification of infinite state systems by using abstractions [60, 124]. The evolution of abstract interpretation is driven by the concern for finding adequate abstract domains for efficient verification of program properties by computing approximations of reachability sets. Model checking has had a broader application scope, including hardware, software and systems. Furthermore, depending on the type of properties to be checked, model checking algorithms may involve computation of multiple fixed points. The combination of the two algorithmic approaches can still lead to significant progress in the state-of-the-art, e.g., by using libraries of abstract domains in model checking algorithms.

1.3 Constructive Verification

Constructive techniques use system structure either to infer a global property of a system from properties of its components (verification) or to build a system meeting a given property from a given set of components (design). A component-based system is usually built from:

- a set of atomic components $\{m_i\}_{i\in I}$. The behavior of atomic components is described as models m_i from M.
- a set of glue operators $\{gl_j\}_{j\in J}$. A glue operator gl_j is a function transforming a set of components into a component. Glue operators are general composition operators: $gl_j(m_1, \cdots, m_n)$ is a n-ary operator that builds a new model by composing n models. They are abstractions of composition operators used in process algebras and can represent mechanisms used for coordinating behavior such as buses, controllers, protocols. Their semantics can be formalized by using operational semantics.

In general, a component-based system can be represented as a term $t = gl(m_1, \cdots m_k, t_1, \cdots t_w)$ involving sub-terms $t_1, \cdots t_w$ of the same form. That is, glue operators allow to construct composite components with hierarchical structure, starting from atomic components and/or already defined composite components.

Constructive verification includes axiomatic verification techniques [99, 128] and compositional verification techniques such as assume-guarantee [1, 135], contract-based [136, 30, 32, 153, 91, 66, 22, 56, 61, 31] or compositional invariant generation techniques [27, 28].

In general, compositional verification uses divide-and-conquer approaches for inferring global properties of a system from the properties of its components. Thus for property verification, it uses compositionality rules of the form: $m_i \models \phi_i$ for $i = 1 \cdots k$ implies $gl(m_1, \cdots m_k) \models \tilde{gl}(\phi_1, \cdots \phi_k)$.

That is for a composite component global properties are inferred as the composition of properties of atomic components. The operator $\tilde{gl}$ is somehow an extension of gl to formulas. Existing results consider instances of this rule where gl is a parallel composition operator and $\tilde{gl}$ is conjunction.

For refinement verification compositionality rules are of the form: $m_i \sqsubseteq m'_i$ for $i = 1 \cdots k$ implies $gl(m_1, \cdots m_k) \sqsubseteq gl(m'_1, \cdots m'_k)$. The rule simply says that the refinement preorder is a pre-congruence for the parallel composition. This is often the case for process calculi [100, 138].

2 Compositional Verification for Functional Properties

The essential part of compositional verification is that the correctness of a system property is derived from properties of its constituent components combined with properties of the parallel composition [70]. This requires an assertional approach where the system and its components are annotated by means of assertions, i.e. formulas over the observable variables. Therefore, the compositional verification that a system m satisfies an assertion ϕ involves two main steps:

1. *Basic verification techniques* to prove the assertion when m cannot be decomposed.
2. *Compositional verification techniques* to prove the assertion when m is the composition of n components $m_1, \ldots, m_n$.

The compositional verification techniques are typically based on proving that each component m_i satisfies some local assertion ϕ_i and on checking some proof obligations on the assertions. This amounts essentially to prove the validity of a finite number of implications involving the assertions $\phi_1, \ldots, \phi_n$ and ϕ. The compositional verification rule has therefore the following general form:

$$\frac{\begin{array}{c}\text{for all } i, 1 \leq i \leq n, m_i \models \phi_i \\ op(\phi_1, \ldots, \phi_n, \phi)\end{array}}{m \models \phi}$$

where m is the composition of $m_1, \ldots, m_n$, and *op* depends on the assertion language, but may be in some cases given by the logical implication $\bigwedge_{1 \leq i \leq n} \phi_i \rightarrow \phi$.

2.1 Assume/Guarantee Methods

When reasoning compositionally, it is often necessary to assume the correctness of a component m_j to prove the property of m_i. This leads to *assumption/guarantee reasoning*, where each assertion Φ is structured in an assumption α and a guarantee γ, meaning that the model satisfies the guarantee γ, provided α holds.

For example, in verification of sequential programs, the well-known notation of a Hoare triple $\{\alpha\}m\{\gamma\}$ means that any terminating execution of the program m starting from a state satisfying the precondition α must terminate in a state satisfying the postcondition γ.

These methods have been extended also to handle concurrent programs, handling both synchronous message passing and shared-variables concurrency. While for sequential programs, the observable points are the initial and final states, assertions for concurrent programs must predicate over sequences/traces of synchronization and communication points. Therefore, the assumptions and guarantees are given in terms of the temporal counterpart of preconditions and postconditions.

When m_i depends on the correctness of $m_j \models \phi_j$ and m_j depends on the correctness of $m_i \models \phi_i$ there is an apparent circularity that requires to specify the assertion in a careful manner to make the composition rule hold. The standard method [1] exploits the induction principle proving that $m_i \models \phi_i$ at time t assuming that $m_j \models \phi_j$ only up to time $t-1$, and vice versa. This has been extended by McMillan to handle also liveness properties [135].

2.2 Contract-based Methods

Contracts have been first defined in the context of object-oriented programming by Meyer [136]. For software programs, assumptions and guarantees are represented respectively by preconditions and postconditions of functions: preconditions define the assumptions that the function caller must satisfy at the entry point of the function; postconditions define the guarantee that the function provider must satisfy at the exit point. Other used assertions are class and loop invariants. In concurrent programs, the interaction between service clients and providers is more complex and requires to assert over execution traces.

Contract-based design is now applied also to embedded systems [30, 32, 153, 91, 66, 22, 56, 61, 31, 160] to structure the component properties into contracts. A contract specifies the properties assumed to be satisfied by the component environment (assumptions), and the properties guaranteed by the component in response (guarantees).

Given a component S, a contract for S is a pair $\langle A, G\rangle$ of assertions over the variables V_S at the interface of S. A and G represent respectively an *assumption* and a *guarantee* for the component.

In some of these works (e.g., [66, 61]), contracts distinguish between assumption and guarantee only for enabling assume-guarantee reasoning. Thus the semantics of a contract is simply the implication “if the assumption holds, also the guarantee holds”. In contract-based design, both in the seminal work of Meyer [136] and in the recent applications to embedded systems, contracts represents two distinct properties, one for the environment of the component and one for the component itself. If the environment does not satisfy the assumption, then the architecture is not correct.

Let $C = \langle A, G\rangle$ be a contract of S. Let I and E be respectively an implementation and an environment of S. We say that I is a implementation satisfying C iff $I \models A \rightarrow G$. We say that E is an environment satisfying C iff $E \models A$. We denote with $\mathcal{I}(C)$ and with $\mathcal{E}(C)$, respectively, the implementations and the environments satisfying the contract C.

Two contracts C and C' are *equivalent* (denoted with $C \equiv C'$) iff they have the same implementations and environments, i.e., iff $\mathcal{I}(C) = \mathcal{I}(C')$ and $\mathcal{E}(C) = \mathcal{E}(C')$.

We say that a contract C' refines a contract C ($C' \leq C$) iff $\mathcal{I}(C') \subseteq \mathcal{I}(C)$ and $\mathcal{E}(C) \subseteq \mathcal{E}(C')$. This notion has been extended in [56] to consider the refinement along a structural decomposition taking into account the contracts of the subcomponents of a component.

AutoFocus3 (AF3) supports contract-based methods for formal verification. The specification of the contracts is done on the ports of components using predicates as sketched in Figure 1.

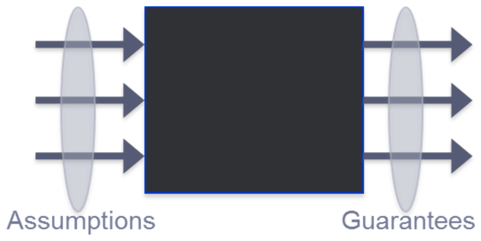

Figure 1: Assumption and Guarantees Specification in AF3

Assumptions are assumed to be invariants on input ports - or in other words - assumptions define what a component can expect from the environment. Guarantees are assumed to be invariants on output ports - or in other words - guarantees define what a component promises to the environment.

AF3 supports several verification steps for contract-based reasoning. One step of the verification is demonstrated in Figure 2. It shows three components A, B and C, which are connected to each other via communication channels. The inputs and outputs can be specified to provide certain guarantees or to expect certain assumptions. The specification enables AF3 to do external (black-box) checks, i.e. to prove if the guarantees of A and B fulfill the assumptions of C. This allows reasoning about the sub-systems interaction based on contracts at a higher level of abstraction than sub-systems implementation.

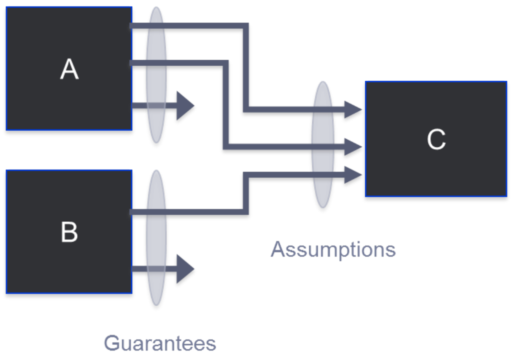

Figure 2: External (Black-Box) Compatibility Checks

Another step in AF3 is the internal view, the verification of the contract's implementation. This step verifies if the implementation of a component satisfy the guarantees based on its assumptions from the environment. This is shown in Figure 3.

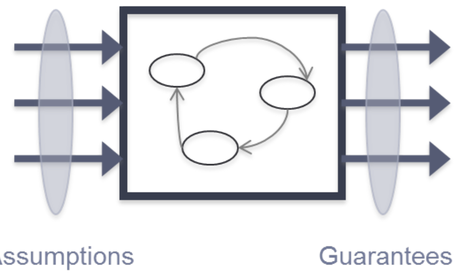

Figure 3: Internal view: Verification of a contract's implementation.

Finally higher level checks can be performed based on the previous steps, as shown in Figure 4. Internal views can be used to check if the sub-systems fulfill their specification (pre-/postcondition). External views can check that neighboring sub-systems are compatible (post-/ precondition). The combination of both allows the reasoning on higher level properties of the system.

At the moment contract-based reasoning in AF3 does not support loops on one component. I.e. a component that has a feedback loop to itself cannot be checked. Also specification of contracts, which are not based on predicates, e.g. temporal logic, is not yet part of AF3.

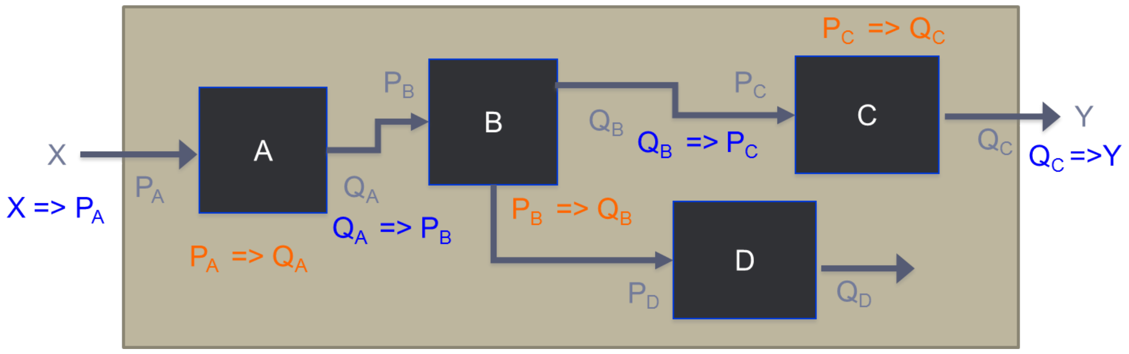

Figure 4: Internal and external views enable higher level verification.

2.3 Invariant-based Methods

A compositional verification method based on automatic generation of invariants has been presented in [29, 27, 28] and implemented in the D-Finder tool [26]. The method is based on the following rule:

$$\frac{\{m_i \models \Box\Phi_i\}_i \quad \Psi \in II(\|_\gamma\{m_i\}_i, \{\Phi_i\}_i) \quad (\bigwedge_i \Phi_i) \wedge \Psi \Rightarrow \Phi}{\|_\gamma\{m_i\}_i \models \Box\Phi}$$

The rule above proves invariance of property Φ for systems obtained by using an n-ary composition operation $||$ on a set of components $(m_i)_i$ parameterized by a set of interactions γ. The rule is based on the computation of a global invariant that is the conjunction of local invariants Φ_i of constituent components m_i and an interaction invariant Ψ. The latter expresses constraints on the global state space induced by interactions between components. In [29, 27], it has been shown that Ψ can be computed automatically from the interactions γ and finite-state abstractions of the system to be verified. That is, we provide an effective procedure, denoted II in the rule, which allows to compute interaction invariants from finite state abstraction of the components m_i with respect to its local invariant Φ_i. The method has been implemented in the D-Finder tool and specialized for checking deadlock-freedom on models described in the BIP (Behavior, Interaction, Priority) framework [21].

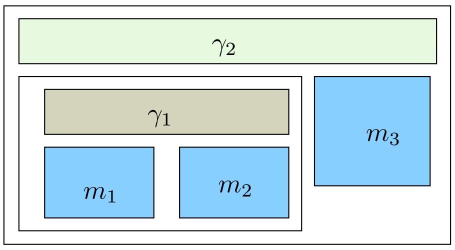

Figure 5: Incremental Design

To increase scalability, the previous method has been further made incremental. The key idea is to reuse the already computed invariants of the constituents of a composite component in order to compute global invariants. This requires the formalization of the construction of hierarchically structured composite components. For instance, Figure 5 shows a hierarchical composite component obtained as the composition of two components by using a set of interactions γ_2, and where one of these components is on its own the composition of two components by using the set of the interactions γ_1. The computation of invariants of the component $\gamma_1(m_1, m_2)$ is obtained by combining invariants of the atomic components m_1 and m_2 with an interaction invariant characterizing the restriction of the interactions γ_1 on the product of the composed components. Following the incremental construction process, invariants of $\gamma_2(m_3, \gamma_1(m_1; m_2))$ are obtained by combining invariants of $\gamma_1(m_1, m_2)$ and invariants of m_3 with an interaction invariant induced by the application of γ_2 and without flattening the composition. This method has been described in [28] and integrated to D-Finder.

2.4 Relation with D-MILS requirements

A significant part of the identified user requirements on both case studies belong to the class of functional properties. These properties can be expressed either as temporal logic formula or state invariants. Some representative examples are provided below, for both of the case studies (extracted from. [63],[64]):

SMG_SA.2 In case of a power outage, the micro grid shall switch to an island mode

SMG_SA.4 If the voltage of the connection between the micro grid and smart grid becomes higher than 10% of the nominal value, the smart micro grid island mode shall be activated [more]

SMG_SA.6 If prosumer island mode is active and a high consuming device is turned on, the power of the corresponding power socket should be turned off in the switch box [more]

SMG_SA.11 In case of island mode, full batteries and a higher production than consumption - the production units shall reduce their production

SMG_SO.14 Command events, which trigger certain actions of devices, shall not be lost [more]

SMG_SF_TA.13 The TSF shall allow user-initiated termination of the user's own interactive session

FVS_BS_R.2 Switch over triggered by loss of service components: Loss of service components or any part of them shall trigger a switch over to the redundant switch half and shall be indicated on the management system.

The properties above could be in principle handled using compositional verification methods like the ones presented earlier in this section. Most of these properties are *safety* properties (in the verification terminology) and perfectly fit the scope of contract-based, assume-guarantee and/or invariant-based methods. However, practical limitations exist and compositional verification of functional properties has been listed as a desirable technical requirement for the verification framework to be developed and used in D-MILS (cf. [65, Sect. 5]).

CV-WP4.10 The verification framework shall provide compositional reasoning for functional properties

Improvements are expected for improving scalability and applicability to larger classes of models/systems. For invariant-generation methods implemented in D-Finder[26] we are seeking techniques to improve the handling of data variables. Actually, D-Finder is able to handle only models where interactions (communications) are restricted to pure control synchronizations. That is, in this tool data are handled locally, at the level of atomic components, using predicate abstraction. At the time being, no support exist for handling data transfer (data communication) between components.

As for the contract-based methods implemented in OCRA [53], they need to be extended in order. first. to manage to prove effectively the compositional rules, and second, to handle data types such as functions and arrays. In fact, the decision problems underlying the compositional verification are undecidable due to presence of data variables with infinite domains such as real and integer numbers. This issue is exacerbated if we consider also functions and arrays. However, these data types seem necessary to abstract elements such as cryptography functions and memory management. We are working on algorithms based on SMT solvers to handle effectively the verification of systems modeled using theories such as integer, reals, uninterpreted functions, and arrays. We already made a good progress in the verification of invariant and safety properties (still in the verification terminology) [54]. We are extending such results to the verification of temporal properties, to deal with the theories that are necessary for D-MILS, and to perform compositional verification using these algorithms.

3 Compositional Verification of Extra-Functional Properties

Compositional verification has been originally developed for basic modeling formalisms such as untimed transition systems, simple concurrent programs, and process algebras. An important research effort is made nowadays towards extending/adapting compositional techniques to richer modeling paradigms. The following sections survey the associated verification methods for two such paradigms, namely for timed/hybrid systems and for probabilistic systems. The former allows for quantitative modeling and analysis about timed phenomena while the latter considers quantitative modeling of uncertainties using probabilities and probabilistic distributions.

3.1 Methods for Timed and Hybrid Systems

Hybrid Automata [95] (HA) are a well known framework used in formal verification to model the discrete and the continuous evolution of hybrid systems.

HA extends a finite state machine (FSM) with continuous components, modeled using real-valued variables. Each state of the FSM, called location, defines the so called *flow conditions*, which describe with differential equations the evolution of the continuous variables over time. For example, the flow condition $\dot{x} = v, \dot{v} = a, \dot{a} = 2$ describes an uniformly accelerated motion. Each location also defines invariant conditions over continuous variables, which must be satisfied whenever the system is inside the location. The discrete transitions of a HA defines how the system changes the current location. Each transition is associated to a *jump condition*: it is a formula over the continuous variables at the current and at the next step, which describes the guards and the effects of the transition. Moreover, transitions are labeled with a symbol to enable synchronizations among automata. We call Σ the set of events which label transitions.

A state of the HA is a tuple $\langle q_i, s_i \rangle$, where q_i is a location and s_i is an assignment to the continuous variables. A run $\sigma = \langle q_0, s_0 \rangle \xrightarrow{a_1} \ldots \xrightarrow{a_n} \langle q_n, s_n \rangle$ of a HA is a sequence of states that the HA can visit. The first state satisfies the initial condition of the HA, while all the states satisfy the invariant condition of the corresponding location. Each state $\langle q_i, s_i \rangle$ moves to the next state $\langle q_{i+1}, s_{i+1} \rangle$ either with a discrete or a continuous transition, depending on the event a_i: if $a_i \in \Sigma$ then there is a discrete transition, otherwise $a_i \geq 0$ means that there is a continuous transition where the time elapsed is equal to a_i. A discrete transition is fired if its *jump* condition is satisfied. A continuous transition updates the continuous variables according to the flow condition in the current location and the amount of time elapsed, while it keeps the location unchanged.

Hybrid automata have been classified with respect to their continuous dynamics, jump conditions and invariants. An important class of hybrid automata are *Timed Automata* [10], which restricts the dynamics of the continuous variables to 1 (i.e. the continuous variables are clocks) and the possible constraints to clock constraints (i.e. clocks may be only compared with constants). Another important class are *Linear Hybrid Automata (LHA)* is an HA where all the conditions are Boolean combinations of linear inequalities and the flow conditions contain variables in $\dot{X}$ only. The reachability problem is already undecidable [11] for this class of automata.

Real hybrid systems are constituted by several components. A *Network of Hybrid Automata* $\mathcal{H} = H_1 \parallel \ldots \parallel H_n$ is a set of HA. Each automaton H_i of the network moves asynchronously with transitions labeled with a local event (i.e. a symbol which is only in the alphabet Σ_i). Instead, automata synchronize on transitions labeled with a shared event (i.e. a symbol which is contained in the alphabet of more automata). Synchronization is used to model the communication between the automata of the network. There exist other formalisms to compose HAs such as *Hybrid I/O Automata* [126].

Monolithic Verification

Verification of hybrid systems has been performed applying different approaches (see [9] for a recent survey).

A common technique consist of computing the set of the reachable states to prove a safety property. These approaches start from the set of the initial states and then they iteratively compute the set of states reachable in a step (either discrete or continuous). This operation is repeated until a fixed point is reached, proving the property, or as soon as the property is violated. Note that, in the case of hybrid system, this kind of analysis may not terminate.

In the context of timed automata, UPPAAL [25] performs a semi-symbolic reachability analysis. The set of the reachable states is computed visiting the graph of the automaton, computing all the possible clock values reachable in each location. The clock values are computed and represented with efficient data structures, called difference bound matrices (DBM). In order to fight the state-explosion problem, alternative approaches [40, 180, 33, 143] differ in the exploration of the discrete state space (i.e. location of the automata) which is “fully symbolic”. These techniques may be effective if the discrete state space is large [111].

Similar approaches have also been developed in the context of hybrid systems. For hybrid system the main difficulty lies in the efficient computation of the time successor (or predecessor) operation and in the representation of the set of the reachable states. The tool HYTECH [96] deals efficiently with *linear hybrid automata*, representing the set of reachable states with polyhedra. Other approaches [84, 85, 93, 51, 14] are able to verify *linear hybrid systems*. Since the computation of the continuous evolution of the system involves a matrix exponential, it is difficult to compute the set of reachable states. SPACEEX [85] approximates the reachable set of states using different representations, such as polyhedra [84] or support functions [93]. Approximate reachability has been applied also to non-linear hybrid systems, for example in [154, 142, 172].

The model checking problem for hybrid system can be reduced to the model checking problem of infinite-state transition systems [167, 15, 2, 112]. Then, the infinite-state transition system is analyzed with techniques such as Bounded Model Checking (BMC) [35] or K-induction [164]. Both the approaches rely on solvers for the Satisfiability Modulo Theory (SMT) problem, which consists of checking if a first-order formula is satisfiable in a specific theory (e.g. in the theory of linear rational arithmetic). However, the approach is limited to some subclasses of *linear hybrid systems* [55].

Safety property can be verified by techniques which use deductive verification [146, 149, 73, 169]. The main idea is to prove that a property P is inductive for a system M, which means that P holds in the initial states of M and that, assuming that P holds in the current state, it also holds in all the states reachable after a transition in M. Usually, it is necessary to infer several invariants of M to be able to prove that P is inductive. *Barrier certificates* [149] are an example of such invariants for hybrid systems. A *barrier certificate* is a condition that ensure, in each location of the hybrid automaton, the separation of the reachable states of the system from the set of "bad" states. Tools such as KEYMAERA [146] or METITARSKI [73] are theorem provers which implement deductive technique to prove properties on hybrid systems.

Finally, abstraction [12, 161, 59, 78, 68] have been extensively applied in the verification of hybrid systems in order to mitigate the complexity in the verification. These techniques compute a new system which is an abstraction of the hybrid system under analysis. The main feature of the abstraction is that if a property holds in the abstract system then it also holds in the hybrid system. Then, the verification process may be carried out more efficiently in the abstract system. Since abstractions over-approximate the original system, they may be too imprecise to prove a specific property, generating spurious counterexamples. While the counterexample exists in the abstract system, it does not exist in the original hybrid system. Several approaches [59, 78] analyze spurious counterexamples in order to increase the precision of the abstraction, eventually proving the property. A popular example of abstraction is *Predicate Abstraction* [12, 92], where the infinite state space of the system is partitioned by a finite set of predicates. There exists other kind of abstractions for hybrid systems [161, 171, 68]. They are suitable for different kinds of systems and generate different kinds of abstractions.

Compositional Verification

Assume-guarantee style reasoning has been applied in the context of hybrid system [83]. In this work, the system is represented by the parallel composition of hybrid automata. An automaton may be abstracted by another automaton which simulates it. The abstract automaton is then used to prove an assumption in the assume-guarantee rules in order to compositionally prove a property. The work describe approaches to compute and check simulations of hybrid automata and a novel circular assume-guarantee rule.

In KEYMAERA [146, 147] hybrid systems are represented as *Hybrid Programs*, an extension of regular programs with continuous evolutions. The verification problem is expressed in the *Dynamic Logic* formalism, which include the specification and the hybrid program. Then, *Dynamic Logic* is verified compositionally.

3.2 Methods for Probabilistic Systems

Probabilistic systems are transition systems where the taking of certain transitions is governed by chance. We first enumerate classes of formalisms to express probabilistic models or properties and note that only few have been used in compositional model checking. We categorize probabilistic systems as having a continuous or discrete notion of time, and as having or lacking nondeterminism:

	without nondeterminism	**with nondeterminism**
discrete time	Discrete-time Markov Chains (DTMCs)	Markov Decision Processes (MDPs), Probabilistic Automata (PA)
continuous time	Continuous-time Markov Chains (CTMCs)	Continuous-time Markov Decision Processes (CTMDPs), Interactive Markov Chains (IMC)

In a DTMC, the transitions are labeled with a probability. In a state s, a transition $s \xrightarrow{p} s'$ to state s' is chosen with probability p. Obviously, the sum of the probabilities of all outgoing transitions for each state, must equal one.

In MDPs, each transition is labeled with an action and a probability. In state s, an action a is chosen nondeterministically and then a transition $s \xrightarrow{a,p} s'$ to state s' is chosen with probability p. The sum of the probabilities of all outgoing transitions for each state with the same action label, must equal one.

The related PA (Segala [163]) allow for nondeterministic choice of a probability distribution for the same action. There are alternating action-labeled and probability-labeled transitions. In state s, an action a is chosen nondeterministically. A transition $s \xrightarrow{a} \mu$ to a probability distribution μ is chosen nondeterministically and execution instantly jumps to a state s' with probability $\mu(s')$. The MDP is a special case of PA, with only one possible probability distribution per action.

The continuous-time variants of these formalisms, the CTMCs and CTMDPs, label transitions with a rate instead of a probability. For a CTMC in state s, a transition $s \xrightarrow{\lambda} s'$ to state s' is taken after an amount of time taken from an exponential distribution with parameter λ. If there are multiple outgoing transitions, the earliest is taken in a random fashion. Similarly, CTMDPs are MDPs with rates instead of probabilities and IMCs (Hermanns [97]) are PA with rates instead of probabilities.

(Non-compositional) model checking of CTMCs has been well-studied, but the verification of CT-MDPs is still at its infancy.

Properties

We categorize specification logics as linear or branching, and as having a discrete or continuous notion of time.

	linear	**branching**
discrete time	Probabilistic modal μ-calculus (pLμ)	
	Linear-time Temporal Logic (LTL), ω-automata	Probabilistic Computation Tree Logic (PCTL)
continuous time	Timed Automata (TA)	Continuous Stochastic Logic (CSL)

For the linear-time properties, the formalisms are the same as in the non-probabilistic setting, only with a different model checking question. For example, "what is the probability that a path through a given model satisfies a given LTL-formula?", or "–is accepted by a given ω-automaton?".

In PCTL (Hansson and Jonsson [94]), the operator $\mathcal{P}_I$, where I is a sub-interval of $[0, 1]$, replaces the path quantification operators in CTL. The formula $\mathcal{P}_I(\varphi)$ expresses that the path-formula φ is satisfied in a state, with a probability in interval I. In models with nondeterminism, the operator additionally requires a quantification over schedulers, which are oracles that resolve nondeterminism.

CSL is similar to PCTL, with time intervals added to the until-operator $\psi\mathcal{U}\varphi$ and next-operator $\bigcirc(\varphi)$, meaning a state satisfying formula φ is reached within that time interval, and an additional operator $\mathcal{S}_I(\varphi)$, meaning the path-formula φ holds in a steady state, with a probability in interval I.

The probabilistic modal μ-calculus (pLμ) (Huth and Kwiatkowska [103]) has the same syntax as the modal μ-calculus (Lμ) with a different semantics. The solution of a pLμ-formula φ gives a probability for each recursion variable and state, instead of a set of states for each recursion variable. This was shown to be insufficient to express PCTL and recently extended with the independent product operator $\odot$ by Mio and Simpson [140], where $p \odot q$ is defined as $p + q - pq$. As shown in that paper, probabilistic modal μ-calculus with independent product (pL$\mu^{\odot}$) is at least as expressive as PCTL.

Compositional Verification

The combination of probabilistic models and compositional verification has not been broadly studied.

Probabilistic models which exclude nondeterminism are not studied for compositional verification. Two important reasons for this are that interleaving re-introduces nondeterminism [98], and that models excluding nondeterminism can trivially be expressed in a formalism including nondeterminism.

There are some interesting results for DTMCs and for PA. The specification formalisms that are used for compositional model checking in probabilistic settings are LTL and Büchi-automata, a class of ω-automata in which LTL-formulae can be expressed. No work is known on compositional verification for branching or continuous-time logics in a probabilistic setting, though the authors of [72] claim their principle can be generalized to branching logics.

All compositional verification efforts for probabilistic systems use assume-guarantee reasoning extended with some probabilistic information. For LTL-formulae, the assumptions and guarantees hold for each run with a given probability.

Kwiatkowska *et al.* [114] perform compositional model checking using assume-guarantee reasoning, where a probability is added to the assumptions and guarantees with which it holds. The assumptions and guarantees take the form of LTL-formulae. Model checking was realized by a reduction to multi-objective probabilistic model checking. Multi-objective probabilistic model checking takes a set of LTL-formulae which all have to be satisfied with a certain probability bound. It has been shown in [77] that this can be solved using linear programming.

Experiments on two large case studies ([116, 115]) with an implementation with the PRISM model checker yield positive results. This approach is faster in most cases, less memory intensive and yields quite tight probability bounds [114].

Delahaye, Caillaud and Legay [72] use a notion of assume-guarantee satisfaction where at least a certain fraction of states in each run satisfying the assumption, should also satisfy the guarantee. The assumption and guarantee are both expressed as a Büchi-automaton. They generalize this notion of satisfaction with a *discount factor* to emphasize states that occur earlier in each run. In the probabilistic case, they extend this with probabilities for the guarantee to hold. Compositional verification is used to calculate bounds on the probabilities for the combined contract on the parallel system.

There is no working implementation of this approach.

Some additional work has been done using refinements and assume-guarantee reasoning. In general terms, a system S *refines* another system S', denoted $S \preceq S'$, if S assumes less and guarantees more than S'. Concretely, this usually means that the behaviors of S' are contained in the behaviors of S. Certain properties are closed under refinement, meaning one can check a given property on the more abstract system and conclude its (non-)validity on the refined system. Delahaye, Caillaud and Legay [72] provide bounds for the probability that a run in a refined system satisfies a given path property, given these probability bounds for the unrefined system. De Alfaro, Hensinger and Jhala [69] have provided compositional deduction rules for determining if one probabilistic system refines another.

Compositional methods to determine if a composite system refines another system have been successfully applied to probabilistic systems by Komuravelli, Păsăreanu and Clarke [113]. The idea is to iteratively refine the components (in the rule below, A is being refined), based on generated counterexamples, until it can be shown that no counterexamples exist. The original question of whether their composition is a refinement of P, can then be answered using the rule:

$$\frac{S_1||A_1 \preceq P \qquad S_2||A_2 \preceq A_1 \qquad \dots \quad S_n \preceq A_{n-1}}{S_1||\dots||S_n \preceq P}$$

This procedure is fully automated and has been implemented in a tool, yielding mostly positive results on the examples of [79], when compared to monolithic verification.

Mio and Simpson [140] introduced Markov proofs, which use probabilistic branching in the proof structure. This allows circular reasoning, as long as every loop is unfolded infinitely often with probability 0. This can be applied to provide proofs for assume-guarantee relations where the assumptions and guarantees hold with a certain probability. In their paper, Mio and Simpson specifically applied this to pLμ-formulae and a particular process algebra, but claim that it can be extended to any property formalism and process algebra. There is no working implementation of this compositional proof system.

3.3 Relation with D-MILS requirements

Many of the identified user requirements on both case studies belong to the class of extra-functional properties. Most of them consider timing and performance properties (e.g., latencies, delays) and consequently, they can be handled using specific techniques and formalisms for modeling and analysis of timed systems. In addition, some of the user requirements consider also hybrid aspects (power, temperature requirements) and henceforth they require hybrid models and specific analysis. Some representative examples extracted from [63, 64] are provided below:

SMG_SA.3 Switching to island mode shall to be accomplished in less than 20 ms

SMG_SA.8 Every battery component shall not be overloaded [more]

SMG_SA.10 The battery temperature should be within the bounds specified in the documents [more]

FVS_BS_R.1 PTT delay: The delay within the system between detecting PTT selection and the activation of the line interface shall have a value of less than 10ms

FVS_BS_R.3 Squelch delay: The delay between the Squelch signal arriving at the Operator Position from the receiver to the audio activation at the operator position loudspeaker/headset, shall have a value not greater than 40ms

FVS_BS_R.5 Start-up after power loss: The system shall be able to start-up to a fully operational state (i.e. all given redundancy working) after a complete or partial power loss without additional maintenance activity (e.g. reset of operator position)

FVS_PA_R.1 Channel Switch-Over: A channel switchover shall not cause an interruption of the audio signal of more than 100 milliseconds

FVS_PA_R.6 Voice delay: The voice delay for ground transmission components shall be a maximum of 130 ms

FVS_PG_R.1 Call setup time: Communication shall be established between controllers within 2 seconds

Compositional verification for extra-functional properties has been identified as a desirable feature for the verification framework developed in the context of D-MILS (cf. [65, Sect. 5]). The two families of models and associated properties are explicitely considered:

CV-WP4.12 The verification framework shall provide compositional reasoning for performance properties

CV-WP4.13 The verification framework shall provide compositional reasoning for timing properties

Compositional verification for timed and/or hybrid models is actively developing nowadays. In D-MILS, we plan to extend the D-Finder method for generation of invariants for timed systems. An idea currently investigated relies on the use of additional *history clocks* as a mean to track the local timing of actions in the model and to enforce global synchronization constraints. That is, several history clocks involved in the same synchronization remain equal until one of them is being reset. This simple observation allows to desynchronize the components and to perform a local analysis on them, while recovering later the global timing constraints based on the analysis of interactions. To some extent, this method can be applied for some classes of hybrid systems, where the dynamics of the continuous variables can be evaluated locally.

Moreover, we plan to extend OCRA to effectively perform compositional verification of temporal properties with real-time constraints. In fact, the current support is limited to BMC techniques that try to falsify the properties, rather than proving that they hold. We are currently combining the algorithm presented in [54] with the technique proposed in [57] in order to prove temporal properties on hybrid and timed systems. The idea is to extend this approach to handle properties with real-time constraints and to use it for compositional reasoning.

4 Compositional Dependability Assessment

Dependability assessment is concerned with the problem of assessing system behavior in degraded situations, that is, when some parts of the system are not working properly, due to malfunctions, and ensure that the system meets the safety requirements that are required for its deployment and use. A dependable system can be defined as a system such that reliance can justifiably be placed on the service it delivers [118], and it covers, in a broad sense, issues related to system safety, reliability and availability.

Typically, safety requirements stating the conditions under which systems must remain operational are defined along with the other system requirements. The safety assessment activities have the goal of identifying all possible hazards and their causes. Safety analysis activities produce artifacts that represent the combinations of failures causing the violation of safety requirements, the effect of failures on the system and their probabilistic evaluation.

Traditional safety analysis activities include Fault Tree Analysis (FTA) and Failure Modes and Effects Analysis (FMEA) [177, 178]. Fault Tree Analysis can be described as a deductive, analytical technique, whereby an undesired state (the so called *top (level) event* (TLE)) is specified, and the system is analyzed for the possible chains (called *cut sets*) of *basic events* (e.g., system faults) that may cause the top event to occur. A fault tree makes use of logical gates to depict the logical interrelationships linking such events. The fault tree model is not in itself a quantitative model, but rather a qualitative model that can be evaluated quantitatively (e.g., to determine the probability of a safety hazard). FMEA is an example of an inductive technique. It starts from the identification of the failure modes of the components of the system under investigation and, using forward reasoning, assesses their effects on the complete system.

Another line of research is provided by the the so-called *Fault Diagnosis* theory introduced in [158, 159] for discrete-event systems. The aim of fault diagnosis is to identify when the system safety requirements have been violated, possibly with some bounded delay. Once such a violation has been detected, it is possible to switch to a degraded mode. In the framework of Fault Diagnosis (FD), the system functional model is extended with *faults* (user-defined) which are special events. Those faults are not directly observable at run-time, but they may be detected or inferred by partially observing or monitoring the events in the system. If faults can be detected on a partially observable system, the system is said *diagnosable*. In this case, a finite state device called a *diagnoser* can be constructed to raise an alarm when a fault has been detected.

FD has been extensively studied for discrete-event systems both in the centralized [106, 182] and decentralized [71, 152, 181] settings and more recently the theory was extended to timed systems [176, 39, 8].

4.1 Formal methods for Safety Assessment

Given the increasing complexity of safety-critical systems, recently there has been a growing interest for the application of formal verification techniques in safety analysis [46]. Significant achievements in this area have been reached within the ESACS [75, 43], ISAAC [104, 44] and MISSA[141] projects, three EU projects involving various research centers and industries from the avionics sector.

These projects aimed at improving the shortcomings of the standard development process of complex systems, by managing all the aspects of safety assessment in a unifying framework, by performing the safety assessment analysis in some automated way directly from the system functional model. This approach is based on the notion of fault injection and automatic model extension, that is, automatic integration of nominal (fault-free) system models with (user-defined) fault models, and the use of formal verification tools to analyze the resulting extended model. Similar ideas have been presented in [107, 108, 137, 174, 175].

These techniques have been implemented in the FSAP/NuSMV3-SA platform [86, 45], and recently integrated into the COMPASS platform [**?**, 42, 47]. The COMPASS platform provides a facility for automatic model extension, and it supports several analysis capabilities ranging from functional verification to safety assessment, dependability analysis and performability, fault tolerance evaluation. FSAP supports both the generation of Fault Trees and FMEA tables using symbolic model checking techniques. It can analyze dynamic systems, in particular it supports both *permanent* (steady-state) failure modes (*once failed, always failed*), and *sporadic* or *transient* (intermittent) ones, that is, when faults are allowed to occur sporadically (e.g., a sensor showing an invalid reading for a limited period of time), or when repairing is possible. Moreover, it is possible to generate Dynamic Fault Trees (DFTs) [127] where priority AND gates are used to identify the mutual order of events appearing in a cut set. The COMPASS platform also supports Dynamic Fault Tree Evaluation using probabilistic model checking techniques.

Fault diagnosis is not very well supported by tools: apart from the UMDES tool [155] developed by the University of Michigan, which can handle discrete-event systems, there is no counterpart that can handle timed and hybrid systems. This is not due to a lack of fundamental theory as it was developed a few years ago [176, 39]. It is now also clear that model-checking tools can well be used to check the diagnosability status of a system both in the discrete and timed setting [49]. Moreover, for timed systems, the theory can be extended [8] to deal with the use of discrete (or digital) clocks to diagnose hybrid systems.

4.2 Fault Tree Analysis

A desirable property of fault trees is compositionality, i.e. the possibility to compose fault trees generated for sub-parts of the model, in order to obtain a fault tree for the overall system. Compositionality requires a suitable system decomposition, e.g. functional decomposition or architectural decomposition. As an example, consider a system whose architecture is based on a number of components, and the case where one would like to replace a component with a component of a different type, showing potentially different behaviors with respect to faults (e.g., imagine the latter component being a different candidate for the same system function, or a model at a different abstraction level of the same component). It would be desirable to obtain the overall fault tree for the new system by updating the sub-trees corresponding to the sub-components that have evolved.

The same remark applies for (fault) diagnosers: it should be desirable to obtain diagnosers incrementally or modularly, especially for timed and hybrid systems. Moreover, once a fault is diagnosed, it is also very important to locate the fault and try to detect where it originates from. This problem can be addressed using the partial-order of the history of the observable events in the system, a formal representation of which is called an *unfolding*.

Some investigations have been carried out, concerning compositional frameworks for Fault Tree Analysis, see e.g. [38, 179]. However, the focus of these works is on the definition of a compositional semantics for fault trees and on a proper organization of such fault trees in accordance with the structure of the design model, in presence of evolution - rather than on the on generating fault trees *per se*. To date, the problem of automatic generation of fault trees, starting from a formal specification of both system and fault models, and enjoying compositionality, is an open research problem. A preliminary investigation in this direction had been carried out in [17], where a framework for the mechanical generation of fault trees, based on the architecture of the system model and an extension of refinement called *retrenchment* has been presented. However, the implications of such a mechanical construction in formal verification - in particular at the symbolic level - is still to be explored.

Another important issue related to system evolution is the comparison of safety assessment results for models at different levels of abstraction. In particular, it is normally expected and desirable that a model at an increasing level of detail does not show failure behaviors that are not present in a more abstract version of the same model (otherwise said, the resistance to faults of the more concrete model should be no worse than that of the more abstract model). A suitable notion of refinement for results of safety assessment (in particular, cut sets in fault trees), a methodology and algorithms for checking such refinement relation is presented in [122].

4.3 Failure Modes and Effect Analysis

Failure Modes and Effects Analysis is a “bottom-up” failure analysis method, which examines the causes and consequences of various failures of components within a given system [168]. It may be applied early on in the design process, in order to derive requirements for the management of significant failures. It can also be applied later in the design process in order to confirm that all expected, and significant, failure modes have been addressed. A simple example of FMEA is shown in Table 1, in which the possible causes and effects of the leak from a chemical container are described. As shown, there may be a number of different requirements that are derived, all of which contribute to the mitigation of a possible spill. The depth and strength of these mitigations is dependent on the severity of the effect. For example, if the chemical in question is naturally occuring in the environment then there is a limited effect. If it is poisonous, then more stringent mitigations would be required.

FMEA can be performed by hand, with a team of engineers examining the failures and constructing possible responses. In this case the contents of the tables are often described in informal natural language. However, some FMEAs can be automatically generated from fault models, or partially constructed using existing knowledge of known failures (e.g. physical faults in commonly used parts).

There are a large number of variations on the theme, commonly an extra column which categorises the criticality of the failure is added. In addition, sometimes guidewords are used to suggest failure types. The HAzard and OPerability studies (HAZOP) technique, was developed for use in the chemical industry, and uses guidewords to suggest deviations in the flows of chemicals around a system, such as, “no”, “more”, “less”, or “reverse”. In software analyses, guidewords such as “Omission”, “Commision”, “Early”, “Late”, and “value error” can be helpful [150]

Component	Failure	Cause	Effect	Derived Requirement
Chemical Container	Leak	Chemical Corrosion, Physical tear	Chemical spills, with possible damage to surrounding environment and wildlife	Regular inspection of the containers. Chemical detection sensors. Regular replacement of containers. Sensors detecting current volume of the chemical, which are monitored for any unusual changes in value.

Table 1: Example FMEA

Composition of FMEA tables in multi-component systems can be a complex process. Whilst a failure may occur within one component, its initial cause may be from another component, and the effect of the failure in yet another. Further, the mitigation could be performed by a fourth. This means that the context within which the FMEA is performed must be carefully understood (to understand whether the causes are credible), and it can be difficult to perform in isolation of a system. Formalising the description of the failures can help considerably when considering these chains of interactions.

4.4 Relation with D-MILS requirements

A small fragment of the user identified requirements on the case studies (cf. [63],[64]) concern dependability assement. These requirements express general safety-related constraints for operation as well as recovery strategies in presence of faults, for example:

SMG_SR.3 Rule System shall consider safety constraints specified according the safety constraints of the devices

FVS_SO.3 The MAPL shall be fail secure and shall recover from failure to a trusted state

FVS_SF_PT.1 The MAPL shall preserve a secure state when the following types of failures occur [more]

FVS_SF_PT.5 When automated recovery from service discontinuity is not possible, the MAPL SF shall enter a maintenance mode where the ability to return to a secure state is provided

FVS_SF_PT.7 The MAPL SF shall ensure that start-up have the property that the function either completes successfully, or for the indicated failure scenarios, recovers to a consistent and secure state

The development of compositional validation for dependability and safety-related properties has been identified and is going to be adressed within the verification framework of D-MILS (cf. [65, Sect. 5]):

CV-WP4.11 The verification framework shall provide compositional reasoning for safety-related properties

These requirements will be addressed with the model-based safety assessment described above. In particular, COMPASS will be used to extend the nominal model of the system in order to include faulty behaviors. The extended model will be verified to check if the security properties are valid also in the presence of failures. Model checking of functional properties will be applied to the extended model to check that it is possible to recover from failures. This way, it will be possible to apply off-the-shelf the compositional techniques developed for functional and security properties. Specific contract-based refinement patterns will be applied to the extended model to take into account the monitoring and recovery properties of the specific components. Finally, the OCRA support to contract-based refinement will be extended for compositional fault-tree analysis.

5 Compositional Verification for Security Properties

In general, security properties are expressed using particular predicates on system behaviors. In addition to classical logic, these properties include some kinds of secrecy properties (for instance, the system never broadcasts the key k). They also include correspondence properties (for instance, if the system deletes file f , then the administrator must have requested it). Such predicates on system behaviors are the focus of many successful methods for security analysis. In these last years, several tools have made it possible to prove many such predicates automatically or semi-automatically, even for infinite-state systems (e.g., [36, 125, 145]).

We provide a brief overview of existing methods for verification of two of the most useful classes of security properties, namely for access control policies and for information flow control policies. The two approaches are complementary. On one hand, access control policies and security protocols are essential for ensuring data confidentiality and integrity, but do not provide end-to-end security guarantees. Access control policies do not track information flow through the entire system and do not cope with implicit information flow. Similarly, cryptographic components are used for secure communication and authentication but do not guarantee any global security properties. On the other hand, information flow policies [88, 24, 34] are natural for specifying end-to-end confidentiality and integrity requirements because they put global constraints on the information flow. For example, an access control policy can require that only users with the appropriate read rights can read the file f, while an information flow policy would require that only users with the appropriate security level can get any information about the content of the file f, even indirectly.

5.1 Access Control Policies

Access control is the process of mediating requests to resources and data maintained by a system and determining whether the requests should be granted or denied. The access control decision is usually enforced by mechanisms implementing access control policies. Different access control policies can be combined, corresponding to different criteria for defining what should, and what should not, be allowed, and, in some sense, to different definitions of what ensuring security means. The verification problem amounts to deciding whether vulnerabilities or inconsistencies exist within such policies and/or comparing different policies against each other.

Nowadays, one of the most widely used access control models is role-based access control (RBAC), which is regarded as more general than the traditional models: discretionary access control (DAC), including access control lists (ACLs) and capabilities; and mandatory access control (MAC), including Bel-LaPadula (BLP) and other lattice-based models. RBAC is in turn a specialization of attribute-based access control (ABAC), which is supported by the eXtensible Access Control Markup Language (XACML).

Security features and access control models are also commonly represented as UML extensions e.g., UMLSec[110], secureUML[123], SecureMOVA[20] since UML is widely accepted as a standard for the design of information systems. On these models, access control properties are then usually expressed using OCL (Object Constraint Language) constraints with specific extensions.

A recent overview of existing verification methods usable on formal models of access control policies can be found in [120]. With few exceptions, most of the verification methods and tools are not compositional. There exist tools that operate directly on the UML models annotated with UML constraints [87, 166, 6]. These methods lack generality as they usually focus only on particular properties e.g., Separation of Duty (SoD) constraints. Another family of methods and associated tools rely on translation of the verification problems as a constraint solving or model checking problem. For example, [162, 185, 173, 183] use the Alloy Analyzer[105] as backend to verify or find counter-examples to such policies. A similar approach is followed by [151] which uses the Z framework as a backend.

Moreover, all the previously mentioned methods are in general limited to verification on static model configurations, entirely dissociated from the functional behaviour of the system. Recent approaches are proposed in order to take into account also the functionality and verify access policies while executing some behavior. For example, the work of [19] rely on translation to CSP[100] and classical model checking. Similarly, [120] advocates the use of the B formal method to relate access control with functionality. The functional behaviour is also taken into account in the SecureMOVA tool [20].

Finally, amongst the few attempts that consider compositional approaches to verify security policies one should mention e.g., [80]. This work proposes an extension of contract-based approach for verification of access control policies based on a formal process-algebra model.

5.2 Information Flow Control Policies

Information flow security, and more precisely non-interference, guarantees the confidentiality and integrity of data by assuring that there is no infiltration or exfiltration of data contrary to an information flow (IF) policy model, and that flows of information permitted by the IF policy may be caused only by identifiable and accountable subjects. Non-interference has been initially defined by Goguen and Meseguer [88] as a global extensional property that requires that the system's secret information does not affect its public behavior. More generally, for a system that supports multiple security domains, a domain A is noninterfering wih a domain B if no input or action of A can influence events observable by B or that may be output by B.

Information flow security has been usually considered on two distinct application domains, respectively, for programming language-based models [157, 165, 156] and for trace-based models [132, 133, 184, 129]. While the former mostly focus on verification of security properties related to data flow in programming languages, the latter is usually treating security in event-based distributed systems. Nowadays, ongoing research efforts try to reconcile and unify the two areas. In general, preserving the safety of data flow in the system does not necessarily preserve safe observability on system's public behavior (i.e., secret/private executions may have an observable impact on system public events). This distinction has been recently emphasized in [3], for data leaks and information leaks in business processes based on system's data-flows and work-flows. Previously, the work in [18] pointed out that formal verification of the system's event behavior is not sufficient to guarantee specific data properties. Another attempt to fill the gap between respectively language-based and process calculus-based information security has been proposed in [81]. An explicit distinction has been made between preventing the data leakage through the execution of programs and preventing secret events from being revealed in inter-process communications.

The first technique for verification of non-interference using the so-called unwinding approach was introduced by Goguen and Meseguer in [89]. This approach reduces the verification of information flow security to the existence of a certain unwinding relation. This relation is usually an equivalence relation on system states that respect certain conditions that are shown sufficient to ensure non-interference. Moreover, under particular restrictions, these unwinding conditions can be formulated in terms of individual interactions/events and therefore easier to handle.

An important family of verification techniques for non-interference is based on type theory and type systems. For language-based models, the Jif [76] framework is currently one of the main security-typed language supporting information flow control for sequential applications. Nevertheless, the use of Jif can be painful for large applications as it requires to assign and propagate security labels to variables and expressions through all the program lines. For distributed application, JifSplit [186] splits the application annotated code into multiple threads and assumes that the communications are secure through the network. Although [82] presents a similar approach as JifSplit, it further enforces the network communication security by providing cryptographic mechanisms.

Non-interference can also be verified using traditional model-checking techniques. For trace-based models, recent works on information flow security in web services rely on Petri-nets models generated from BPEL (Business Process Execution Language) orchestration tools [4]. Since BPEL does not allow the modeling of shared resources, the Petri-nets need to be further manually modified by the designer to take into account this information (mainly between users sessions) and security annotations are added at interaction level. Actually, Petri-nets are actually gaining popularity to verify information flows in system's model due to their expressivity for modeling and availability in tool support. Nevertheless, current implemented information flow verification tools based on Petri-nets, such as InDico [5], do not scale well for big Petri-nets models since verification is based on global state space exploration.

Compositional methods for verification of non-interference and security properties in general have been also considered in the litterature [109, 13, 16, 37, 48, 90, 117, 130, 170, 7, 131]. As for functional properties, compositionality allows reduction of the verification complexity and ensures scalability. For example, [7] presents a method that enables compositional verification of non-interference driven by the system topology. It relies on a series of scalable architectural checks (for simpler, star-like or circular topologies). Another interesting compositional method is presented in [131]. This paper considers thread-based concurrent programs and rely on assume-guarantee reasoning to ensure secure access to shared variables.

5.3 Relation with D-MILS requirements

The vast majority of user requirements identified for the two case studies concern security aspects. Many of them concern access control and information flow. In addition, other properties consider confidentiality and integrity data and will require the explicit use of encryption. Amongst the most relevant security properties, that is, mandatory for the two case studies considered (cf. [63],[64]) we find, for example:

SMG_SO.1 Communication / Information Flow between components shall be according to the policy - No other information flow shall occur [more]

SMG_SO.4 The prosumer energy agent shall communicate only required data by the micro grid - The only exception is the data of the main smart meter, since this is required for stability control [more]

SMG_SO.7 Authentication: A user shall authenticate himself to the control software in both cases: prosumer and micro grid. Every prosumer system shall be authenticated to the micro grid

SMG_SO.8 Authorization: Every user shall have a limited set of rights. He shall not be able to obtain more rights than he has been assigned

SMG_SO.9 The admin of each prosumer and smart grid system shall be able to access only his system. Access to other systems shall not be possible

SMG_SO.12 The persistency component (a) shall always be invoked and (b) shall be tamper-proof [more]

SMG_SF_DP.2 The TSF shall enforce the information flow control policy on all subjects and objects represented in the flow graph and all operations that cause that information to flow to and from subjects covered by the SFP

SMG_SF_DP.3 The TSF shall ensure that all operations that cause any information in the TOE to flow to and from any subject in the TOE are covered by an information flow control SFP

FVS_SO.1 The MAPL shall be non-bypassable

FVS_SO.4 The access shall be protected with access control

FVS_SO.8 The MAPL shall guarantee sender/receiver authenticity, no data shall be transferred to unintended receiver, no data shall be received from unintended transmitter

FVS_SF_DP.3 The TSF shall enforce the information flow control SFP on all components of the MILS architecture between the separated subjects and all operations that cause that information to flow to and from subjects covered by the SFP

FVS_SF_DP.4 The TSF shall ensure that all operations that cause any information in the MAPL to flow to and from any subject in the MAPL are covered by an information flow control SFP

FVS_SF_DP.6 The TSF shall permit an information flow between a controlled subject and controlled information via a controlled operation if the following rules hold: for each operation, the IFCA must be identified and authorized

FVS_SF_PT.2 The MAPL SF shall protect SF data from disclosure and modification when it is transmitted between separate parts of the MAPL

In the context of D-MILS, the consortium identified explicitly several requirements for technology in relation to compositional verification of security properties (cf. [65, Sect. 5]), namely:

CV-WP4.14 The verification framework shall provide compositional reasoning for properties related to access control

CV-WP4.15 The verification framework shall provide compositional reasoning for properties related to information flow

CV-WP4.16 The verification framework shall provide compositional reasoning for properties related to encryption

These requirements are currently being adressed by different project partners. The Secure-BIP framework actually developed by **Universite Joseph Fourier** is the extension of the BIP component-based framework [21] with annotations for tracking and reasoning about information flow. This model is used to define and study different types of non-interference and moreover, to develop compositional methods for automated verification. So far, we have investigated two forms of transitive non-interference, repsetively event-based and data-based. For both of them, verification can be reduced to the checking of a number of syntactic conditions defined on the structure of the model, and henceforth, compositional. The model and the associated verification methods are currently being extended for more relaxed types of non-interference (e.g., intransitive and/or using declassification mechanisms). These extensions are mandatory for handling realistic case studies, as the ones considered in the project.

Other approaches based on epistemic temporal logic and diagnosability will be investigated by **Fondazione Bruno Kessler** following the current work on failure detection and identification [41]. In fact, this is similar to information flow properties in that non-observable information is detected and identified based on a sequence of observations.

6 Conclusions

Compositional techniques have been developed in order to achieve scalability for verification of complex, structured systems. Assume-guarantee, contract-based and compositional invariant generation techniques were amongst the first to cope with state explosion occuring in traditional model-based verification by exploiting the system structure. These techniques have been originally applied to functional properties, like safety and liveness on simple models of programs and/or reactive systems. Since then, they have been continuously adapted to cope with richer models of systems (including timing, hybrid or stochastic features) and improved for analysis of more expressive classes of properties (like extra-functional, dependability or security). Last but not least, most of these techniques are currently implemented within automated tools and as such provide an effective support for the design of complex systems nowadays.

This deliverable provides a brief survey of the most relevant compositional techniques for different classes of properties. It appears that compositional verification for functional properties has always been one step beyond others and continously shaped the landscape of research for all other families of properties. Moreover, compositional techniques for functional properties are actually fully supported in state-of-the-art verification and design tools such as AutoFocus[101], NuSMV[52, 50], OCRA [144] or D-Finder[26], to cite only a few developed by D-MILS project partners. Based on the existing results, the D-MILS project aims on further developing compositional techniques for specific properties and/or models as well as to an integrated use, within an unified methodological framework, for all relevant properties in the context of distributed MILS systems.

References

[1] M. Abadi and L. Lamport. Conjoining Specifications. *ACM Trans. Program. Lang. Syst.*, 17(3):507–534, 1995.

[2] E. Ábrahám, B. Becker, F. Klaedtke, and M. Steffen. Optimizing Bounded Model Checking for Linear Hybrid Systems. In *VMCAI*, pages 396–412, 2005.

[3] Rafael Accorsi and Andreas Lehmann. Automatic information flow analysis of business process models. In *Proceedings of the 10th international conference on Business Process Management*, BPM'12, pages 172–187. Springer-Verlag, 2012.

[4] Rafael Accorsi and Claus Wonnemann. Static information flow analysis of workflow models. In *ISSS/BPSC*, pages 194–205, 2010.

[5] Rafael Accorsi, Claus Wonnemann, and Sebastian Dochow. Swat: A security workflow analysis toolkit for reliably secure process-aware information systems. In *ARES*, pages 692–697, 2011.

[6] Gail-Joon Ahn and Hongxin Hu. Towards realizing a formal rbac model in real systems. In Volkmar Lotz and Bhavani M. Thuraisingham, editors, *SACMAT 2007, 12th ACM Symposium on Access Control Models and Technologies, Sophia Antipolis, France, June 20-22, 2007, Proceedings*, pages 215–224. ACM, 2007.

[7] Alessandro Aldini and Marco Bernardo. Component-oriented verification of noninterference. *Journal of Systems Architecture - Embedded Systems Design*, 57(3):282–293, 2011.

[8] K. Altisen, F. Cassez, and S. Tripakis. Monitoring and fault-diagnosis with digital clocks. In *Proceedings of the 6th International Conference on Application of Concurrency to System Design (ACSD'06)*, pages 101–110. IEEE Computer Society, 2006.

[9] R. Alur. Formal verification of hybrid systems. In *EMSOFT*, pages 273–278, 2011.

[10] R. Alur and D.L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[11] R. Alur, T.A. Henzinger, G. Lafferriere, and G.J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, 2000.

[12] Rajeev Alur, Thao Dang, and Franjo Ivancic. Predicate abstraction for reachability analysis of hybrid systems. *ACM Trans. Embedded Comput. Syst.*, 5(1):152–199, 2006.

[13] Suzana Andova, Cas J. F. Cremers, Kristian Gjøsteen, Sjouke Mauw, Stig Fr. Mjølsnes, and Sasa Radomirovic. A framework for compositional verification of security protocols. *Inf. Comput.*, 206(2-4):425–459, 2008.

[14] Eugene Asarin, Thao Dang, and Oded Maler. The d/dt tool for verification of hybrid systems. In *CAV*, pages 365–370, 2002.

[15] G. Audemard, M. Bozzano, A. Cimatti, and R. Sebastiani. Verifying Industrial Hybrid Systems with MathSAT. *ENTCS*, 119(2):17–32, 2005.

[16] Michael Backes, Anupam Datta, Ante Derek, John C. Mitchell, and Mathieu Turuani. Compositional analysis of contract-signing protocols. *Theor. Comput. Sci.*, 367(1-2):33–56, 2006.

[17] R. Banach and M. Bozzano. Retrenchment, and the Generation of Fault Trees for Static, Dynamic and Cyclic Systems. In *SAFECOMP*, pages 127–141, 2006.

[18] Cesare Bartolini, Antonia Bertolino, Eda Marchetti, and Ioannis Parissis. Data flow-based validation of web services compositions: Perspectives and examples. In Rogério Lemos, Felicita Giandomenico, Cristina Gacek, Henry Muccini, and Marlon Vieira, editors, *Architecting Dependable Systems V*, pages 298–325. Springer-Verlag, 2008.

[19] David A. Basin, Samuel J. Burri, and Günter Karjoth. Dynamic enforcement of abstract separation of duty constraints. In Michael Backes and Peng Ning, editors, *Computer Security - ESORICS 2009, 14th European Symposium on Research in Computer Security, Saint-Malo, France, September 21-23, 2009. Proceedings*, volume 5789 of *Lecture Notes in Computer Science*, pages 250–267. Springer, 2009.

[20] David A. Basin, Manuel Clavel, Jürgen Doser, and Marina Egea. Automated analysis of security-design models. *Information & Software Technology*, 51(5), 2009.

[21] A. Basu, M. Bozga, and J. Sifakis. Modeling Heterogeneous Real-time Systems in BIP. In *Software Engineering and Formal Methods SEFM'06 Proceedings*, pages 3–12. IEEE Computer Society Press, 2006.

[22] S.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 *FASE*, pages 43–58, 2012.

[23] Gerd Behrmann, Agnes Cougnard, Alexandre David, Emannuel Fleury, Kim Guldstrand Larsen, and Didier Lime. UPPAAL-Tiga: Time for playing games! In *CAV*, pages 121–125, 2007.

[24] D. E. Bell and L. J. LaPadula. Secure computer system: Unified exposition and multics interpretation. Technical report, ESD-TR-75-306, MITRE Corp. MTR-2997, Bedford, MA, 1975.

[25] J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL - a Tool Suite for Automatic Verification of Real-Time Systems. In *Hybrid Systems*, pages 232–243, 1995.

[26] S. Bensalem, M. Bozga, T-H. Nguyen, and J. Sifakis. D-Finder: A tool for compositional deadlock detection and verification. In *Proceedings of the 21st International Conference on Computer Aided Verification*, pages 614–619, Berlin, Heidelberg, 2009. Springer-Verlag.

[27] S. Bensalem, M. Bozga, T.-H. Nguyen, and J. Sifakis. Compositional verification for component-based systems and application. *IET Software*, 4(3):181–193, 2010.

[28] Saddek Bensalem, Marius Bozga, Axel Legay, Thanh-Hung Nguyen, Joseph Sifakis, and Rongjie Yan. Incremental component-based construction and verification using invariants. In *FMCAD*, pages 257–265, 2010.

[29] Saddek Bensalem, Marius Bozga, Thanh-Hung Nguyen, and Joseph Sifakis. Compositional verification for component-based systems and application. In *Proceedings of the 6th International Symposium on Automated Technology for Verification and Analysis*, pages 64–79, Berlin, Heidelberg, 2008. Springer-Verlag.

[30] A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis. Multiple Viewpoint Contract-Based Specification and Design. In *FMCO*, pages 200–225, 2007.

[31] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. Henzinger, and K.G. Larsen. Contracts for System Design. Technical Report RR-8147, INRIA, November 2012.

[32] A. Benveniste, B. Caillaud, and R. Passerone. A Generic Model of Contracts for Embedded Systems. Technical report, INRIA, 2007.

[33] Dirk Beyer, Claus Lewerentz, and Andreas Noack. Rabbit: A tool for bdd-based verification of real-time systems. In *CAV*, pages 122–125, 2003.

[34] K. J. Biba. Integrity considerations for secure computer systems. Technical report, USAF Electronic Systems Division, Bedford, MA,, 1977.

[35] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In *Proceedings of TACAS'99*, pages 193–207, 1999.

[36] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14 2001), 11-13 June 2001, Cape Breton, Nova Scotia, Canada*, pages 82–96, 2001.

[37] Michele Boreale and Daniele Gorla. On compositional reasoning in the spi-calculus. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, volume 2303 of *Lecture Notes in Computer Science*, pages 67–81. Springer, 2002.

[38] H. Boudali, P. Crouzen, and M.I.A. Stoelinga. A rigorous, compositional, and extensible framework for dynamic fault tree analysis. *IEEE Transactions on Dependable and Secure Computing*, 99(1), 2009.

[39] P. Bouyer, F. Chevalier, and D. D'Souza. Fault diagnosis using timed automata. In V. Sassone, editor, *Proceedings of the 8th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'05)*, volume 3441 of *Lecture Notes in Computer Science*, pages 219–233, Edinburgh, U.K., April 2005. Springer-Verlag.

[40] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A Model-Checking Tool for Real-Time Systems. In *CAV*, pages 546–550, 1998.

[41] M. Bozzano, A. Cimatti, M. Gario, and S. Tonetta. Formal Design of Fault Detection and Identification Components Using Temporal Epistemic Logic. In *TACAS*, 2014. To appear.

[42] M. Bozzano, A. Cimatti, J.-P. Katoen, V.Y. Nguyen, T. Noll, and M. Roveri. The COMPASS Approach: Correctness, Modelling and Performability of Aerospace Systems. In B. Buth, G. Rabe, and T. Seyfarth, editors, *Proc. 28th International Conference on Computer Safety, Reliability and Security (SAFECOM 2010)*, volume 5775 of *lncs*, pages 173–186. Springer, 2009.

[43] M. Bozzano et al. ESACS: An integrated methodology for design and safety analysis of complex systems. *Proc. ESREL 2003*, pages 237–245, 2003.

[44] M. Bozzano et al. ISAAC, a framework for integrated safety analysis of functional, geometrical and human aspects. In *Proc. European Congress on Embedded Real Time Software*, 2006.

[45] M. Bozzano and A. Villafiorita. The FSAP/NuSMV-SA Safety Analysis Platform. *International Journal on Software Tools for Technology Transfer*, 9(1):5–24, 2007.

[46] M. Bozzano and A. Villafiorita. *Design and Safety Assessment of Critical Systems*. CRC Press (Taylor and Francis), an Auerbach Book, 2010.

[47] Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Viet Yen Nguyen, Thomas Noll, and Marco Roveri. Safety, dependability, and performance analysis of extended AADL models. *The Computer Journal*, 54(5):754–775, May 2011. doi: 10.1093/com.

[48] Michele Bugliesi, Riccardo Focardi, and Matteo Maffei. Compositional analysis of authentication protocols. In David A. Schmidt, editor, *Programming Languages and Systems, 13th European Symposium on Programming, ESOP 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings*, volume 2986 of *Lecture Notes in Computer Science*, pages 140–154. Springer, 2004.

[49] F. Cassez. A Note on Fault Diagnosis Algorithms. In *48th IEEE Conference on Decision and Control and 28th Chinese Control Conference*. IEEE Computer Society, 2009.

[50] R. Cavada, A. Cimatti, E. Olivetti C.A. Jochim, G. Keighren, M. Pistore, M. Roveri, and A. Tchaltsev. *NuSMV 2.1 User Manual*, 2002.

[51] Alongkrit Chutinan and Bruce H. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In *HSCC*, pages 76–90, 1999.

[52] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In *CAV*, pages 359–364, 2002.

[53] A. Cimatti, M. Dorigatti, and S. Tonetta. OCRA: A tool for checking the refinement of temporal contracts. In *ASE*, pages 702–705, 2013.

[54] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta. IC3 Modulo Theories via Implicit Predicate Abstraction. In *TACAS*, 2014. To appear.

[55] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. A quantifier-free smt encoding of non-linear hybrid automata. In *FMCAD*, pages 187–195, 2012.

[56] Alessandro Cimatti and Stefano Tonetta. A Property-Based Proof System for Contract-Based Design. In *EUROMICRO-SEAA*, pages 21–28, 2012.

[57] K. Claessen and N. Sörensson. A liveness checking algorithm that counts. In *FMCAD*, pages 52–59, 2012.

[58] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[59] Edmund M. Clarke, Ansgar Fehnker, Zhi Han, Bruce H. Krogh, Joël Ouaknine, Olaf Stursberg, and Michael Theobald. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *Int. J. Found. Comput. Sci.*, 14(4):583–604, 2003.

[60] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, 1994.

[61] D.D. Cofer, A. Gacek, S.P. Miller, M.W. Whalen, B. LaValley, and L. Sha. Compositional Verification of Architectural Models. In *NFM*, pages 126–140, 2012.

[62] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Robert M. Graham, Michael A. Harrison, and Ravi Sethi, editors, *Fourth ACM Symposium on Principles of Programming Languages, POPL, Los Angeles, California, USA*, pages 238–252. ACM, 1977.

[63] Safety and security requirements for fortiss Smart Microgrid demonstrator. Technical Report D1.1, Version 1.1, D-MILS Project, January 2013.

[64] Safety and security requirements for Frequentis Voice Service demonstrator. Technical Report D1.2, Version 1.1, D-MILS Project, January 2013.

[65] Requirements for distributed MILS technology (draft). Technical Report D1.3, Version 0.1, D-MILS Project, April 2013.

[66] W. Damm, H. Hungar, B. Josko, T. Peikenkamp, and I. Stierand. Using contract-based component specifications for virtual integration testing and architecture design. In *DATE*, pages 1023–1028, 2011.

[67] Werner Damm and David Harel. Lscs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19:45–80, 2001.

[68] Thao Dang, Oded Maler, and Romain Testylier. Accurate hybridization of nonlinear systems. In *HSCC*, pages 11–20, 2010.

[69] Luca de Alfaro, Thomas A. Henzinger, and Ranjit Jhala. Compositional methods for probabilistic systems. In *Proc. the 12th International Conference on Concurrency Theory, Springer Lect. Notes Comp. Sci*, pages 351–365, 2001.

[70] W.P. de Roever, F.S. de Boer, U. Hannemann, J. Hooman, Y. Lakhnech, M. Poel, and J. Zwiers. *Concurrency Verification: Introduction to Compositional and Noncompositional Methods*, volume 54 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2001.

[71] R. Debouk, S. Lafortune, and D. Teneketzis. Coordinated decentralized protocols for failure diagnosis of discrete event systems. *Discrete Event Dynamic Systems*, 10(1-2):33–86, 2000.

[72] Benoît Delahaye, Benoît Caillaud, and Axel Legay. Probabilistic contracts: A compositional reasoning methodology for the design of stochastic systems. In Luís Gomes, Victor Khomenko, and João M. Fernandes, editors, *ACSD*, pages 223–232. IEEE Computer Society, 2010.

[73] William Denman, Behzad Akbarpour, Sofiène Tahar, Mohamed H. Zaki, and Lawrence C. Paulson. Formal verification of analog designs using metitarski. In *FMCAD*, pages 93–100, 2009.

[74] C. Eisner and D. Fisman. *A Practical Introduction to PSL.* Springer-Verlag, 2006.

[75] The ESACS project. `http://www.esacs.org`.

[76] Andrew Myers et al. The JIF framework. `http://www.cs.cornell.edu/jif/`, 2013.

[77] Kousha Etessami, Marta Kwiatkowska, Moshe Vardi, and Mihalis Yannakakis. Multi-objective model checking of Markov decision processes. In O. Grumberg and M. Huth, editors, *Proc. 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07)*, volume 4424 of *LNCS*, pages 50–65. Springer, 2007.

[78] Ansgar Fehnker, Edmund M. Clarke, Sumit Kumar Jha, and Bruce H. Krogh. Refining abstractions of hybrid systems using counterexample fragments. In *HSCC*, pages 242–257, 2005.

[79] L. Feng, M. Kwiatkowska, and D. Parker. Automated learning of probabilistic assumptions for compositional reasoning. In D. Giannakopoulou and F. Orejas, editors, *Proc. 14th International Conference on Fundamental Approaches to Software Engineering (FASE'11)*, volume 6603 of *LNCS*, pages 2–17. Springer, 2011.

[80] Hakim Ferrier-Belhaouari, Pierre Konopacki, Régine Laleau, and Marc Frappier. A design by contract approach to verify access control policies. In Isabelle Perseil, Karin Breitman, and Marc Pouzet, editors, *17th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2012, Paris, France, July 18-20, 2012*, pages 263–272. IEEE Computer Society, 2012.

[81] Riccardo Focardi, Sabina Rossi, and Andrei Sabelfeld. Bridging language-based and process calculi security. In *In Proc. of Foundations of Software Science and Computation Structures (FOSSACS'05)*, volume 3441 of *LNCS*, pages 299–315. Springer-Verlag, 2005.

[82] Cédric Fournet, Gurvan Le Guernic, and Tamara Rezk. A security-preserving compiler for distributed programs: from information-flow policies to cryptographic mechanisms. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 432–441. ACM, 2009.

[83] Goran Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations*. PhD thesis, Radboud Universiteit Nijmegen, 2005.

[84] Goran Frehse. Phaver: algorithmic verification of hybrid systems past hytech. *STTT*, 10(3):263–279, 2008.

[85] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. Spaceex: Scalable verification of hybrid systems. In *CAV*, pages 379–395, 2011.

[86] The FSAP/NuSMV-SA platform. `https://es.fbk.eu/tools/FSAP`.

[87] Martin Gogolla, Fabian Büttner, and Mark Richters. Use: A uml-based specification environment for validating uml and ocl. *Sci. Comput. Program.*, 69(1-3):27–34, 2007.

[88] Joseph A. Goguen and José Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pages 11–20, 1982.

[89] Joseph A. Goguen and José Meseguer. Unwinding and inference control. In *IEEE Symposium on Security and Privacy*, pages 75–86, 1984.

[90] Roberto Gorrieri, Fabio Martinelli, Marinella Petrocchi, and Anna Vaccarelli. Compositional verification of integrity for digital stream signature protocols. In *3rd International Conference on Application of Concurrency to System Design (ACSD 2003), 18-20 June 2003, Guimaraes, Portugal*, pages 142–149. IEEE Computer Society, 2003.

[91] S. Graf, R. Passerone, and S. Quinton. Contract-Based Reasoning for Component Systems with Complex Interactions . In *TIMOBD'11*, 2011.

[92] Susanne Graf and Hassen Saïdi. Construction of abstract state graphs with pvs. In *CAV*, pages 72–83, 1997.

[93] Colas Le Guernic and Antoine Girard. Reachability analysis of hybrid systems using support functions. In *CAV*, pages 540–554, 2009.

[94] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:102–111, 1994.

[95] T.A. Henzinger. The theory of hybrid automata. In *IEEE Symp. on Logic in Computer Science (LICS)*, pages 278–292. IEEE CS Press, 1996.

[96] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: A Model Checker for Hybrid Systems. In *CAV*, pages 460–463, 1997.

[97] Holger Hermanns. *Interactive Markov Chains: The Quest for Quantified Quality*, volume 2428 of *Lecture Notes in Computer Science*. Springer, 2002.

[98] Holger Hermanns and Lijun Zhang. From concurrency models to numbers - performance and dependability. In *Software and Systems Safety - Specification and Verification*, volume 30 of *NATO Science for Peace and Security Series - D: Information and Communication Security*, pages 182–210. IOS Press, 2011.

[99] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.

[100] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall International, 1984.

[101] Florian Hölzl and Martin Feilkas. Autofocus 3 - a scientific tool prototype for model-based development of component-based, reactive, distributed systems. In *Model-Based Engineering of Embedded Real-Time Systems*, volume 6100 of *Lecture Notes in Computer Science*, pages 317–322. Springer, 2007.

[102] G.J. Holzmann. *Design and validation of computer protocols*. Prentice-Hall, Inc., 1991.

[103] Michael Huth and Marta Z. Kwiatkowska. Quantitative analysis and model checking. In *LICS*, pages 111–122. IEEE Computer Society, 1997.

[104] The ISAAC project. `http://www.isaac-fp6.org`.

[105] Daniel Jackson. Alloy: a lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.*, 11(2):256–290, 2002.

[106] S. Jiang, Z. Huang, V. Chandra, and R. Kumar. A polynomial algorithm for testing diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, 46(8), August 2001.

[107] A. Joshi and M. P. E. Heimdahl. Model-Based Safety Analysis of Simulink Models Using SCADE Design Verifier. In R. Winther, B.A. Gran, and G. Dahll, editors, *Proc. Conference on Computer Safety, Reliability and Security (SAFECOMP 2005)*, volume 3688 of *Lecture Notes in Computer Science*, pages 122–135. Springer, 2005.

[108] A. Joshi, S. P. Miller, M. Whalen, and M. P. E. Heimdahl. A Proposal for Model-Based Safety Analysis. In *Proc. AIAA / IEEE Digital Avionics Systems Conference (DASC 2005)*, 2005.

[109] Jan Jürjens. Secure information flow for concurrent processes. In Catuscia Palamidessi, editor, *CONCUR 2000 - Concurrency Theory, 11th International Conference, University Park, PA, USA, August 22-25, 2000, Proceedings*, volume 1877 of *Lecture Notes in Computer Science*, pages 395–409. Springer, 2000.

[110] Jan Jürjens. *Secure systems development with UML*. Springer, 2005.

[111] Roland Kindermann, Tommi A. Junttila, and Ilkka Niemelä. Modeling for symbolic analysis of safety instrumented systems with clocks. In *ACSD*, pages 185–194, 2011.

[112] Roland Kindermann, Tommi A. Junttila, and Ilkka Niemelä. Smt-based induction methods for timed systems. In *FORMATS*, pages 171–187, 2012.

[113] Anvesh Komuravelli, Corina S. Pasareanu, and Edmund M. Clarke. Assume-guarantee abstraction refinement for probabilistic systems. *CoRR*, abs/1207.5086, 2012.

[114] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Assume-guarantee verification for probabilistic systems. In J. Esparza and R. Majumdar, editors, *Proc. 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'10)*, volume 6105 of *LNCS*, pages 23–37. Springer, 2010.

[115] M. Kwiatkowska, G. Norman, and R. Segala. Automated verification of a randomized distributed consensus protocol using Cadence SMV and PRISM. In G. Berry, H. Comon, and A. Finkel, editors, *Proc. 13th International Conference on Computer Aided Verification (CAV'01)*, volume 2102 of *LNCS*, pages 194–206. Springer, 2001.

[116] Marta Kwiatkowska, Gethin Norman, David Parker, and Jeremy Sproston. Performance analysis of probabilistic timed automata using digital clocks, 2003.

[117] Stéphane Lafrance. Using equivalence-checking to verify robustness to denial of service. *Computer Networks*, 50(9):1327–1348, 2006.

[118] J. C. Laprie. *Dependability: Basic Concepts and Terminology*. Springer, 1991.

[119] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *STTT*, 1(1-2):134–152, 1997.

[120] Y. Ledru, A. Idani, J. Milhau, N. Qamar, R. Laleau, J-L. Richier, and M-A. Labiadh. Taking into account functional models in the validation of IS security policies. In *Proceedings of CAISE 2011 Workshop*, volume 83 of *LNBIP*, pages 592–606, 2011.

[121] Nancy G. Levenson. Completeness in formal specification language design for process-control systems. In *FMSP*, pages 75–87, 2000.

[122] O. Lisagor, M.Bozzano, M. Bretschneider, and T.P. Kelly. Incremental Safety Assessment: Enabling the Comparison of Safety Analysis Results. In *Proceedings of ISSC 2010*. System Safety Society, 2010.

[123] Torsten Lodderstedt, David A. Basin, and Jürgen Doser. Secureuml: A uml-based modeling language for model-driven security. In Jean-Marc Jézéquel, Heinrich Hußmann, and Stephen Cook, editors, *UML 2002 - The Unified Modeling Language, 5th International Conference, Dresden, Germany, September 30 - October 4, 2002, Proceedings*, volume 2460 of *Lecture Notes in Computer Science*, pages 426–441. Springer, 2002.

[124] Claire Loiseaux, Susanne Graf, Joseph Sifakis, Ahmed Bouajjani, and Saddek Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6(1):11–44, 1995.

[125] Gavin Lowe. Breaking and fixing the needham-schroeder public-key protocol using fdr. In *Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS '96, Passau, Germany, March 27-29, 1996, Proceedings*, pages 147–166, 1996.

[126] Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. Hybrid I/O Automata. *Inf. Comput.*, 185(1):105–157, 2003.

[127] R. Manian, J. B. Dugan, D. Coppit, and K. J. Sullivan. Combining Various Solution Techniques for Dynamic Fault Tree Analysis of Computer Systems. In *Proc. High-Assurance Systems Engineering Symposium (HASE 1998)*, pages 21–28. IEEE, 1998.

[128] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems*. Springer-Verlag New York, Inc., 1992.

[129] Heiko Mantel. Possibilistic definitions of security - an assembly kit. In *Proceedings of the 13th IEEE workshop on Computer Security Foundations*, CSFW '00, pages 185–. IEEE Computer Society, 2000.

[130] Heiko Mantel. On the composition of secure systems. In *2002 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 12-15, 2002*, pages 88–101. IEEE Computer Society, 2002.

[131] Heiko Mantel, David Sands, and Henning Sudbrock. Assumptions and guarantees for compositional noninterference. In *Proceedings of the 24th IEEE Computer Security Foundations Symposium, CSF 2011, Cernay-la-Ville, France, 27-29 June, 2011*, pages 218–232. IEEE Computer Society, 2011.

[132] Daryl McCullough. Noninterference and the composability of security properties. In *Proceedings of the 1988 IEEE conference on Security and privacy*, SP'88, pages 177–186. IEEE Computer Society, 1988.

[133] John McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proceedings of the 1994 IEEE Symposium on Security and Privacy*, SP '94, pages 79–. IEEE Computer Society, 1994.

[134] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[135] K.L. McMillan. Circular Compositional Reasoning about Liveness. In *CHARME*, pages 342–345, 1999.

[136] B. Meyer. Applying "Design by Contract". *Computer*, 25(10):40–51, 1992.

[137] S. P. Miller, A. C. Tribble, and M. P. E. Heimdahl. Proving the Shalls. In *Proc. Formal Methods Europe (FM 2003)*, volume 2805 of *Lecture Notes in Computer Science*, pages 75–93. Springer, 2003.

[138] R. Milner. *A Calculus of Communication Systems*, volume 92 of *LNCS*. Springer, 1980.

[139] Robin Milner. A complete axiomatisation for observational congruence of finite-state behaviors. *Information and Computation*, 81(2):227–247, 1989.

[140] Matteo Mio and Alex Simpson. A proof system for compositional verification of probabilistic concurrent processes. In *Proceedings of the 16th international conference on Foundations of Software Science and Computation Structures*, FOSSACS'13, pages 161–176, Berlin, Heidelberg, 2013. Springer-Verlag.

[141] The MISSA project. `http://www.missa-fp7.eu`.

[142] Ian M. Mitchell and Yoshihiko Susuki. Level set methods for computing reachable sets of hybrid systems with differential algebraic equation dynamics. In *HSCC*, pages 630–633, 2008.

[143] Georges Morbé, Florian Pigorsch, and Christoph Scholl. Fully symbolic model checking for timed automata. In *CAV*, pages 616–632, 2011.

[144] The Othello Contract Refinement Analysis (OCRA) tool. `https://es.fbk.eu/tools/ocra`.

[145] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1-2):85–128, 1998.

[146] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reasoning*, 41(2):143–189, 2008.

[147] André Platzer and Jan-David Quesel. Keymaera: A hybrid theorem prover for hybrid systems (system description). In *IJCAR*, pages 171–178, 2008.

[148] A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.

[149] Stephen Prajna and Ali Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *HSCC*, pages 477–492, 2004.

[150] David Pumgrey. *The Principled Design of Computer System Safety Analyses*. PhD thesis, University of York, U.K., 1999.

[151] Nafees Qamar, Yves Ledru, and Akram Idani. Validation of security-design models using z. In Shengchao Qin and Zongyan Qiu, editors, *Formal Methods and Software Engineering - 13th International Conference on Formal Engineering Methods, ICFEM 2011, Durham, UK, October 26-28, 2011. Proceedings*, volume 6991 of *Lecture Notes in Computer Science*, pages 259–274. Springer, 2011.

[152] W. Qiu and R. Kumar. Decentralized failure diagnosis of discrete event systems. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 36(2):384–395, 2006.

[153] S. Quinton and S. Graf. Contract-Based Verification of Hierarchical Systems of Components. In *SEFM*, pages 377–381, 2008.

[154] Stefan Ratschan and Zhikun She. Safety verification of hybrid systems by constraint propagation-based abstraction refinement. *ACM Trans. Embedded Comput. Syst.*, 6(1), 2007.

[155] L. Ricker, S. Lafortune, and S. Genc. DESUMA: A Tool Integrating GIDDES and UMDES. In *Proc. of the* 8^{th} *Workshop on Discrete Event Systems (WODES'08)*, Ann Arbor, MI, USA, 2006. IEEE.

[156] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on selected areas in communications*, 21(1), 2003.

[157] Andrei Sabelfeld and David Sands. A per model of secure information flow in sequential programs. *Higher Order Symbol. Comput.*, 14(1):59–91, March 2001.

[158] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. C. Teneketzis. Diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, 40(9), September 1995.

[159] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. C. Teneketzis. Failure diagnosis using discrete-event models. *IEEE Transactions on Control Systems technology*, 4(2), March 1996.

[160] A.L. Sangiovanni-Vincentelli, W. Damm, and R. Passerone. Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems. *Eur. J. Control*, 18(3):217–238, 2012.

[161] S. Sankaranarayanan and A. Tiwari. Relational abstractions for continuous and hybrid systems. In *CAV*, pages 686–702, 2011.

[162] A. Schaad and J.D. Moffett. A lightweight approach to specification and analysis of role-based access control extensions. In *Proc. of 7th SACMAT*. ACM Press, New York, 2002.

[163] Roberto Segala. *Modeling and verification of randomized distributed real-time systems*. PhD thesis, Massachusetts Institute of Technology, 1995.

[164] M. Sheeran, S. Singh, and G. Stålmarck. Checking Safety Properties Using Induction and a SAT-Solver. In *FMCAD*, pages 108–125, 2000.

[165] Geoffrey Smith and Dennis Volpano. Secure information flow in a multi-threaded imperative language. In *Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '98, pages 355–364. ACM, 1998.

[166] Karsten Sohr, Michael Drouineaud, Gail-Joon Ahn, and Martin Gogolla. Analyzing and managing role-based access control policies. *IEEE Trans. Knowl. Data Eng.*, 20(7):924–939, 2008.

[167] Maria Sorea. Bounded model checking for timed automata. In *Electronic Notes in Theoretical Computer Science*, page 2002. Elsevier, 2002.

[168] Neil Storey. *Safety-Critical Computer Systems*. Addison Wesley Longman, 1996.

[169] Thomas Sturm and Ashish Tiwari. Verification and synthesis using real quantifier elimination. In *ISSAC*, pages 329–336, 2011.

[170] Simone Tini. Rule formats for compositional non-interference properties. *J. Log. Algebr. Program.*, 60-61:353–400, 2004.

[171] A. Tiwari. Abstractions for hybrid systems. *Formal Methods in System Design*, 32(1):57–83, 2008.

[172] Ashish Tiwari and Gaurav Khanna. Nonlinear systems: Approximating reach sets. In *HSCC*, pages 600–614, 2004.

[173] Manachai Toahchoodee, Indrakshi Ray, Kyriakos Anastasakis, Geri Georg, and Behzad Bordbar. Ensuring spatio-temporal access control for real-world applications. In Barbara Carminati and James Joshi, editors, *SACMAT 2009, 14th ACM Symposium on Access Control Models and Technologies, Stresa, Italy, June 3-5, 2009, Proceedings*, pages 13–22. ACM, 2009.

[174] A. C. Tribble, D. L. Lempia, and S. P. Miller. Software Safety Analysis of a Flight Guidance System. In *Proc. AIAA / IEEE Digital Avionics Systems Conference (DASC 2002)*, 2002.

[175] A. C. Tribble and S. P. Miller. Software Safety Analysis of a Flight Management System Vertical Navigation Function – A Status Report. In *Proc. AIAA / IEEE Digital Avionics Systems Conference (DASC 2003)*, 2003.

[176] S. Tripakis. Fault diagnosis for timed automata. In W. Damm and E.-R. Olderog, editors, *Proceedings of the International Conference on Formal Techniques in Real Time and Fault Tolerant Systems (FTRTFT'02)*, volume 2469 of *Lecture Notes in Computer Science*, pages 205–224. Springer-Verlag, 2002.

[177] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl. Fault tree handbook. Technical Report NUREG-0492, Systems and Reliability Research Office of Nuclear Regulatory Research U.S. Nuclear Regulatory Commission, 1981.

[178] W. E. Vesely, M. Stamatelatos, J. Dugan, J. Fragola, J. Minarick III, and J. Railsback. Fault Tree Handbook with Aerospace Applications. Technical report, NASA, 2002.

[179] M. Walker, L. Bottaci, and Y. Papadopoulos. Compositional Temporal Fault Tree Analysis. In Saglietti and Oster, editors, *Computer Safety, Reliability, and Security - SAFECOMP 2007*, volume 4680 of *lncs*, pages 105–119. Springer, 2007.

[180] Farn Wang. Efficient model-checking of timed automata with clock-restriction diagram. In *APLAS*, pages 207–224, 2001.

[181] Y. Wang, T.-S. Yoo, and S. Lafortune. Diagnosis of discrete event systems using decentralized architectures. *Discrete Event Dynamic Systems*, 17(2):233–263, 2007.

[182] T. S. Yoo and S. Lafortune. Polynomial-time verification of diagnosability of partially-observed discrete-event systems. *IEEE Transactions on Automatic Control*, 47(9):1491–1495, September 2002.

[183] L. Yu, R. France, I. Ray, and S Ghosh. A rigorous approach to uncovering se- curity policy violations in uml designs. In *Int. Conf. on Engineering Complex Computer Systems*. IEEE, Los Alamitos, 2009.

[184] A. Zakinthinos and E. S. Lee. A general theory of security properties. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, SP '97, pages 94–. IEEE Computer Society, 1997.

[185] J. Zao, H. Wee, J. Chu, and D Jackson. Rbac schema verification using lightweight formal model and constraint analysis. In *Proceedings of 8th ACM Symposium on Access Control Models and Technologies*, 2003.

[186] Steve Zdancewic, Lantian Zheng, Nathaniel Nystrom, and Andrew C. Myers. Secure program partitioning. *ACM Trans. Comput. Syst.*, 20(3):283–328, August 2002.