Model-Based Design of Dependable Systems: Limitations and Evolution of Analysis and Verification Approaches

Jose Ignacio Aizpurua and Eñaut Muxika
Department of Signal Theory and Communications
University of Mondragon
Spain
{*jiaizpurua,emuxika*}*@mondragon.edu*

***Abstract*—Designing a dependable system successfully is a challenging issue that is an ongoing research subject in the literature. Different approaches have been adopted to analyse and verify the dependability of a system design. This process is far from obvious and often hampered due to the limitations of the classical dependability analysis and verification approaches. This paper provides an overview of model-based dependability analysis, design and verification approaches. Firstly, model-based analysis approaches are grouped by the limitations of the classical approaches. Secondly, design approaches have been classified looking at their underlying recovery strategies: hardware replication and hardware reuse. Then, the ins and outs of model-based verification approaches are identified starting from fault injection approaches towards their evolution into model-based integrative approaches. Finally, a model-based hybrid design process is presented making use of the reviewed analysis, design and verification approaches.**

Keywords-Model-based dependability analysis, System design, Heterogeneous redundancy, Dependability verification.

Acronyms and Abbreviations

AADL	Architecture Analysis and Design Language
BDMP	Boolean logic Driven Markov Process
CFP	Compositional Failure Propagation
CFT	Component Fault Tree
(D)FTA	(Dynamic) Fault Tree Analysis
(D)RBD	(Dynamic) Reliability Block Diagram
(D(S))PN	(Deterministic and (Stochastic)) Petri Nets
(D)(C)TMC	(Discrete)(Continuous) Time Markov Chain
DOE	Design of Experiments
FI	Fault Injection
(FM)E(C)A	(Failure Mode) and Effect (Criticality) Analysis
FPT(C)(N)	Failure Propagation and Transformation (Calculus)(Notation)
HiP-HOPS	Hierarchically Performed Hazard Origin and Propagation Studies
IMA	Integrated Modular Avionics
MC	Model Checking
MCS	Monte Carlo Simulations
SEFT	State-Event Fault Trees

I. Introduction

Designing a dependable system presents many challenges throughout the development phase - from system specification to system validation and verification. This process is further complicated owing to the increasing complexity of the current systems, which use many and different components. Model-based design approaches provide mechanisms to manage this complexity effectively. However, these approaches together with dependability analysis and verification approaches, add some limitations to the system design process. Hence, those who are not familiar with the field of model-based design of dependable systems will find it useful to have a list of sources characterized by their limitations and evolutions. Our goal is not to exhaustively evaluate the specific features of these approaches, neither to give means to study these techniques in great detail. Instead, we are aimed at aggregating a comprehensive list of works grouped by their main characteristics.

We have completed our previous work [1] providing the reader a guided collection of sources to indicate where to learn about the model-based design of dependable systems. Namely, we have extended previous concepts and characterized three additional aspects: (1) tool-support of the analysis and verification approaches, (2) classification of dependable design approaches with respect to their recovery strategies, and (3) evolution of model-based fault injection approaches.

In computing systems, dependability is defined as "ability of a system to deliver a service that can be justifiably trusted" [2]. Such trustworthiness is based on the assurance of dependability requirements. These requirements are defined through dependability attributes: *Reliability*, *Availability*, *Maintainability*, *Safety (RAMS)*, *confidentiality* and *integrity*. The scope of this overview focuses on RAMS attributes. Consequently, security aspects (confidentiality and integrity) are not addressed.

Reliability is the ability of an item to perform a required function under given conditions for a stated period of time. It is usually characterized through Mean Time To Failure (MTTF) [3]. *Maintainability* is the ability to undergo repairs and modifications to restore to a state in

which the system can perform its required actions and it is commonly characterized by Mean Time To Restore (MTTR). *Availability* is the readiness for correct service defined by the ratio between MTTF and MTTF plus MTTR. *Safety* is the absence of catastrophic consequences on the user(s) and the environment. These four attributes are closely interrelated in such a way that if an attribute fails to meet the requirements, systems dependability is seriously threatened.

This survey concentrates on three main phases: dependability analysis, system design and verification. Despite being aware of the relevance of software code for system dependability in each of these phases, we will consider software code as a black box component to limit the extension of this paper (interested readers can refer to [4] [5] as an example).

Dependability analysis techniques can be organised by looking at how different system failures are characterized with its corresponding underlying formalisms. On one hand, *event-based* approaches reflect the system functionality and failure behaviour through combination of events. This analysis results in either Fault Tree (FT) like [6] or Failure Mode and Effect Analysis (FMEA) like [7] structures, which emphasizes the reliability and safety attributes. On the other hand, *state-based* approaches map the analysis models into a state-based formalisms (e.g., Stochastic Petri Nets (SPN) [8]). Those approaches analyse system changes with respect to time and concentrate on reliability and availability attributes.

Concerning the design of dependable systems, there exist alternative recovery strategies that add redundancies to the system design in order to avoid single points of failure and thus, provide fault tolerance. Traditionally, hardware replication have been viewed as a feasible solution to recover from failures. However, there is also another design strategy which compensates for component failures through the reuse of hardware components. We will address these design strategies and their influence on dependability and cost.

Fault Injection (FI) and Model-Checking (MC) [9] approaches have been widely adopted for the verification of design decisions. Since these approaches enable to learn about the system behaviour in the presence of faults and verify its correctness relying on a fully automated model-based approach, we concentrate on model-based FI approaches. They evaluate the dependability requirements using functional and failure behaviour models. Moreover, we also cover the recent evolution of these approaches towards the integration multiple approaches using a single reference model, i.e., model-based integrative verification approaches. We are conscious that there are other verification approaches, e.g., correct by construction paradigm aimed at ensuring the validity of the system by design [10] (i.e., formal verification). Nonetheless, we have decided to limit the overview to model-based analysis and verification approaches due to their potential to integrate in the system design process seamlessly.

The remainder of this paper is organized as follows: Section II classifies dependability analysis techniques based on the limitations of classical dependability analysis techniques. Section III groups dependable design approaches characterized by the replication and reuse of hardware components. Section IV discusses the characteristics of the verification approaches when designing a dependable system. Section V outlines a model-based hybrid design process based on the reviewed analysis, design and verification approaches. Finally, Section VI draws conclusions remarking different challenges for the model-based design of dependable systems.

II. Review and Classification of Model-Based Dependability Analysis Techniques

Event-based approaches analyse the failure behaviour of the system by investigating the logic succession of faults. They identify an event sequence leading to equipment or function failure. Differences are mainly based on representation and analysis structures. Two of the most widely used techniques are: FT Analysis (FTA) [6] and FMEA [7].

- *FTA* is a top-down deductive analysis technique aimed at finding all the ways in which a failure can occur. Starting from an undesirable system-level failure (i.e., top-event), its immediate causes to occur are identified until reaching the lowest component-level (i.e., basic-event). The top-event is broken down into intermediate and basic-events linked with logic gates organised in a tree-like structure. The resulting FT, is a qualitative model in the form of combinations of events which are necessary to the top-event to occur. This model can be quantified by ascribing probabilities to the basic events and combining them to evaluate the probability of the top-event. Moreover, importance measurements can be carried out to quantify the contribution of the basic-event occurrences to the top-event failure. There exist different importance measurement methods, e.g., Fussell-Vessely, Birnbaum [6].
- *FMEA* is a bottom-up inductive analysis technique. Starting from the different ways that a system can fail (i.e., known Failure Modes (FM)), it evaluates the effects that these failures can have on a process. Its main objective is an early identification of critical failure possibilities within the system design. Results are organized in a table identifying at least component's failure modes, effects, safeguards and actions. FMEA is characterised for being a qualitative technique for design analysis, while quantification of failure modes is carried out by Failure Mode and Effect Criticality Analysis (FMECA). The criticality analysis ranks critical failure mode effects by taking into account their occurrence probability.

Both techniques focus on the identification of events that jeopardize the system design objectives. However, their initial assumptions as well as their logical orientation are different: while FTA moves from known effects towards unknown causes in a deductive manner, FMEA progresses from known causes towards unknown effects inductively. They are not orthogonal techniques, indeed they are complementary and in some cases they overlap [11]. The extended usage of these approaches for dependability related tasks have lead to the identification of the main limitations. Subsequently, there has been a long list of works aimed at overcoming them:

- L1: FMEA and FTA are static representations of the system, neither time information nor sequence dependencies are taken into account [12].
- L2: Orientation of FTA and FMEA concentrate on the analysis of failure chain information. Consequently, their hierarchy reflects failure influences without considering system functional architecture (design) information [13].
- L3: FMEA and FTA depend on the analyst's skill to reflect the aspects of interest. Failure modes and undesired events must be foreseen, resulting in a process highly dependent on analyst's knowledge of the system [14].
- L4: Manageability and legibility of FTA and FMEA models is hampered when analysing complex systems. Model size, lack of resources to handle interrelated failures and repeated events, in conjunction with few reusability means, are its main impediments [13] [15].

L1 refers to the capability of the technique to handle temporal notions. This is of paramount importance when analysing fault tolerant systems. L2 emphasizes the interdisciplinary work between dependability analysis and architectural design. Joining both procedures helps obtaining a design, which meets dependability requirements consistently. L3 entails a trade-off solution between the time consuming analysis resulted from understanding the failure behaviour of the system and the acquired experience. A substantial body of works have been oriented towards the automatic generation of analysis models from design models (refer to groups 3, 5 in Table IV) addressing limitations L2 and L3. These approaches promote the reuse of design models showing the effects of design changes in the analysis results. However, note that the correctness of the analysis lies in the accuracy of the failure annotations. Finally, L4 underlines the capability of the model to handle the component-wise nature of embedded systems. This permits obtaining a model that better adheres to the real problem and avoids confusing results.

Many authors have developed new alternatives or extended existing ones. Three groups are identified in order to gather the approaches and limitations strategically. Firstly, L1 is addressed in the Subsection *Dynamic Solutions for Static-Logic Approaches*. Secondly, L2 and L4 are covered in the Subsection *Compositional Failure Propagation Analysis Approaches*. Finally, specifically focusing on L3 and generally addressing the remainder of limitations *Model-Based Transformational Approaches* are studied. Note that some approaches cannot be limited to a specific group, hence they are classified accordingly to its main contribution.

A. Dynamic Solutions for Static-Logic Approaches

The limitation concerning the lack of time information has been addressed by several authors to deal with system *dynamics* such as redundancy or repair strategies.

Dugan et al. [12] paved the way to cope with configuration changing analysis using FTs by defining Dynamic Fault Tree (DFT) methodology. New gates were introduced accounting for event ordered situations like common cause failures and redundancy strategies. In [16], temporal notions and FT models were integrated in order to handle the probabilistic timed behaviour of the system. The model reflects how modular FT models are switched through discrete points in time taking into account time dependant basic events.

Other alternatives to analyse DFT models are based on Monte Carlo Simulations (MCS) by specifying temporal failure and repair behaviours of components through state-time diagrams [17]. In [18], MatCarloRe approach is presented based on Simulink [19] for DFT modelling and reliability analysis. The technique integrates MCS and FTA methodologies resulting in a intuitive model-based simulating environment.

Following the way of DFTs, an approach emerged based on Reliability Block Diagrams (RBD) [3]. RBD is focused on the analysis of the success of the system, instead of the failure analysis orientation of FTs. Dynamic RBDs (DRBDs) [20] model failures and repairs of components based on their dependencies using state machines. To analyse these models, an algorithm which converts DRBDs into Coloured Petri Nets was presented in [21]. However, an integrated modelling and analysis toolset for DRBDs is lacking. Signoret et al. in [22] presented an approach called RBD driven Petri nets (RdP) which uses RBDs as an interface to build large Petri nets systematically. The modular characterization of Petri nets enables the intuitive creation of RdP models from predefined module libraries. Aligned with these formalisms, the OpenSESAME modelling environment connects RBDs and state-based formalisms [23]. It enables an straightforward evaluation of highly available system's fault tolerant behaviour including system *dynamics*. Its input models are based on RBDs and failure dependency diagrams, while component and repair tables are used to indicate component-specific characteristics (MTTF, MTTR) and repair groups. To carry out system analyses, these models are transformed into state-based SPN and Stochastic Process Algebra (SPA) models.

Lopatkin et al. [24] utilise FMEA models for system formal specifications. The approach defines generic patterns establishing direct correspondence between FMEA and state-based Event-B [25] formalism. The characterization of error detection and recovery patterns lead to analysing and verifying whether safety properties are preserved in the presence of faults. The use of these patterns, allows tracing from static FMEA considerations towards system *dynamics*.

Progression in the conjoint use of event and state formalisms is reflected with Boolean logic Driven Markov Processes (BDMP) [26]. BDMP employs static FT as a structure function of the system and associates Markov processes to each leaf of the tree. Similarly, State-Event Fault Tree (SEFT) [27] formalism combines elements from FT with both statecharts [28] and Markov chains, increasing the expressiveness of the model. SEFT deals with functional and failure behaviour, accounts for repeated states and events and allows straightforward transformations of SEFT models into Deterministic and Stochastic Petri Net (DSPN) models for state-based analysis.

In [29], Steiner et al. described a process to create and analyse SEFTs based on the ESSaRel tool [30]. Subsequently, the SEFT models are exported and analysed by means of a DSPN analyser tool called TimeNET [31]. The model described in Figure 1, shows some of the SEFT's strengths and characteristics in a simple behaviour model: (1) combination of statecharts with Markov chains by means of states (S) and events (E); (2) compositional characterization of system functionalities connected through required inports and provided outports (event_out_vel_ok, event_out_vel_high), which enables linking components in a FT-like structure while managing system's complexity; (3) characterization of functional and failure behaviour of the system, i.e., functional state (out_vel_ok), failure state (out_vel_high), and transition events between these states (no_obstacle_detected and sensors_working); and (4) underlying transformable logic to analyse with a target state-based formalism.

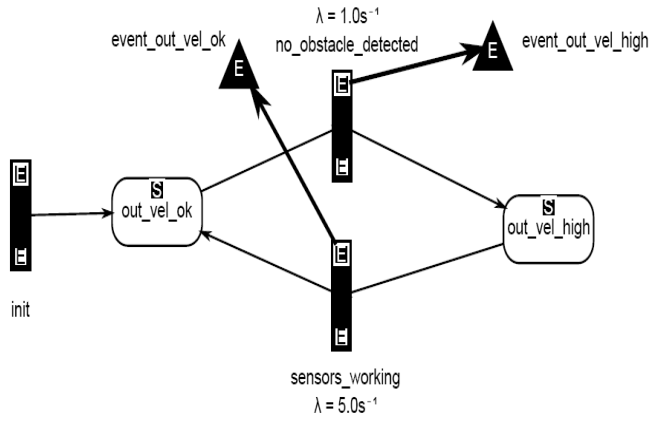

Figure 1. SEFT model example [29].

Likewise, El Ariss et al. in [32] presented an approach called Integrated Functional and Safety Specification (IFSS) model, in which they provide a systematic transformation from FT models into statecharts accounting for the temporal behaviour of faults. As a result of this integration process, they eliminate inconsistencies when using functional and non-functional models and account for the order of failure occurrences.

Table I displays among the addressed approaches those which has tool support for the specification and analysis of the dynamic behaviour of systems.

Table I
TOOL-SUPPORT OF THE DYNAMIC APPROACHES

Approach - Work	Tool Support	Type of Tool	Latest Release
DFT	e.g., Galileo [33]	Commercial, Educational	2003
Rao et al. [17]	DRSIM tool	Internal	2009
MatCarloRe	RAATSS [34]	Academic evaluation copy	2012
RdP	BStoK [35]	Comercial	2011
OpenSESAME	OpenSESAME [36]	Available	2009
Lopatkin et al. [24]	Rodin Plugin [25]	Available	2012
BDMP	KB3 Workbench [37]	Available	2012
SEFT	ESSaRel extension & transformation [29]	Internal*	2013

* Available for research purposes under agreement

DFT is a well-known mature approach for the evaluation of the system's *dynamics*. It has been adopted with different tools over the last years (e.g., Galileo [33]). In contrast, approaches such as IFSS, DRBD or SEFTs do not have a integrated tool support. The compositional and transformational features of the SEFT approach, provide an adequate abstraction of the system structure and behaviour. Developing a model-based tool which extracts DSPN models from SEFT models automatically would create an adequate environment for building a sound approach for manageable and consistent dependability analyses.

B. Compositional Failure Propagation Analysis Approaches

The common factors for Compositional Failure Propagation (CFP) approaches are: the characterization of the system architectures by design components; the annotation of the failure behaviour of each of them and the system failure analysis based on inter-components influences. Conceptually they all are very similar: the main objective of CFP approaches is to avoid unexpected consequences resulting from the failure generation, propagation and transformation of components.

Generally, CFP approaches characterise the system as component-wise developed FT-like models linked with a

causality chain. System architectural specifications and subsequent dependability analyses of CFP approaches rely on a hierarchical system model. This model comprises components composed from subcomponents specifying system structure and/or behaviour. CFP approaches analyse the system failure behaviour through characterizations of individual components, which lead to achieving a manageable failure analysis procedure. Failure Propagation and Transformation Notation (FPTN) [38], Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) [39] and Component Fault Tree (CFT) [13] are the principal CFP approaches. Their main difference is in the failure annotations of components, which specify incoming, outgoing and internal failures to each component. In order to annotate the logical combinations of these failures, FPTN uses logical equations, HiP-HOPS makes annotations using Interface Focused FMEA (IF-FMEA) tables and CFT associates to each component individual FTs. Subsequently, the connections between system components determines the *failure flow* of the system, linking related failure annotations of each component.

Concerning the different contributions of CFP approaches, FPTN first addressed the integration of system-level deductive FTA (from known effects to unknown causes) with component-level inductive FMEA (from known causes to unknown effects). HiP-HOPS integrates design and dependability analysis concepts within a hierarchical system model. However, instead of exclusively linking functional components with their failure propagations like in FPTN, first the hierarchical system model is specified and then, compositional failure annotations are added to each component by means of IF-FMEA annotations. These annotations describe the failure propagation behaviour of the component in terms of outgoing failures specified as logical combinations of incoming and internal failures (cf. Figure 2).

From the IF-FMEA annotations shown in Figure 2, the outgoing failures at the port *out_1* will be specified as follows: *omission-out_1 = omission-in_1 AND omission-in_2 OR Stuck at 0*. Once characterized all the outgoing failures of all the ports, a FT synthesis algorithm analyses the propagation of failures between connected components. Traversing the hierarchical system model, while parsing systematically the IF-FMEA annotations of its constituent components, allows the extraction of the system FT and FMEA models.

CFTs are a model-based extension of FTA models. The component FTs can be combined and reused to systematically obtain the FT for any failure without having to create and annotate a FT for each failure. In order to integrate analysis and design concepts, it has been extended in [40] resulting in the Safe Component Model (SCM) approach. The approach separates components' functional/failure specification and realization views and through the integration of the failure propagation and hierarchical abstraction, SCM allows obtaining a hierarchical component based abstraction

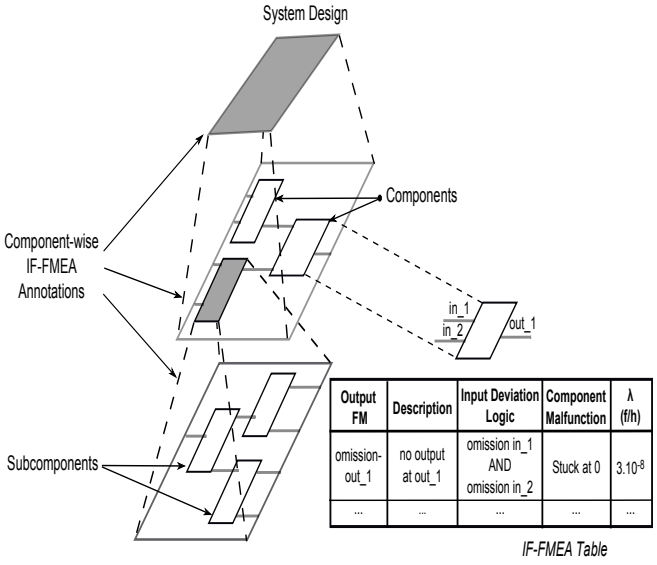

Figure 2. Hierarchical structure and CFP annotations in HiP-HOPS.

of CFTs.

They all have been extended to cope with occurrences of temporal events. Temporal extensions for FPTN [41] and HiP-HOPS [42] have been influenced by the DFT methodology. They concentrate on the analysis of non-repairable systems examining the order of events so as to identify specific sequence of events leading to failures. Integration of CFT concepts with state-based techniques resulted in the SEFT formalism, which is able to handle availability and maintainability properties of repairable systems.

Transformation of CFP approaches into dependability analysis formalisms is an ongoing research subject (see Subsection II-C). HiP-HOPS extracts FTA and FMEA models thanks to its underlying logic and SEFT applies translation schemes to generate DSPN models.

Other interesting extensions include mechanisms to automate and reuse analysis concepts. Failure Propagation and Transformation Calculus (FPTC) [43] approach adds the characterization of the nominal behaviour to FPTN models and generalizes the FPTN equations to improve the manageability and analysability. Moreover, an algorithm is implemented to handle the general inability of CFP approaches to cope with cyclic dependencies of feedback structures. In [44], general failure logic annotation patterns were defined for HiP-HOPS. Similarly, the CFP approach presented in [45] by Priesterjahn et al., emphasizes the reuse of failure propagation properties specified at the port level of components. These specifications centre on the physical properties of different types of flows, which allow reusing failure behaviour patterns for functional architectures.

Concerning the model-based generation of FMEA models, in [46] Struss and Fraracci presented an approach to extract FMEA models mechanically. To do so, they based on devia-

tion models which describe formally the constraints of system functionalities and it provides the necessary predictions for extracting FMEA models. The approach implements reuse mechanisms through the use of generic libraries and it has been implemented in the Raz'r tool [47].

The evolution of CFP approaches focus on reusability, automation and transformation properties. Since the annotations of the components' failure behaviour depend upon designers experience, reusing failure annotations leads to reducing the error proneness. Based on the fact that different dependability analyses have to be performed when designing a system, definition of a unique consistent model covering all analyses would benefit these approaches. This is why recent publications in this field centre on integrating existing approaches (cf. Subsection II-C and Section IV).

Regarding the tool support of the CFP approaches (cf. Table II) we can see that all approaches have been turned into toolsets. Nonetheless, the CFP approaches are moving one step further, integrating design languages in order to link the design and analysis processes (cf. Subsection II-C).

Table II
TOOL-SUPPORT OF THE CFP APPROACHES

Approach - Work	Tool Support	Type of Tool	Latest Release
FPTN	SSAP Toolset [38]	Unavailable	2006
HiP-HOPS	HiP-HOPS Tool [39]	Available	2012
CFT	ESSaRel tool [30]	Available	2009
SCM [40]	ComposeR	Internal	2012
FPTC	Epsilon [48]	Available	2009
Priesterjahn et al. [45]	MechatronicUML, Fujaba [49]	Available	2012
Struss & Fraracci [46]	Raz'r Tool [47]	Comercial	2012

C. Model-Based Transformational Approaches

The main aim of the transformational approaches is to construct dependability analysis models (semi-)automatically. The process starts from a compositional design description using computer science modelling techniques. The failure behaviour is specified either by extending explicitly the design model or developing a separate model, which is allocated to the design model. Transformation rules and algorithms extract dependability analysis models from it.

These approaches lead to adopting trade-off decisions between dependability design and analysis processes. On one hand, the automation and reuse of analysis techniques in a manageable way makes it a worthwhile approach for design purposes. The impact of design changes on dependability attributes are analysed systematically. On the other hand, from purist's point of view of classical analysis techniques, the automation process removes the ability of these techniques to identify and analyse hazards or malfunctions in a comprehensive and structured way.

Architectural Description Languages (ADLs) provide an adequate abstraction to overcome the limitations. Simulink [19], AADL [50] and UML [51] have been used for both architectural specification and failure behaviour specification. UML is a widely used modelling language, which has been extended for dependability analyses following Model Driven Architecture (MDA) concepts [52]. Namely, profiles allow extending and customizing modelling mechanisms to the dependability domain [53].

Lately, a wide variety of independently developed extensions and profiles have come up for dependability analysis [54]. However, some generally applicable metamodel is lacking. In an effort to provide a consistent profile CHESS ML emerged [55]. CHESS ML provides all necessary mechanisms to model dependable systems and extract either event-based (FMECA, FPTC) or state-based (SPN) analysis models. Recently in [56], a model-driven failure logic analysis method called CHESS-FLA has been presented. This approach is built upon Formalism for Incompletion, Inconsistency, Interference and Impermanence Failures' Analysis (FI^4FA) approach [57] which is based on the concepts of FPTC. FI^4FA differs from FPTC in that it offers two additional features. Namely, it provides mechanisms to analyse more types of failures than FPTC and it allows modelling and analysing the mitigation behaviour. The CHESS-FLA approach supports back-propagation of the results to ease the readability of the analysis results.

CFP approaches have been shifted towards the transformational paradigm. The toolset for FPTC approach [43] relies on a generic metamodel in order to support transformations from SysML and AADL models. In [58], a metamodel is developed so as to extract CFT models from functional architecture models specified in UML. This process permits the generation of reusable CFT models consistent with the design model. In the same line, integration of HiP-HOPS model with EAST-ADL2 automotive UML profile is presented in [59]. Translations from high-level ADLs to well established CFP analysis techniques, enable an early dependability analysis and allow undertaking timely design decisions.

Architecture Analysis and Design Language (AADL) captures the system architectural model in terms of components and their interactions describing functional, mapping and timing properties among others. The core language can be extended to meet specific requirements with annex libraries. Behaviour and error model annexes are provided with the tool. The error annex links system architecture components to their failure behaviour specification making possible the analysis of the dependability attributes of the system. It has been used for both state-based [60] and event-based [61] analysis.

AltaRica [62] is a dependability language, which enables describing the behaviour of systems when faults occur. The model is composed of several components linked together representing an automaton of all possible behaviour scenarios, including those cases when reconfigurations occur due to the occurrence of a failure [63]. It is possible to process such models by other tools for model-checking or generation of FTs [64]. Transformations from AADL to AltaRica are presented in [65], based on MDA concepts so as to perform dependability analyses and avoid inconsistencies while working with different formalisms.

Riedl and Siegle presented a language for the specification of reconfigurable and dependable systems called LARES [66]. It expresses system's fault tolerant behaviour using a generic language in which any kind of discrete-event stochastic system can be specified. It is based on fully automated model transformations to measure systems dependability. Namely, transformations into TimeNET [31] and CASPA [67] tools are carried out in order to solve state-based SPA and SPN models respectively.

In [68], Cressent et al. defined a method for RAMS analysis centred on SysML [69] modelling language from where a FMEA model is deduced. SysML diagrams define a functional model connected to a dysfunctional database enabling the identification of failure modes. This database contains the link between system architecture and failure behaviour giving the key for FMEA extraction. Further, the methodology for dependability assessment is extended using AltaRica, AADL and Simulink models. They address reliability and timing analyses and simulation of the effects of faults respectively.

Definition of a model for the extraction of all necessary formalisms for dependability analysis is the common goal for the aforementioned works. Interconnections between different formalisms in order to take advantage of the strengths of each ADL, allow analysing dependability properties accurately. AltaRica and AADL cover adequately the analysis of reliability, availability and maintainability attributes. Extraction of the main CFP approaches from ADLs should help to analyse comprehensively system safety properties. Moreover, Simulink model simulations allow evaluating the effects of failure and repair events in the system. Thereby, integrations between language specific models like in [68] helps evaluating accurately all dependability aspects of a system.

The acceptance of the transformational approaches depends on the availability of toolsets capable of performing (automatic) transformations. As it is shown in Table III, all ADLs have their own implementation toolsets. Namely, transformations from ADL models into CFP models have been carried out through metamodels and profiles implemented as plugins.

Table III
TOOL-SUPPORT OF THE TRANSFORMATIONAL APPROACHES

Approach - Work	Tool Support	Type of Tool	Latest Release
Simulink	Matlab [19]	Comercial	2012
UML, SysML	e.g., Eclipse Papyrus [70]	Available	2012
AltaRica	e.g., AltaRica Tools [71]	Available	2013
AADL	e.g., Osate [72]	Available	2012
CHESS-ML	CHESS Plugins [73]	Partially available	2012
FPTC	Epsilon [48]	Available	2009
Adler et al. [58]	CFT UML Profile	Internal	2012
HiP-HOPS	EAST-ADL2 Eclipse Plugin [74]	Available	2010
LARES [66]	LARES toolset	Internal	Ongoing
Cressent et al. [68]	MéDISIS Framework	Internal	Ongoing

D. Classification of Techniques

In order to classify the covered approaches, Table IV groups them taking into account limitations specified in Section II.

Table IV
SUMMARY OF LIMITATIONS OVERCOME BY APPROACHES

Group	Approach	Limitations
1	[12] [16] [17] [18] [23] [26]	L1
2	[13] [38] [40] [43] [46]	L2, L4
3	[39] [45] [56] [61]	L2, L3, L4
4	[20] [22] [32] [41] [63]	L1, L2, L4
5	[24] [27] [42] [55] [60] [66] [68]	L1, L2, L3, L4

Approaches gathered within the group 5 contain all necessary features in order to analyse dynamic systems consistently and in a manageable way. Compositional failure annotation, dynamic behaviour and automatic extraction of analysis models are the key features addressed by these approaches. Utilization of failure annotation patterns promote flexibility and reuse and consequently, reduce the error proneness. Nevertheless, as noted in [75], characterization of the failure behaviour of components depends on the component context, which conditions compositional and reuse properties. Moreover, automatic generation of the analysis model does not completely alleviate the dependency on the knowledge of the analyst. However, it lets managing and specifying the failure behaviour in a clear and consistent way.

III. Design of Dependable Systems: Trade-Off Between Dependability and Cost

Generally, dependability design decisions and objectives are related to trade-off decisions between system dependability attributes and cost. Dependability requirements often conflict with one another, e.g., safety-availability compromise when a faults leads the system to a safe shutdown in order to prevent it from propagating. The time at which design decisions are taken determines the cost that the design process can incur.

Designing a dependable system within considered requirements requires a process to match and tune combination of architectural components so as to find an optimal solution satisfying design constraints. There are other approaches concentrated on the design of dependable systems under the correct-by-construction paradigm. For instance, the approach presented in [76] creates a formal system specification preserving the correctness through gradual refinements of the system design model. However, instead of addressing formal correct-by-construction dependable design approaches, we will overview those approaches which are aimed at characterizing at design time the implications of design decisions (combination of components) on dependability and cost.

More specifically, we group dependable design approaches by looking at how system recovery strategies are implemented. Traditionally, hardware replication has been viewed as a feasible solution to recover from failures, e.g., N modular redundancy. Nonetheless, there are other kinds of recovery strategies which make the repair possible reusing already existing hardware components. On one side, in Subsection III-A we group those approaches that replicate the nominal functionality by aggregating additional hardware resources, i.e., *homogeneous redundancies*. On the other side, in Subsection III-B we group those approaches which are aimed at reusing hardware components to provide a compatible functionality and reduce hardware costs, i.e., *heterogeneous redundancies* [77]. Finally, in Subsection III-C, we have summarized and characterized these approaches with respect to our design criteria and identified key design activities to progress in the use of *heterogeneous redundancies*.

A. Design of Dependable Systems by Means of Homogeneous Redundancies

The principal question addressed by the approaches grouped in this subsection is the evaluation of the effect of design choices (e.g., robustness level of components, redundancy configurations) on dependability and cost.

Cauffriez et al. [78] and Clarhaut et al. [79] focused on designing a dependable system based on a design methodology presented in [80]. The main focus of this methodology relies on the early and systematic characterization of dependability criteria during the system design activities. As it is shown in Figure 3, the approach comprehends three types of architectures: functional architecture, equipment architecture and operational architecture.

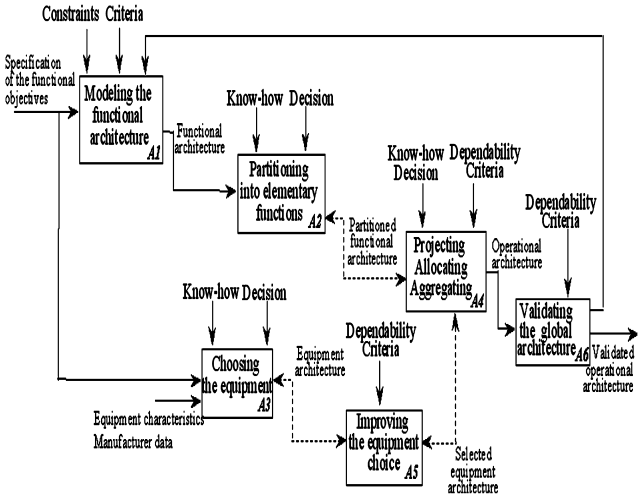

Figure 3. Methodology for designing distributed control systems [80].

The design process starts from the characterization of functional and equipment architectures addressing functional and dependability criteria. Subsequently, the allocation of the functional architecture onto the equipment architecture is evaluated in relation to dependability. As a result, the operational architecture is produced which could require reconsidering functional and/or equipment decisions in order to obtain a validated operational architecture with respect to dependability requirements.

Cauffriez et al. in [78] concentrated on the analysis of repairable architectures evaluating how the use of alternative hardware components affects system functionality and dependability. To do so, they characterized system-level functions in a top-down manner until lowest level subfunctions are reached. At the bottom layer, failure and repair rates of hardware components permit analysing system top layer's performance, reliability and availability using Monte Carlo simulations. In this way, a structural function is characterized which links functions and hardware resources and allows evaluating alternative operational modes by associating different subfunctions to perform the system-level function. The overall design methodology for modelling and analysing alternative architectural design choices has been integrated within a design tool.

Clarhaut et al. described a design approach overcoming the static-logic limitation of event-based analysis techniques (cf. Subsection II-A) by identifying sequential component-wise contributions to system-level failures [79]. During the design process, a *functional hierarchical tree model* characterizes dependencies between functions and hardware resources. This model accounts for alternative resources and hence, architectures to perform the modelled control func-

tions. Subsequently, the *Improved Multi-Fault Tree* (IFT) is constructed characterizing sequential failure relationships between components' failure modes (FM) and system functions which lead to the system-level undesired effects designated as dreaded events.

As shown in Figure 4, the structure of the design methodology revolves around the characterization, analysis, and optimization of system architectures so as to adopt optimal design decisions regarding dependability and cost. The IFT determines the dependability level of the overall architecture weighting the contribution of each component to the system-level dreaded events. Architectural design choices cover active and passive redundancies. The cost associated with each hardware component enables progressing between alternative architectures toward an optimal architecture which maximizes dependability and reduces hardware cost.

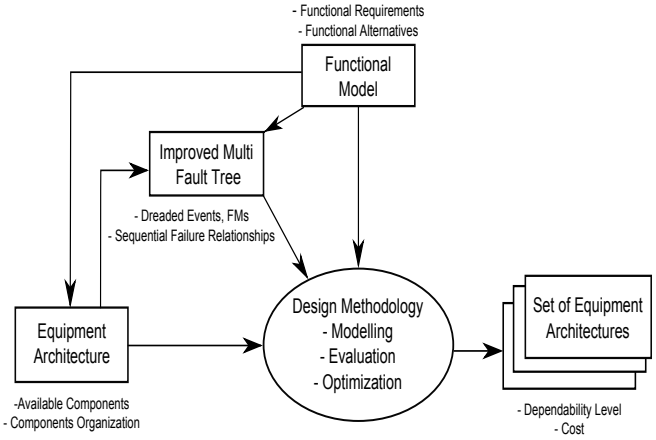

Figure 4. Clarhaut et al.'s design approach.

Adachi et al. in the work presented in [81], extended the HiP-HOPS approach with recovery strategies in order to design optimal architectures reducing cost and increasing dependability. The recovery strategies are formally represented using patterns. These patterns characterize the potential to detect, mitigate, and block affecting component failures which are previously identified with HiP-HOPS and analysed by means of FTA and FMEA. Finally, starting from an abstract architecture, recovery strategies are introduced without violating user constraints and an evolutionary optimization algorithm allows converging through dependability and cost requirements.

All the covered approaches aim at increasing system dependability through the replication of nominal components. This design decision implies a cost increase. Consequently, this drawback needs to be justified through an exhaustive and adequate analysis of how the system design meets functional and dependability requirements.

B. Design of Dependable Systems by Means of Heterogeneous Redundancies

One of the key properties of the systems which exercise *heterogeneous redundancies* is the ability to successfully accommodate changes in case of failure occurrences. Consequently, the system design approaches that we will cover in this subsection not only address dependability issues, but also adaptation capabilities. Thus, we group them as adaptive dependable design approaches.

Shelton and Koopman first worked on the concept of *heterogeneous redundancies* by means of *functional alternative* strategies. It allows to compensate for component failures changing the system functionality [77]. The approach models alternative system configurations and assigns a relative utility value to each of them weighing their contribution to the system performance and dependability. From this model, system's overall utility value is calculated which enables the evaluation and comparison of design choices as to where allocate resources for functional alternatives or redundancy. This characterization make it possible to evaluate how component failures affect system utility.

Wysocki et al. in [82] addressed the same design strategy under the *shared redundancy* concept. They concentrated on the reuse of processing units through the strategic distribution of software modules. Consequently, given the failure occurrence of a software component, it is possible to still continue operating through the reconfiguration of communication routes. To evaluate the reliability and safety of the alternative architectures, first a FTA is carried out. This analysis permits extracting minimal combination of events which leads the system to failure. Additionally, this information is used as input for further analysis through Design Of Experiments (DOE) to calculate system cost and failure probabilities. Based on the same design concept, Galdun et al. in [83] analysed the reliability of a networked control system structure using Petri Nets (PN) and Monte Carlo simulations.

Rawashdeh and Lump in [84] presented a framework for designing reconfigurable architectures for fault tolerant embedded systems called ARDEA. The approach is based on processing units' reconfigurations to achieve *graceful degradation* and cope with hardware failures. A gracefully degrading system tolerates system failures by providing the same or equivalent functionality with the remaining system components. Dependency Graphs (DGs) are used to model the functional flow of information from input to output considering alternative implementations. A centralized system manager uses DGs and a hardware resource list to find a viable mapping of software on the available processing units. It decides when to schedule/un-schedule software modules moving object code among available processing units without exceeding processor time and bandwidth.

In the MARS project, Trapp et al. [85] proposed a component based modelling and analysis method to exploit *implicit redundancies* so as to react to system failures by reusing hardware resources. It provides methodological support for modelling and gathering system configurations. Moreover, reasonable system configurations are elicited from a set of possible candidates. The system's adaptive behaviour is modelled based on quality types, which drive the system's graceful degradation possibilities. The quality type system determined at design time and modelled with a system inheritance tree, defines the possibilities for exchanging quality types between components.

Each system component operates under different configurations determined by Quality Attributes (QAs). These QAs are attached to each component's every I/O port. Configuration activation rules are defined over these ports based on the needed QAs to activate a configuration (preconditions) and provided QAs by this configuration (postconditions) (cf. Figure 5).

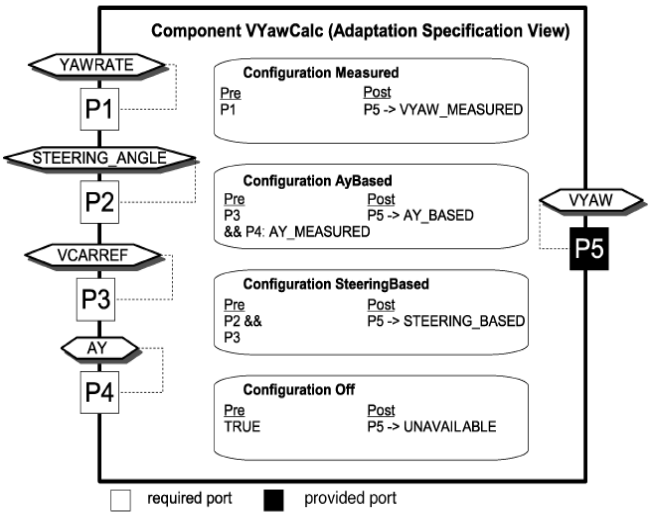

Figure 5. Example of an adaptation specification view [86].

As it is shown in Figure 5, for each component within the hierarchical system architecture, its possible configuration variants are defined. Each port has linked its own quality type which will determine the corresponding constraints over the port. Each configuration has its own activation preconditions and propagation postconditions defined over the component ports' QAs. This characterization leads to determining compatible components based on QAs. As a result, component compositions are abstracted into hierarchical components with a explicitly defined adaptation behaviour.

From this modelling paradigm (MARS modelling), different analyses have been carried out. In [87], transformations of MARS models into hybrid-CFT were performed in order to calculate configuration probabilities (cf. Subsection II-A). Moreover, in order to ensure the causality of the reconfiguration sequences and safety-related properties, verification activities have been carried out in [86]. Last but not least, methodological support for identifying an adaptation model meeting availability-cost trade-off were addressed in [88].

Similar design concepts are addressed in the avionics field. Namely, the Integrated Modular Avionics (IMA) design paradigm defines robust partitioning in on-board avionic systems so that one computing module (Line Replaceable Unit - LRU) is able to execute one or more applications of different criticality levels independently. The standardised generic hardware modules forming a network leads to looser coupling between hardware and software applications [89].

SCARLETT project [90] aims at designing reconfigurable IMA architectures in order to mitigate the effect of failures of functional, fault detection and reconfiguration implementations. Monitoring and fault detection functions aim at detecting component failures. Once a permanent failure is detected, the reconfiguration supervisor performs two key activities. Firstly, it manages the modifications given the current configurations and failed module and secondly, it checks the correctness of the system configuration and the loaded data in the LRU. The centralized supervisor determines a suitable configuration based on a reconfiguration graph, which contains all possible configurations. Reconfiguration policies and real-time and resource constraints, define the set of reachable safe transitions and states. In order to analyse the reconfiguration behaviour when failures occur, a safety model leads to finding the combinations of functional failures [91]. Based on the same concepts, DIANA project [92] aims at distributing these functionalities. This approach improves the availability of the reconfiguration mechanisms at the expense of relying on a complex, resource consuming communication protocol.

C. Summary of the Design Approaches

In order to characterize the reviewed approaches within this section, the following design properties have been described in the Table V:

1) Type of recovery strategy.
2) Dependability analysis approach.
3) Cost evaluation.
4) Other tasks: optimization or verification.

Since the use of *heterogeneous redundancies* requires considering system *dynamics*, the dependability analysis approaches described so far address system's temporal behaviour either by linking event-based static-logic approaches with state-based formalisms (e.g., Hybrid-CFT) or evaluating through approaches which integrate the temporal behaviour explicitly (e.g., MCS, DFT, PN) (cf. Subsection II-A). Moreover, given the extra design complexity of the systems which use *heterogeneous redundancies*, the mechanisms which help structuring the analysis and

Table V
APPROACHES AND ADDRESSED DESIGN PROPERTIES

Works	1	2	3	4
[78]	Homogeneous Redundancies	MCS	HW cost	NA
[79]	Homogeneous Redundancies	IFT	HW cost	Optimization
[81]	Homogeneous Redundancies	HiP-HOPS	HW & SW cost	Optimization
[77]	Heterogeneous Redundancies	Utility Values	NA	Optimization
[82] [83]	Shared Redundancies	FTA, MCS, DOE; PN, MCS	Mainten. cost	NA
[84]	Graceful Degradation	NA	NA	NA
[85]	Implicit Redundancy	Hybrid-CFT	HW & SW cost	Optimization, Verification
[90] [92]	Reconfigurable IMA	Safety analysis, AltaRica	NA	NA

Legend: *NA*: Not addressed; *Mainten*: Maintenance

reusing the models are necessary (e.g., libraries, hierarchical abstractions).

In order to obtain a predictable system design and avoid unexpected failure occurrences, all the approaches address design-time determined reconfigurations. Nonetheless, it is necessary to go beyond and overcome their underlying assumptions concerning the system's critical functionalities to perform reconfigurations effectively. Namely, among all the reviewed approaches only [91] takes into account a possible faulty behaviour of the fault detection and reconfiguration implementations. Similarly, the faulty behaviour of the communication network is only explicitly addressed in [83]. The evaluation of the possible faulty behaviour of these implementations leads to obtaining an approach which better adheres to the reality and consequently, more reliable results. Despite not addressing *heterogeneous redundancy* like concepts directly, in [93] an approach called component logic models is presented which does address the faulty behaviour of fault detectors. To this end, it associates modes with services and in the case of detection measures each service-port evaluates the veracity of the fault detectors (e.g., modes of false veracity: false positive and false negative).

As a result, from our perspective it is necessary cover the following design activities:

- *A1*: Systematic identification of *heterogeneous redundancies* and extraction of system configurations to react in the presence of failures.
- *A2*: Design of the system architecture to make the use of *heterogeneous redundancies* possible, i.e., fault detection and reconfiguration implementations.
- *A3*: Evaluation of the system dependability with respect to dependability and adaptivity constraints.

The first design activity calls for an approach which allows identifying systematically existing hardware components to provide a compatible functionality. To the best of our knowledge, only the work we presented in [94] works towards this goal. In [86], Adler et al. worked on the systematic extraction of system configurations annotating component by component their adaptive behaviour. During this process they evaluate in a ad-hoc manner if it is possible to provide another configuration variant using alternative hardware components and finally extract system configurations based on inter-component influences.

The second activity requires addressing design decisions regarding the organization of fault detection and reconfiguration implementations, i.e., their distribution and replication. On one hand, when implementing the fault detection function within a networked control system, it is possible to allocate it either on the source Processing Unit (PU) where the information is produced (e.g., sensor, controller) or in the destination PU, which is the target PU of the source information (e.g., controller, actuator) or in both PUs. On the other hand, when dealing with reconfiguration implementations, its distribution influences the overall dependability and cost of the system (cf. Table VI).

Table VI
DESIGN DECISIONS AND INFLUENCED ATTRIBUTES

Design / Attribute	Fault Detection		Reconfiguration	
	Source	*Destination*	*Centralised*	*Distributed*
Dependability	Det. at origin, unable to cope with comm. failures	Det. of wrong value & omission. Prone to CCF	SPOF	Multiple reconfig. redundancies
Cost	HW/SW implementation costs	Costly id. of all failures: failure transf.	Single reconfig. HW/SW costs	Higher cost: multiple reconfigs
Complexity	Direct failure handling	Further failure sources	Less comm. overhead	Complex comm; resource handling

Legend: *SPOF*: Single Point of Failure; *CCF*: Common Cause Failure; *comm*: communication; *det*: detection; *reconfig*: reconfiguration; *id*: identification; *transf*: transformations.

Additionally, when adopting design decisions within the second activity, it is necessary to address adaptivity constraints which also has influence on dependability, e.g., *timeliness constraints*: maximal duration in which the adaptation of one component can be performed [95], *dependency constraints*: dependencies between system components, where adapting one component requires further adaptation on other components [86] or *hardware resource constraints*: limit the use of hardware resources, e.g., processing power, memory [84].

In the third activity, previously adopted system design decisions are analysed with respect to dependability and cost. To this end, we include the alternative configurations and possible faulty behaviour of the fault detection, the reconfiguration and the communication implementations.

IV. Model-Based Verification: From Fault Injection to Integrative Approaches

Fault Injection (FI) approaches concentrate on verifying system behaviour in the presence of faults according to target dependability properties. The outcome of this process may lead to considering design changes. However, changes adopted late in the design process are costly. This is why we focus on model-based FI approaches adopted at the preliminary design phase. This process is based on the analyst's knowledge to reason about the functional, failure and recovery behaviour of the system. Timely evaluation of these properties provides a valuable feedback for design purposes. However, difficulties arise from the accuracy of the system behaviour, which requires an accurate knowledge of the system.

Within the model-based FI approaches, the verification process is characterized as follows: first, a functional model is specified which is then converted into an *extended system model* accounting for the functional and failure behaviour of the system. Temporal logic languages are used to define system *requirements*. They describe how the truth values of assertions change over time, either qualitatively (Computation Tree Logic (CTL), Linear Time Logic (LTL)) or quantitatively (Continuous Stochastic Logic (CSL), probabilistic CTL (PCTL)). Model-checking (MC) engine assesses whether such requirements are met or not by the extended system model, using a *analysis model*. To do so, it is necessary to transform the extended system model into the analysis model. When the analysis model fails to meet these requirements, its effects are deduced automatically identifying the paths that violate the conditions (counter-examples (CEs)) [9]. The logical orientation of this analysis process results in FMEA-like cause-effect analysis.

Automatic transformation from extended system models to MC analysis models and definition of predefined libraries for correct and complete failure specification and injection are the main points in order to obtain a consistent and robust FI process. ESACS project [96] addressed these characteristics prominently providing mechanisms to extend the model straightforwardly and extract FTA models from it. Other approaches concentrated on the generation of (probabilistic) FMEA (pFMEA) models [97] [98]. Application of pFMEA models, improvement in the interpretation of counter-examples [97] and automated tool support for injecting faults and analysing its consequences [98] are their main contributions.

Albeit these approaches provide a means to extract classical dependability models from the FI process, few of them concentrate on integrating existing CFP approaches. There are some incipient works linking CFP and verification approaches. They are influenced by HiP-HOPS [99] and FPTC [100]. Both approaches address the integration of qualitative design models with quantitative analysis via probabilistic MC. In [99], Gomes et al. described an approach to verify quantitatively system's dependability requirements. To do so, they did a systematic transformation of Simulink models into CTMC models by means of PRISM model checker [101]. The annotation of the failure behaviour is carried out as in HiP-HOPS using IF-FMEA. In [100] Ge et al. proposed a probabilistic variant of FPTC called Failure Propagation and Transformation Analysis (FPTA). The approach links architectural models with probabilistic model checkers specified in PRISM. The ins and outs of these approaches are grouped in the Table VII.

Results extracted from this process helps verifying if a system behaves as intended in the presence of failures. Concerning the analysis of dependability attributes, generated counter-examples as well as extraction of classical dependability analysis models, provides useful mechanisms to evaluate these attributes. However, there are some limitations hampering the analysis and interpretation of these approaches. Representation structures of the results, state-explosion problems, technical specification difficulties and model inconsistencies are some challenges to be addressed.

Due to the complexity and difficulties emerged from these approaches there have been a shift in the use of model-based FI approaches. Instead of developing purely verification oriented FI approaches, model-based *integrative verification* approaches are gaining support. These works result from the integration of design, analysis and verification tasks. They are aimed at combining dependability analysis techniques examined within the group 5 (cf. Table IV) with FI approaches. They express system behaviour using a compositional model, which gathers nominal, failure and recovery behaviours. Integrating approaches using model transformations, allows using a single design model for dependability and verification analyses. As a result, limitations concerning the ease of use, consistency and completeness of the analyses and automated tool support are addressed.

Within the model-based integrative verification approaches, the overall verification process is specified as follows: the system *design model* specified using a Architectural Description Language (ADL) characterizes functional, failure and recovery behaviour and structure of system components. This design model needs to be transformed in a target *analysis model* which allows analysing and verifying system requirements. There are two options to carry out this transformation: (1) transformation of the design model to the target analysis approach through direct transformation rules; (2) intermediate transformation into a tool independent *Intermediate Model (IM)*, so that consistency and traceability between different design, analysis and verification approaches

Table VII
SUMMARY OF MODEL-BASED FAULT INJECTION APPROACHES

Works	*Extended Model*	*Analysis Model*	*Reqs.*	*Results*	*Specific Features*	*Tool Support*	*Limitations & Improvements*
[96]	NuSMV	NuSMV	CTL, LTL	FT	Model construction facility: library of FM and requirements; Unification of formal verification & safety analysis	FSAP/NuSMV-SA, Internally Available [102]	State explosion; FT structure; Probabilistic analysis; Failure ordering
[97]	CTMC (PRISM)	PRISM	CSL	pFMEA, CE	CE improvement; Auto. probability calculations; Multiple failures	PRISM [101]	State explosion; Pattern based FI
[98]	BT	SAL*	LTL	FMEA	FMEA automatic generation from BT models; Analysis reduction strategies; Effect analysis of multiple failures	BTE tool [103], SAL Symbolic MC [104]	CE readability; Timed FI; pFMEA; Reduction techs.
[99]	Simulink	PRISM*	CSL	Prob. calc.; CTMC	Systematic generation of analysis models from CFP design models	Partialy available [105]	No CE; State-explosion; Dynamic behaviour
[100]	Epsilon	FPTA, PRISM	CSL	CE	Integration of CFP approach and probabilistic model checking	FPTC toolset, available [48]	Failure prone manual transformation; Translate CE to design model

Legend: *CTL*: Computation Tree Logic; *LTL*: Linear Time Logic; *CSL*: Continuous Stochastic Logic; *CE*: Counter-Example; *BT*: Behaviour Tree; *FM*: Failure Modes; **Symbols**: *: Automatic Transformation;

are attained. In the second case, these models enable the reuse of the same high level models for different target approaches. Subsequently, the analysis of the corresponding approach (either FI, state-based or event-based analysis) makes use of the respective underlying characteristics (cf. Figure 6). FI approaches allows extracting counter-examples, which in turn can be transformed into dependability analysis models (indicated with dashed lines in Figure 6).

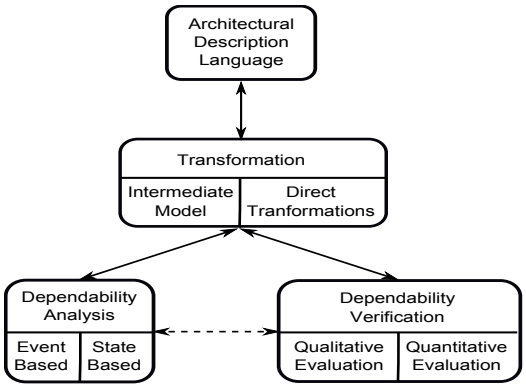

Figure 6. Structure of model-based integrative approaches.

The COMPASS project [106] is an integrative approach based on System-Level Integrated Modelling (SLIM) language which uses direct transformations [107]. The semantics of SLIM cover the nominal and error behaviour of AADL. The complete specification of SLIM consists of a nominal model, a failure model and a description of the effects of failures in the nominal model (*extended model*). Due to its underlying formal semantics, various types of analyses are possible: validation of functional, failure, and extended models via simulation and MC; dependability analysis and performance evaluation; diagnosability analysis and evaluation of the effectiveness of fault detection, isolation and recovery strategies.

Similarly, Güdemann and Ortmeier [108] proposed an intermediate tool-independent model called Safety Analysis Modelling Language (SAML). SAML describes a finite state automata, which is used to characterise the extended system model. This model specifies the nominal behaviour, failure occurrences, its effects and the physical behaviour of the surrounding environment. From this single model, quantitative and qualitative MC analyses are performed. The quantitative analysis, based on Deductive Cause Consequence Analysis (DCCA) [109], identifies minimal critical sets using CEs to indicate time-ordered combinations of failures causing the system hazard. The qualitative analysis, focusing on probabilistic DCCA (pDCCA), calculates per-demand and per-time failure probabilities.

Topcased project [110] aims at developing critical embedded systems including hardware and software. Topcased integrates ADLs and formal methods. The approach transforms high-level ADL models (SysML, UML and AADL) into an IM model specified in FIACRE language [111]. FIACRE specifies behavioural and timing aspects of high-level models making use of timed Petri nets primitives. Subsequent transformations of the IM model into MC tools (TINA and CADP) make it possible the formal verification and simulation of the specified requirements. TINA Petri Nets analyser [112] evaluates requirements specified in the state variant of LTL proposition logic (State/Event LTL (SELTL)) focusing on timeliness properties. CADP toolbox [113] transforms FIACRE models into LOTOS programs, which are handled by its underlying tools for validation via MC and simulation.

The pros and cons of the covered integrative works are summarized in the Table VIII.

The addressed works integrate well-known tools and formalisms. However, integration of analysis and verification

Table VIII
SUMMARY OF MODEL-BASED INTEGRATIVE APPROACHES

Works	*Design Model*	*Transf. Type*	*Analysis Model*	*Reqs.*	*Results*	*Specific Features*	*Tool Support*	*Future works*
[107]	SLIM	Direct	NuSMV*, MRMC*, RAT*	CTL, LTL, CSL	DFT, FMEA, prob. calc.	Req. patterns; Integrated verification, dependability and performance analyses of extended models	Toolset available for ESA members states [114]	Manual extension of the nominal model; Redundant FTA-FMEA results; State-explosion
[108]	SCADE, Simulink	IM: SAML	NuSMV*, PRISM*, MRMC*, Cadence SMV*	CTL, PCTL	DCCA, pDCCA	Combination of qualitative and quantitative analyses on the same model	S^3 tool [115]; publicly available	Manual extension of the nominal model; Transf. of ADLs (Simulink, Scade) into SAML; Req. patterns.; Library of FMs
[111]	SysML, UML, AADL	IM: FIACRE	TINA, CADP	LTL, SELTL	Timing, prob. calc.	Req. patterns; Integrated design, analysis and verification approaches	Topcased toolset [110]	State explosion; Back annotation of results

Legend: *IM*: Intermediate Model; *(P)CTL*: (Probabilistic) Computation Tree Logic; *LTL*: Linear Time Logic; *CSL*: Continuous Stochastic Logic; *SELTL*: State/Event LTL; *FMs*: Failure Modes; *Req*: requirements; *Transf*: transformations; **Symbols**: *: Automatic Transformation;

approaches when designing a dependable system is an ongoing research subject. There is an increasing interest in reusing and generalizing CFP approaches (e.g., transformation of CFP approaches into metamodels [43] [58] [59] and integration of CFP and verification approaches [100] [116]). Additionally, there exist other approaches closely related to the model-based integrative verification approaches, which cover the design process using formal analysis and verification approaches. For instance, under the correct by construction paradigm, the work presented in [117] matches with the idea of developing dependable systems by integrating specific approaches well-suited to each development phase. This methodology is based on a portfolio of different formalisms: starting from initial informal requirements, passing through formal requirements specification, modelling in Event-B and verification, towards the generation of executable code.

Interestingly, there still remain some challenges to be addressed. Back-annotation of the results into the design models would permit gaining more consistency between models. Additionally, it will lead to identifying the influence of the outcomes on the system components straightforwardly. Another issue is the size of the verification model extracted from the design model, which suffers from state-explosion problems.

The construction of a user friendly toolset integrating all these approaches is an issue itself. The last breakthroughs among integrative approaches' toolsets focus towards two main directions. On one hand, the automatic translations of design models into target specification formalisms is an unceasing goal [115]. These "push-button" toolsets avoid the designer to be exposed to the laborious, difficult and fail-prone process of creating a analysis model. For instance, in the COMPASS toolset [114], the designer is exposed to the SLIM formal specification language directly instead of using a ADL. However, one of the subtle problems underlying these tools is the size of the target models as a result of the design model transformations [110]. Another issue when creating these toolsets is related with the analysis results: the outcomes of the analysis results need to be displayed in a intuitive way to be understood by the designers and if necessary, take the corresponding countermeasures. To do so, they would benefit from an automatic transformation or back-propagation of the analysis results to the design model. Taking the automated tool support as an indicator to measure the potential to successfully integrate these approaches into the industrial practice, it is recognizable that all the covered approaches are working towards the construction of "push-button" automated toolsets to gain higher acceptance.

V. MODEL-BASED HYBRID DESIGN PROCESS

The goal of this section is not to provide a new design approach. Our aim is to make use of the reviewed analysis, design and verification approaches so as to outline a consistent and reusable model-based design process. This process arises from the structure of the model-based integrative verification approaches (cf. Section IV).

The separation of dependability analysis and verification tasks may lead to hampering the system design since results identified from either task need to be reconsidered during the design process (cf. Section III). On one hand, dependability analyses characterized by transformational approaches (cf. Subsection II-C), allow tracing from design models toward dependability analysis models. These approaches evaluate the dynamic system behaviour, as well as the effect of particular component failure occurrences at the system-level. On the other hand, purely verification oriented approaches mainly focus on the verification of the adequacy of the design model with respect to RAMS requirements. This is why we centred on covering integrative verification approaches.

When matching and tuning design components so as to find optimal design solutions satisfying design constraints, possible inconsistencies may arise due to the independent considerations of these approaches. This is why we should

focus on outlining a model-based hybrid design process, which unifies design, analysis and verification tasks. This process relies on initial system requirements, models, transformations and reuse of designer's considerations and results extracted from analysis and verification tasks (cf. Figure 7).

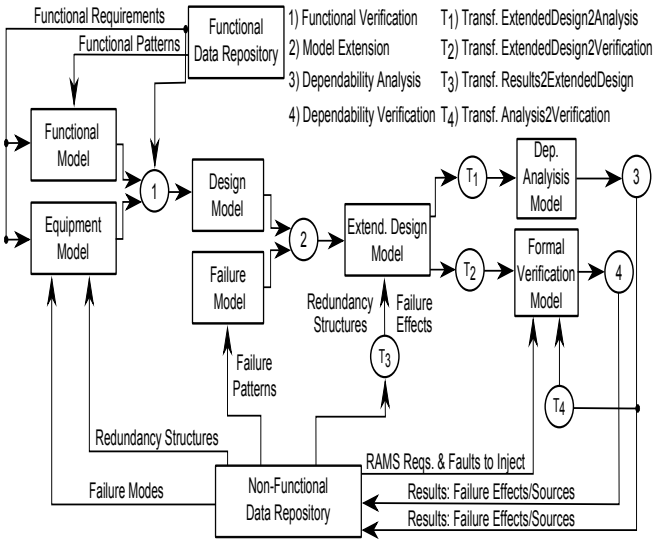

Figure 7. Model-based hybrid design process.

This design process starts from initial functional and physical characterizations. *Functional verification* analysis evaluates the adequacy of the allocation of the *functional model* into the *equipment model* according to functional requirements. The outcome of this process allows considering the verified *design model*. Subsequently, this model is *extended* with the *failure model* accounting for failure occurrences of the system. Failure patterns aid in the construction of the failure model allowing the reuse non-functional characterizations. Further, the effects of the considered failures and recovery strategies are annotated in the *extended design model* in order to counteract failure occurrences and its effects. With the aim to carry out *dependability analysis* and *formal verification* evaluations of the extended design model, twofold transformations need to be performed. The means to perform these transformations have been presented in Subsection II-C and Section IV respectively. Transformations of these models make the evaluation of the adequacy of the extended design model respect to RAMS requirements possible. Dependability analysis and verification tasks enable finding further failure effects and failure sources (apart from occurrence probabilities) either by CEs or dependability specific models. These results need to be transformed for design and analysis purposes. For the sake of reusing and refining the design process, data repositories have been considered consisting of annotation patterns for requirements and models (both functional and non-functional) and reusable recovery strategies.

On one hand, the outlined design approach enables benefiting from consistent design considerations. Moreover, data repositories allow the reuse of designer's characterizations as well as analysis results. Furthermore, user-friendly means make the annotation processes more evident. On the other hand, the automation of the extraction of dependability models hides information about the failure behaviour. Additionally, the flexibility of the approach depends on the system context, which would determine the reuse of functional and non-functional considerations.

VI. Conclusion and Future Work

Designing a dependable system, poses a wide variety of challenges on all its phases. This paper groups different approaches in order to identify and classify them.

The listed limitations guided the evolution of the analysis techniques towards Compositional Failure Propagation (CFP) and transformational analysis approaches. Automatic extraction of analysis models from design models is an ongoing research field, which leads to achieving consistency between design and analysis models.

However, this is not the cure-all remedy, which alleviates analysts from identifying and analysing failure behaviours, but helps obtaining a manageable analysis compared to the difficult and laborious traditional process. User friendly resources, such as design components or failure annotation libraries, enable the reuse of nominal and failure models.

Dependable design strategies have been characterized grouping them with respect to their underlying recovery strategy. Reuse of the existing hardware components is a promising solution to design dependable systems at the expense of reducing hardware costs. In this way, we have characterized the existing approaches addressing this design concept. Moreover, we have identified some of the activities which will help extending their use. Mainly, we have concentrated on the systematic identification of reusable resources and overcoming the assumptions of existing approaches, i.e., perfect fault detection, reconfiguration and communication.

For verification purposes, firstly Fault Injection (FI) approaches have been studied. Since the adoption of purely verification oriented FI approaches may incur complexities and difficulties in the system design process, then we have addressed model-based integrative verification approaches. Their main objective is to address consistently dependability analysis, design and verification tasks at the preliminary design phase. An early integration of these tasks would add value to the dependable design process. There are many challenging tasks to address when constructing an end-to-end dependable design methodology. Integration of the CFP approaches within this methodology or validation of the correctness of the faults to be injected are some of the subjects to be addressed.

Finally, we have outlined a model-based hybrid design process integrating addressed design, analysis and verifi-

cation approaches. The main purpose of this methodology is to sketch an intuitive model-based dependable design process attaining consistency and reuse among different models. The integration of the approaches should allow undertaking timely design decisions by reducing costs and manual failure-prone annotations. Additionally, it will alleviate the need to clutter a model with redundant information. Nevertheless, note that when designing a new system, special care should be taken, since reuse properties depend on the system context. The reuse of failure annotations in the design process, eases the architectural iterative refinement process. This makes possible the analysis of different implementations using the same component failure models.

Therefore, we foresee that instead of developing independent approaches to identify, analyse and verify dependability requirements, future directions will focus on integrating different approaches. This process requires tracing verification results to the initial dependable design model and vice versa. In this field, challenging work remains to do sharing information between existing approaches so as to take advantage of complementary strengths of different approaches.

Our current work focuses on the (re-)design of dependable systems by means of exploiting the benefits of heterogeneous redundancies, which may exist in some systems, e.g., trains or buildings. Our research challenge concentrates on the integration of dependability design and analysis activities, systematizing all the design steps, and overcoming assumptions adopted by other approaches for the system's operation and recovery process.

REFERENCES

[1] J. I. Aizpurua and E. Muxika, "Design of Dependable Systems: An Overview of Analysis and Verification Approaches," in *Proceedings of DEPEND 2012*, 2012, pp. 4–12.

[2] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secur. Comput.*, vol. 1, pp. 11–33, January 2004.

[3] M. Rausand and A. Høyland, *System Reliability Theory: Models, Statistical Methods and Applications Second Edition*. Wiley-Interscience, 2003.

[4] A. Arora and S. Kulkarni, "Component based design of multitolerant systems," *IEEE Trans. on Sw. Eng.*, vol. 24, no. 1, pp. 63–78, 1998.

[5] M. Hiller, A. Jhumka, and N. Suri, "An approach for analysing the propagation of data errors in software," in *Proc. of DSN'01*, 2001, pp. 161–170.

[6] W. Vesely, J. Dugan, J. Fragola, Minarick, and J. Railsback, "Fault Tree Handbook with Aerospace Applications," NASA, Handbook, 2002.

[7] US Department of Defense, *Procedures for Performing, a Failure Mode, Effects, and Criticality Analysis (MIL-STD-1629A)*. Whasington, DC, 1980.

[8] M. A. Marsan, "Advances in petri nets 1989." Springer-Verlag, 1990, ch. Stochastic Petri nets: an elementary introduction, pp. 1–29.

[9] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT Press, 2008.

[10] Y. Prokhorova, L. Laibinis, E. Troubitsyna, K. Varpaaniemi, and T. Latvala, "Derivation and formal verification of a mode logic for layered control systems," in *Proc. of APSEC'11*, 2011, pp. 49–56.

[11] P. Fenelon, J. McDermid, and M. Nicolson, "Towards integrated safety analysis and design," *ACM SIGAPP Applied*, 1994.

[12] J. Dugan, S. Bavuso, and M. Boyd, "Dynamic fault-tree models for fault-tolerant computer systems," *IEEE Trans. on Reliability*, vol. 41, no. 3, pp. 363–377, 1992.

[13] B. Kaiser, P. Liggesmeyer, and O. Mäckel, "A new component concept for fault trees," in *Proc. of SCS'03*, 2003, pp. 37–46.

[14] A. Galloway, J. McDermid, J. Murdoch, and D. Pumfrey, "Automation of system safety analysis: Possibilities and pitfalls," in *Proc. of ISSC'02*, 2002.

[15] C. Price and N. Taylor, "Automated multiple failure FMEA," *Reliability Eng. & System Safety*, vol. 76, pp. 1–10, 2002.

[16] M. Ĉepin and B. Mavko, "A dynamic fault tree," *Reliability Eng. & System Safety*, vol. 75, no. 1, pp. 83–91, 2002.

[17] Rao, K. Durga, V. Gopika, V. V. S. Sanyasi Rao, H. S. Kushwaha, A. K. Verma, and A. Srividya, "Dynamic fault tree analysis using Monte Carlo simulation in probabilistic safety assessment," *Reliability Eng. and System Safety*, vol. 94, no. 4, pp. 872–883, 2009.

[18] G. Manno, F. Chiacchio, L. Compagno, D. D'Urso, and N. Trapani, "MatCarloRe: An integrated FT and Monte Carlo Simulink tool for the reliability assessment of dynamic fault tree," *Expert Systems with Applications*, vol. 39, no. 12, pp. 10 334–10 342, 2012.

[19] "MathWorks," http://www.mathworks.com; Last access: 15/06/2013.

[20] S. Distefano and A. Puliafito, "Dynamic reliability block diagrams vs dynamic fault trees," *In Proc. of RAMS'07*, vol. 8, pp. 71–76, 2007.

[21] R. Robidoux, H. Xu, L. Xing, and M. Zhou, "Automated modeling of dynamic reliability block diagrams using colored petri nets," *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, vol. 40, no. 2, pp. 337–351, 2010.

[22] J.-P. Signoret, Y. Dutuit, P.-J. Cacheux, C. Folleau, S. Collas, and P. Thomas, "Make your petri nets understandable: Reliability block diagrams driven petri nets," *Reliability Engineering & System Safety*, vol. 113, pp. 61 – 75, 2013.

[23] "OpenSESAME: the simple but extensive, structured availability modeling environment," *Reliability Engineering & System Safety*, vol. 93, no. 6, pp. 857 – 873, 2008.

[24] I. Lopatkin, A. Iliasov, A. Romanovsky, Y. Prokhorova, and E. Troubitsyna, "Patterns for representing FMEA in formal specification of control systems," in *Proc. HASE'11*, 2011, pp. 146–151.

[25] "Event-B and the Rodin platform," http://www.event-b.org; Last access: 15/06/2013.

[26] M. Bouissou, "A generalization of Dynamic Fault Trees through Boolean logic Driven Markov Processes (BDMP)," in *Proc. of ESREL'07*, vol. 2, 2007, pp. 1051–1058.

[27] B. Kaiser, C. Gramlich, and M. Forster, "State-Event Fault Trees - A Safety Analysis Model for Software-Controlled Systems," *Reliability Eng. System Safety*, vol. 92, no. 11, pp. 1521–1537, 2007.

[28] D. Harel, "Statecharts: A visual formalism for complex systems," 1987.

[29] M. Steiner, P. Keller, and P. Liggesmeyer, "Modeling the effects of software on safety and reliability in complex embedded systems," in *Computer Safety, Reliability, and Security*. Springer Berlin Heidelberg, 2012, vol. 7613, pp. 454–465.

[30] TU Kaiserslautern, AG Seda and Fraunhofer IESE, "Embedded system safety and reliability analyzer (ESSaRel)," 2009, http://essarel.de; Last access: 15/06/2013.

[31] TU Berlin, Real-Time Systems and Robotics group, "TimeNET 4.0," 2007, http://tu-ilmenau.de/TimeNET; Last access: 15/06/2013.

[32] O. el Ariss, D. Xu, and W. E. Wong, "Integrating safety analysis with functional modeling," *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, vol. 41, no. 4, pp. 610–624, 2011.

[33] University of Virginia, "Galileo," 2003, http://http://www.cs.virginia.edu/~ftree/; Last access: 15/06/2013.

[34] Ferdinando Chiacchio, "RAATS Tool," 2012, http://www.dmi.unict.it/~chiacchio/?m=5&project=raatss; Last access: 15/06/2013.

[35] GRIF Workshop, "BStoK Module," http://grif-workshop.com/grif/bstok-module/; Last access: 15/06/2013.

[36] Max Walter, "OpenSESAME - Simple but Extensive Structured Availability Modeling Environment," 2009, http://www.lrr.in.tum.de/~walterm/opensesame/; Last access: 15/06/2013.

[37] EDF, "KB3 Workbench," 2012, http://research.edf.com/research-and-the-scientific-community/software/kb3-44337.html; Last access: 15/06/2013.

[38] P. Fenelon and J. A. McDermid, "An integrated tool set for software safety analysis," *J. Syst. Softw.*, vol. 21, pp. 279–290, 1993.

[39] Y. Papadopoulos, M. Walker, D. Parker, E. Rüde, R. Hamann, A. Uhlig, U. Grätz, and R. Lien, "Engineering failure analysis and design optimisation with HiP-HOPS," *Engineering Failure Analysis*, vol. 18, no. 2, pp. 590–608, 2011.

[40] D. Domis and M. Trapp, "Component-based abstraction in fault tree analysis," in *Computer Safety, Reliability, and Security*, ser. LNI. Springer, 2009, vol. 5775, pp. 297–310.

[41] R. Niu, T. Tang, O. Lisagor, and J. McDermid, "Automatic safety analysis of networked control system based on failure propagation model," in *Proc. of ICVES'11*, 2011, pp. 53–58.

[42] M. Walker and Y. Papadopoulos, "Qualitative temporal analysis: Towards a full implementation of the fault tree handbook," *Control Eng. Practice*, vol. 17, no. 10, pp. 1115–1125, 2009.

[43] R. Paige, L. Rose, X. Ge, D. Kolovos, and P. Brooke, "FPTC: Automated safety analysis for Domain-Specific languages," in *MoDELS Workshops '08*, vol. 5421, 2008, pp. 229–242.

[44] I. Wolforth, M. Walker, L. Grunske, and Y. Papadopoulos, "Generalizable safety annotations for specification of failure patterns," *Softw. Pract. Exper.*, vol. 40, pp. 453–483, 2010.

[45] C. Priesterjahn, C. Sondermann-Wölke, M. Tichy, and C. Hölscher, "Component-based hazard analysis for mechatronic systems," in *Proc. of ISORCW'11*, 2011, pp. 80–87.

[46] P. Struss and A. Fraracci, "Automated model-based fmea of a braking system," in *IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes*, vol. 8, 2012, pp. 373–378.

[47] OCC'M Software GmbH, "Raz'r Model Editor Ver. 3," http://www.occm.de; Last access: 15/06/2013.

[48] R. F. Paige, L. M. Rose, X. Ge, D. S. Kolovos, and P. J. Brooke, "Fptc: Automated safety analysis for domain-specific languages." in *MoDELS Workshops*, ser. Lecture Notes in Computer Science, M. R. V. Chaudron, Ed., vol. 5421. Springer, 2008, pp. 229–242.

[49] University of Paderborn, "FUJABA Tool Suite," 2012, http://www.fujaba.de/no_cache/home.html; Last access: 15/06/2013.

[50] P. Feiler and A. Rugina, "Dependability Modeling with the Architecture Analysis & Design Language (AADL)," Technical Note CMU/SEI-2007-TN-043, CMU Software Engineering Institute, 2007.

[51] "The Unified Modeling Language," http://www.uml.org/; Last access: 15/06/2013.

[52] OMG, "MDA Guide Version 1.0.1," http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf, 2003. [Online]. Available: http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf

[53] L. Fuentes-Fernández and A. Vallecillo-Moreno, "An Introduction to UML Profiles," *Journal of UML and Model Engineering*, vol. 5, no. 2, pp. 5–13, 2004.

[54] S. Bernardi, J. Merseguer, and D. Petriu, "Dependability modeling and analysis of software systems specified with UML," *ACM Computing Survey*, In Press.

[55] L. Montecchi, P. Lollini, and A. Bondavalli, "An intermediate dependability model for state-based dependability analysis," University of Florence, Dip. Sistemi Informatica, RCL group, Tech. Rep., 2011.

[56] B. Gallina, M. A. Javed, F. U. Muram, and S. Punnekkat, "Model-driven dependability analysis method for component-based architectures," in *Euromicro-SEAA Conference*. IEEE Computer Society, 2012.

[57] B. Gallina and S. Punnekkat, "FI^{4}FA: A Formalism for Incompletion, Inconsistency, Interference and Impermanence Failures Analysis," in *Proc. of EUROMICRO*, ser. SEAA '11. IEEE Computer Society, 2011, pp. 493–500.

[58] R. Adler, D. Domis, K. Höfig, S. Kemmann, T. Kuhn, J. Schwinn, and M. Trapp, "Integration of component fault trees into the UML," in *MoDELS'10*, 2010, pp. 312–327.

[59] M. Biehl, C. DeJiu, and M. Törngren, "Integrating safety analysis into the model-based development toolchain of automotive embedded systems," in *Proc. of LCTES '10*. ACM, 2010, pp. 125–132.

[60] A. Rugina, K. Kanoun, and M. Kaâniche, "A system dependability modeling framework using AADL and GSPNs," in *Architecting dependable systems IV, LNCS*, vol. 4615. Springer, 2007, pp. 14–38.

[61] A. Joshi, S. Vestal, and P. Binns, "Automatic Generation of Static Fault Trees from AADL models," in *DNS Workshop on Architecting Dependable Systems*. Springer, 2007.

[62] A. Arnold, G. Point, A. Griffault, and A. Rauzy, "The AltaRica formalism for describing concurrent systems," *Fundamenta Informaticae*, vol. 40, no. 2-3, pp. 109–124, 1999.

[63] B. Romain, J.-J. Aubert, P. Bieber, C. Merlini, and S. Metge, "Experiments in model based safety analysis: Flight controls," in *DCDS'07*, 2007, pp. 43–48.

[64] P. Bieber, C. Castel, and C. Seguin, "Combination of fault tree analysis and model checking for safety assessment of complex system," in *Proc. of EDCC'02*, vol. 2485. Springer, 2002, pp. 624–628.

[65] K. Mokos, P. Katsaros, N. Bassiliades, V. Vassiliadis, and M. Perrotin, "Towards compositional safety analysis via semantic representation of component failure behaviour," in *Proc. of JCKBSE'08*. IOS Press, 2008, pp. 405–414.

[66] M. Riedl and M. Siegle, "A LAnguage for REconfigurable dependable Systems: Semantics & Dependability Model Transformation," in *Proc. 6th International Workshop on Verification and Evaluation of Computer and Communication Systems (VECOS'12)*. British Computer Society, 2012, pp. 78–89.

[67] M. Riedl, J. Schuster, and M. Siegle, "Recent Extensions to the Stochastic Process Algebra Tool CASPA," in *Proceedings of the 2008 Fifth International Conference on Quantitative Evaluation of Systems*, ser. QEST '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 113–114.

[68] R. Cressent, V. Idasiak, F. Kratz, and P. David, "Mastering safety and reliability in a Model Based process," in *Proc. of RAMS'11*, 2011.

[69] "OMG Systems Modelling Language," http://www.omgsysml.org/; Last access: 15/06/2013.

[70] Eclipse Foundation, "Eclipse Papyrus," 2012, http://www.eclipse.org/papyrus/; Last access: 15/06/2013.

[71] Labri, "Altarica," http://www.altarica.labri.fr/; Last access: 15/06/2013.

[72] Carnegie Melon University, "OSATE," 2012, https://wiki.sei.cmu.edu/aadl/index.php/AADL_tools; Last access: 15/06/2013.

[73] CHESS partners, "CHESS Project," 2012, http://www.chess-project.org/; Last access: 15/06/2013.

[74] "ATESST2 Homepage," 2010, http://www.east-adl.fr; Last access: 15/06/2013.

[75] O. Lisagor, "Failure logic modelling: A pragmatic approach," Ph.D. dissertation, Department of Computer Science, The University of York, 2010.

[76] I. Lopatkin, A. Iliasov, and A. Romanovsky, "Rigorous development of dependable systems using fault tolerance views," in *Proceedings of the 2011 IEEE 22nd International Symposium on Software Reliability Engineering*, ser. ISSRE '11. IEEE Computer Society, 2011, pp. 180–189.

[77] C. P. Shelton and P. Koopman, "Improving system dependability with functional alternatives," in *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, ser. Proceedings of DSN '04. IEEE Computer Society, 2004, pp. 295–304.

[78] L. Cauffriez, D. Renaux, T. Bonte, and E. Cocquebert, "Systemic modeling of integrated systems for decision making early on in the design process." *Cybernetics and Systems*, vol. 44, pp. 1–22, 2013.

[79] J. Clarhaut, S. Hayat, B. Conrard, and V. Cocquempot, "Optimal design of dependable control system architectures using temporal sequences of failures," *Ieee Transactions On Reliability*, vol. 58, no. 3, pp. 511–522, 2009.

[80] L. Cauffriez, J. Ciccotelli, B. Conrard, and M. Bayart, "Design of intelligent distributed control systems: a dependability point of view," *Reliability Engineering & System Safety*, vol. 84, no. 1, pp. 19–32, 2004.

[81] M. Adachi, Y. Papadopoulos, S. Sharvia, D. Parker, and T. Tohdo, "An approach to optimization of fault tolerant architectures using hip-hops," *Softw. Pract. Exp.*, 2011.

[82] J. Wysocki, R. Debouk, and K. Nouri, "Shared redundancy as a means of producing reliable mission critical systems," in *Proc. of RAMS'04*, 2004, pp. 376 – 381.

[83] J. Galdun, J. Ligus, J.-M. Thiriet, and J. Sarnovsky, "Reliability increasing through networked cascade control structure - consideration of quasi-redundant subsystems," in *IFAC Proc. Volumes*, vol. 17, 2008, pp. 6839–6844.

[84] O. Rawashdeh and J. Lumpp Jr., "Run-time behavior of ardea: A dynamically reconfigurable distributed embedded control architecture," in *IEEE Aerospace Conference Proceedings*, 2006.

[85] M. Trapp, R. Adler, M. Förster, and J. Junger, "Runtime adaptation in safety-critical automotive systems," in *Proc. of International Conference on Software Engineering*, 2007.

[86] R. Adler, I. Schaefer, M. Trapp, and A. Poetzsch-Heffter, "Component-based modeling and verification of dynamic adaptation in safety-critical embedded systems," *ACM Trans. Embed. Comput. Syst.*, vol. 10, no. 2, pp. 20:1–20:39, Dec. 2010.

[87] R. Adler, D. J. Domis, M. Förster, and M. Trapp, "Probabilistic analysis of safety-critical adaptive systems with temporal dependences," in *Proc. of RAMS'08*. IEEE Computer Society, 2008, pp. 149–154.

[88] R. Adler, D. Schneider, and M. Trapp, "Engineering dynamic adaptation for achieving cost-efficient resilience in software-intensive embedded systems," in *Proc. of Engineering of Complex Computer Systems*. IEEE Computer Society, 2010, pp. 21–30.

[89] J. Moore, *The Avionics Handbook*. CRC Press, 2001, ch. Advanced Distributed Architectures.

[90] P. Bieber, E. Noulard, C. Pagetti, T. Planche, and F. Vialard, "Preliminary design of future reconfigurable IMA platforms," *SIGBED Rev.*, vol. 6, no. 3, 2009.

[91] P. Bieber, J. Brunel, E. Noulard, C. Pagetti, T. Planche, and F. Vialard, "Preliminary Design of Future Reconfigurable IMA Platforms - Safety Assessment," in *27th International Congress of the Aeronautical Sciences*, 2010.

[92] C. Engel, A. Roth, P. H. Schmitt, R. Coutinho, and T. Schoofs, "Enhanced dispatchability of aircrafts using multi-static configurations," in *Proc. of ERTS'10*, 2010.

[93] M. Förster and D. Schneider, "Flexible, any-time fault tree analysis with component logic models," in *ISSRE*. IEEE Computer Society, 2010, pp. 51–60.

[94] J. I. Aizpurua and E. Muxika, "Dependable Design: Trade-Off Between the Homogeneity and Heterogeneity of Functions and Resources," in *Proceedings of DEPEND 2012*, 2012, pp. 13–17.

[95] C. Priesterjahn, D. Steenken, and M. Tichy, "Component-based timed hazard analysis of self-healing systems," in *Proceedings of the 8th workshop on Assurances for self-adaptive systems*, ser. ASAS '11. ACM, 2011, pp. 34–43.

[96] M. Bozzano and A. Villafiorita, "The FSAP/NuSMV-SA Safety Analysis Platform," *Int. J. Softw. Tools Technol. Transf.*, vol. 9, pp. 5–24, February 2007.

[97] H. Aljazzar, M. Fischer, L. Grunske, M. Kuntz, F. Leitner-Fischer, and S. Leue, "Safety analysis of an airbag system using probabilistic fmea and probabilistic counterexamples," in *QEST'09*. IEEE Computer Society, 2009, pp. 299–308.

[98] L. Grunske, K. Winter, N. Yatapanage, S. Zafar, and P. A. Lindsay, "Experience with fault injection experiments for fmea," *Software: Practice and Experience*, 2011.

[99] A. Gomes, A. Mota, A. Sampaio, F. Ferri, and E. Watanabe, "Constructive model-based analysis for safety assessment," *International Journal on Software Tools for Technology Transfer*, vol. 14, pp. 673–702, 2012.

[100] X. Ge, R. Paige, and J. McDermid, "Probabilistic failure propagation and transformation analysis," in *SAFECOMP'09*, 2009, vol. 5775, pp. 215–228.

[101] M. Z. Kwiatkowska, G. Norman, and D. Parker, "Prism 4.0: Verification of probabilistic real-time systems." in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 585–591.

[102] FBK, "The FSAP/NuSMV-SA Web Page," https://es.fbk.eu/tools/FSAP/; Last access: 15/06/2013.

[103] "Behaviour Tree Editor (BTE)," http://www.sqi.griffith.edu.au/gse/tools/overview.html; Last access: 15/06/2013.

[104] "Symbolic Analysis Laboratory (SAL)," http://sal.csl.sri.com; Last access: 15/06/2013.

[105] Alexandre Mota, "Simulink to prism," http://www.cin.ufpe.br/~acm/simulinktoprism/; Last access: 15/06/2013.

[106] "Correctness, Modelling and Performance of Aerospace Systems," http://compass.informatik.rwth-aachen.de; Last access: 15/06/2013.

[107] M. Bozzano, A. Cimatti, J.-P. Katoen, V. Nguyen, T. Noll, and M. Roveri, "Safety, dependability and performance analysis of extended AADL models," *Computer Journal*, vol. 54, no. 5, pp. 754–775, 2011.

[108] M. Güdemann and F. Ortmeier, "Towards model-driven safety analysis," in *Proc. of DCDS 11*, 2011, pp. 53 – 58.

[109] M. Güdemann, F. Ortmeier, and W. Reif, "Using Deductive Cause-Consequence Analysis (DCCA) with SCADE," in *Proc. of SAFECOMP'07*, vol. 4680, 2007, pp. 465–478.

[110] "The Open-Source Toolkit for Critical Systems," http://www.topcased.org; Last access: 15/06/2013.

[111] B. Berthomieu, J.-P. Bodeveix, P. Farail, M. Filali, H. Garavel, P. Gaufillet, F. Lang, and F. Vernadat, "Fiacre: an intermediate language for model verification in the topcased environment," in *ERTS'08*, 2008.

[112] B. Berthomieu, P.-O. Ribet, and F. Vernadat, "The tool TINA - Construction of abstract state spaces for petri nets and time petri nets," *International Journal of Production Research*, vol. 42, no. 14, pp. 2741–2756, 2004.

[113] J.-C. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, and M. Sighireanu, "CADP - A Protocol Validation and Verification Toolbox," 1996.

[114] COMPASS Consortium, "Compass toolset," http://compass.informatik.rwth-aachen.de/download.html; Last access: 15/06/2013.

[115] M. Lipaczewski, S. Struck, and F. Ortmeier, "Using Tool-Supported Model Based Safety Analysis - Progress and Experiences in SAML Development," in *IEEE 14th International Symposium on High-Assurance Systems Engineering (HASE 2012)*, 2012.

[116] A. Gomes, A. Mota, A. Sampaio, F. Ferri, and J. Buzzi, "Systematic model-based safety assessment via probabilistic model checking," in *ISoLA'10*. Springer, 2010, pp. 625–639.

[117] R. Gmehlich, K. Grau, M. Jackson, C. Jones, F. Loesch, and M. Mazzara, "Towards a formalism-based toolkit for automotive applications," Computing Science, Newcastle University, Tech. Rep., 2012.