System-Software Co-Engineering: Dependability and Safety Perspective

Y. Yushtein*, M. Bozzano†, A. Cimatti†, J.-P. Katoen‡, V.Y. Nguyen‡, Th. Noll‡, X. Olive§, M. Roveri†

*Systems, Software & Technology Department, European Space Agency, Noordwijk, NL, yuri.yushtein@esa.int
†Embedded Systems Research Unit, Fondazione Bruno Kessler, Trento, I, {bozzano,cimatti,roveri}@fbk.eu
‡Software Modeling and Verification Group, RWTH Aachen University, D, {katoen,nguyen,noll}@cs.rwth-aachen.de
§Research Department, Thales Alenia Space France, Cannes la Bocca, F, xavier.olive@thalesaleniaspace.com

Abstract—The need for an integrated system-software co-engineering framework to support the design of modern space systems is pressing. The current tools and formalisms tend to be tailored to specific analysis techniques and are not amenable for the full spectrum of required system aspects such as safety, dependability and performability. Additionally, they cannot handle the intertwining of hardware and software interaction. As such, the current practice lack integration and coherence. We recently developed a coherent and multidisciplinary approach towards developing space systems at architectural design level, linking all of the aforementioned aspects, and assessed it with several industrial evaluations. This paper reports on the approach, the evaluations and our perspective on current and future developments.

Keywords—correctness, dependability, fault tree analysis, model checking, performability, safety

I. INTRODUCTION

The engineering process for on-board critical embedded systems employs a wide range of diverse approaches and techniques to verify the operational correctness of a system, to ensure its dependability and safety level, and to guarantee the required system performance. Evaluating these system-level aspects is a highly challenging task. Inherently heterogeneous subsystems and their constituents (e.g., avionics, software, power distribution, electrically controlled actuators) with the specific development approaches employ different specification formalisms and corresponding analysis techniques. The design data necessary for verifying and validating various system-level aspects come from different engineering disciplines and different sub-system development lifecycles. This impairs effective complementary use of these analyses to the system-level extent.

To guarantee completeness and consistency across the results coming from the heterogeneous specification and analysis techniques, and to ensure effective and consistent complementary use of the diverse verification and analysis approaches, an integrated System-Software Co-Engineering approach is required. This approach can be defined through (1) a general-purpose modeling and specification formalism, accompanied by (2) the corresponding formal analysis techniques for all properties and measures of interest, and (3) a set of powerful tools implementing these techniques.

II. STATE-OF-THE-PRACTICE CHALLENGES

Achieving mission objectives and ultimate mission success relies on dependability and safety of the space systems. As software plays a more and more prominent role in space systems, its contribution to the overall system dependability becomes a vital aspect of system development. The ECSS-Q80 standard [1] identifies methods to support the assessment of software dependability and safety. These include Software Failure Modes, Effects and Criticality Analysis (FMECA), Software Fault Tree Analysis (FTA), Hardware-Software Interaction Analysis (HSIA), Software Hazard Analysis, and Software Common Cause Failure Analysis. HSIA is of particular interest, as it targets the software behavior in relation to the hardware failure modes, and the corresponding software compensatory provisions, implying the provision of inputs to the Fault Detection, Isolation and Recovery (FDIR) requirements [2].

Current development practices, however, provide mostly ad-hoc solutions to fulfill these requirements. They suffer from discontinuity between the software and system (hardware) Reliability, Availability, Maintainability and Safety (RAMS) activities, as representative system/hardware data (e.g., system FMEA/FMECA, FTA) only become available at a late stage in the design process when software development is far under way. Consequently, only the most general system-level dependability requirements are initially taken into account in the software development process. This leads to multiple assumptions about the hardware, the neglect of the probabilistic nature of hardware failures and degraded modes of operation. Likewise, during hardware development, a strong assumption on complete software correctness is taken without regard for software failures. Moreover, FTA approaches do neither take fault timing aspects into consideration, nor do they reflect dynamic behaviors and possible recovery actions. FMEAs do not establish formal mappings to design models and do not generally cover fault modes’ (timed) dependencies. On top of that, these safety and dependability models are usually created separately from the design models. This raises issues of semantic correspondence between these sets of models and those corresponding to software and hardware. It complicates the use of safety and dependability analysis results for design
improvements and makes tracing fault propagation through design models a challenging and error-prone task. As a result, current practices cannot assure completeness of the analyzed effects and as a consequence undermine the total analysis confidence level.

This applies in particular to the software and in particular to the FDIR specification and development, as it relies on the very concrete system-level input. These data become available only in the later development stages, leading to late initiation of the FDIR development, having a detrimental effect on the eventual FDIR maturity. All possible fault and failure combinations are inherently complex to analyze and it is hard to define an adequate software behavior and FDIR strategy to handle these. Changes in the sub-systems design, their fault modes, and possible fault combinations and scenarios, hamper the process of achieving a stable FDIR design. As a result the FDIR design, on the one hand, is approached in an ad-hoc manner to leverage the available and stable system and sub-system data, whereas on the other hand it experiences schedule pressures to fit the software development process and system-level testing. The latter is a challenging process from the dependability and safety perspective due to the physical limitations to reproduce all the possible faults and their combinations during the test campaigns.

Summarizing, current approaches to address the dependability and safety matters lack coherence between the system and software perspectives in terms of specification capabilities, analysis facilities availability (techniques, technologies, and tool support), and corresponding processes.

III. THE COMPASS APPROACH

European Space Agency set up the COMPASS project (COrrectness, Modeling, and Performance of AeroSpace Systems; [3]) for tackling the challenges in Section II. Its goal was to develop a coherent and multi-disciplinary approach towards developing systems at architectural design (i.e., systems engineering) level, linking the aspects of hardware and software co-engineering, performability, and RAMS. By employing state of the art model checkers, rigorous means were available to analyze all possible effects and ensure completeness. The project was coordinated by RWTH Aachen University (RWTH) as prime contractor with Fondazione Bruno Kessler (FBK) and Thales Alenia Space (TAS) as subcontractors. It started in February 2008 and ended in June 2010.

A. System Specification

An architectural specification language has been designed that offers a convenient way to describe hardware and software operation, hybridity, (probabilistic) faults and their propagation, error recovery, and degraded modes of operation. A system specification is hierarchically organized into components which interact through connections via ports allowing for both message (event) and continuous (data) communication, and which can be reconfigured dynamically. The specification formalism is based on the Architecture Analysis and Design Language AADL [4] and its Error Model Annex [5]. It is named System-Level Integrated Modeling (SLIM) language [6].

A formal semantics of the SLIM language [7] was developed that precisely characterizes the complete set of nominal and error behaviors of a given system model. It opens up the possibility to apply a wealth of formal methods for various kinds of verification and validation (V&V) activities. The SLIM language can be considered as an extended subset of AADL. It exhibits the following design features:

- **Modeling the system’s nominal behavior.** To this aim, our input language allows to specify both hardware and software operation, specified at the level of processors, storages, buses, sensors, actuators, processes and threads.
- **Specifying timed and hybrid behavior.** In particular, in order to model continuous physical systems, the language supports continuous real-valued variables with (linear) time-dependent dynamics.
- **Modeling the system’s faulty behavior.** This is supported by primitives to describe software and hardware faults, error propagation (that is, turning fault occurrences into failure events), sporadic (transient) and permanent faults, and degraded modes of operation (by mapping failures from architectural to service level). It also includes probabilistic aspects such as random faults, repairs, and their stochastic timing.
- **Specifying (partial) observability and the associated observability requirements.** These notions are essential to deal with diagnosability and FDIR analyses.

We use temporal property patterns to describe both the qualitative and the quantitative properties that the system under analysis has to satisfy. Those patterns act as parameterized templates to the engineers and thus offer a comprehensible and easy-to-use framework for requirements specification. In particular, their usage does not require any specific knowledge about temporal logics underneath.

B. Analysis Facilities

The V&V activities are supported by an integrated toolset that implements the following functionalities:

- **Requirements Validation:** In order to ensure the quality of requirements, they can be validated independently of the system. It checks whether requirements do no exclude each other, whether properties are logically compatible and whether an assertion is a logical consequence of the requirements.
- **Functional Verification:** Analyzing operational correctness is the first step to be performed during the system development life-cycle. It consists in verifying that the system will behave correctly with respect to a set of
functional requirements, under the hypothesis of nominal conditions, that is, when software and hardware components are assumed to be fault-free.

Safety and Dependability Analysis: [8], [9], [10], [11]: Analyzing system safety and dependability is a fundamental step that is performed in parallel with system design and verification of functional correctness. The goal is to investigate the behavior of a system in degraded conditions (that is, when some parts of the system are not working properly, due to malfunctioning) and to ensure that the system meets the safety requirements that are mandatory for its deployment and use. Key techniques in this area are (dynamic) FTA, (dynamic) FMEA, fault tolerance evaluation, and criticality analysis.

Performability Analysis: [9], [12]: To guarantee the required system performance in the presence of faults, integrated hardware and software models can be evaluated with respect to their performance behavior in degraded modes of operation.

FDIR Analysis: [2]: System models can include a formal description of both the fault detection and isolation sub-systems, and the recovery actions to be taken. Based on these models, tool facilities are provided to analyze the operational effectiveness of the FDIR measures, and to assess whether the observability of system parameters is sufficient to make failure situations diagnosable.

The integrated toolset is fully graphical-driven and works under Linux. Screenshots of functional verification, fault tree analysis, and performability are shown in Figures 1, 2, and 3.

C. Performance and Scalability

The time and memory resources needed by the analyses implemented in the COMPASS toolset directly correspond to the model’s characteristics. More components, more interactions and increasing detail affect the scalability of the approach. A precise definition of contributing factors in this context is still an open theoretical question. Nevertheless, to get a current perspective on the performance we crafted a synthetic benchmark based on a redundant sensor-filter system. The degree of redundancy is a measure of increasing complexity. As shown Figure 4, the verification time increases exponentially with respect to this parameter. A majority of the time is spent in the model checker NuSMV. The probabilistic computation (by MRMC) is negligible, as symbolic model representations are reduced significantly before performability analysis by MRMC.

This exponential increase can be curbed by careful modeling and accounting for a suitable level of abstraction. This however is a user effort. We believe that the user should be oblivious to this and instead focus on crafting a complete and correct model. For this reason, we are driving our current efforts towards automated abstractions and more efficient model checking procedures. Our initial version of a slicing technique (in Section V) is an exemplary automated abstraction technique and led to promising results. In parallel, we
are investigating the use of compositional reasoning during model checking (in Section VI) and combine this to exploit the increasing availability of multi-core systems.

IV. INDUSTRIAL EVALUATION

TAS evaluated the COMPASS toolset on several industrially representative cases. The evaluation covers the main functionalities of COMPASS: correctness (simulation, property checking, and deadlock checking), safety analyses (fault tree and FMEA generation) and FDIR analyses (fault detection, fault isolation, and fault recovery). A case study that is concerned with thermal regulation of satellites is presented in detail below. The other case studies have been performed to check and analyse the behaviour of an FDIR strategy. The most complex model represents an abstraction of a complete satellite platform. It is composed of five AOCS (Altitude and Orbit Control Subsystem) components, the AOCS mode management (six modes) and equipment allocation per mode, two abstracted functional chains (power and thermal), a simple representation of the core avionics (modeling only parts of the involved boards), and the on-board procedures (i.e., a sequence of telecommands with delay) to be verified, and the associated mechanism based on PUS (Packet Utilization Standard) to execute them on-board. Timing aspects have not been addressed and are subject to current investigations.

A. Thermal Regulation Case Study

The thermal regulation case study has been investigated at two different levels of abstraction: the functional level (cf. Figure 5) and the physical level (cf. Figure 6). The functional model consists of five functions: (1) acquisition based on the temperature sensors, (2) control and monitoring based on a software part, (3) safety and command switches based on an hardware part, (4) passive thermal units (such as OSR, MLI, shields, . . . ), and (5) heating based on heater lines. Overall, 12 units (modeled as single SLIM model) are used by these five functions, hence 12 instances of this generic model are present in the complete model. The reduction is possible due to the abstraction to the functional level, where the behaviour of units is not represented. There is no need to incorporate such details.

From the function decomposition and the allocation of units by function, the SLIM model has then been derived to obtain the model at the physical level. The information flows are identical for both models. The hierarchical structure of the SLIM language allowed us to tackle the case by the model refinement. The acquisition function is mapped onto the three temperature sensors (T1 to T3); the heating function onto the heaters (H1 and H2); the control and monitoring function onto the Satellite Management Unit (SMU); and safety and command switches to a dedicated board in the Platform Distribution and Interface Unit (PFDIU). The passive unit function is not represented, as the involved components have no behaviour, and thus the functional level is sufficient to perform the verification of this function.

At functional level, two kinds of faults are anticipated: a function can become inoperational, or a unit can become inoperational. Each kind of fault is instantiated for each function and each unit. At the physical level, six different kinds of faults have been considered: (a) temperature sensor: stuck at minimum value, stuck at maximum value, (b) heater: never heating, always heating, (c) switch: stuck open, stuck closed. The model used for performing the analysis contains 18 injected faults (i.e., instances of the previously defined kind).

B. Analysis Results

For all the SLIM models, the absence of deadlocks has been checked. They are all deadlock-free. This check is mandatory to ensure the correctness of the other analysis results. Most of the models have been extensively simulated to compare their behaviour with the expected one. As formal modeling languages are often hard to manage and comprehend by system engineers, simulation can be considered as a kind of model debugging and a first sanity check. It
is recommended before starting further formal verification steps, in particular when the model is hand-crafted.

Figure 7 shows an example simulation of an extended model where faults are raised. The scenario starts by a fault injection: the nominal heater is always heating. The temperature reaches the maximal threshold and a recovery is started to switch from nominal to redundant heater. The model is then re-initialized by resetting the temperature value. The last action is required as the transitory phase is not modeled.

Safety analyses have been performed on both models. The functional model permits to identify a critical path in the function. Due to the level of abstraction used, the fault tree contains a single point failure (SPF). Moving to the physical level, the model is more precise and the redundancy can be expressed. The SPF is no more present, and the analysis provides the expected results.

FMEA tables have been generated for the thermal regulation, and then individually for each function. Initially, there were many redundant entries. In a short follow-up project, this was resolved using a compactification scheme without excluding useful information (cf. Section V-D).

Also the effectiveness of FDIR has been analysed. For this purpose, the model was extended with observable variables which are used to detect and identify the faults. They are defined in the design of the system. Analysis shows that the set of observables allows one to detect all anticipated faults. The fault identification analysis is performed by selecting all faults to which an observable variable is sensitive, and by checking that the observations are sufficient to uniquely explain the fault reasons.

The last analysis is fault recovery. The thermal regulation model has been defined with three different configurations for the heater recovery:

- no reconfiguration (only H1 is present in this configuration);
- single reconfiguration (H1 nominal and H2 redundant; switch from H2 to H1 is not allowed); and
- double reconfiguration (H1 and H2 can be nominal or redundant; the system can never fail totally).

The table below provides the results of fault recovery analysis. It yields the expected outcome, depending on the configurations.

<table>
<thead>
<tr>
<th>Reconfiguration</th>
<th>H1 failed</th>
<th>H2 failed</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Not recoverable (H2 not available)</td>
<td>Not applicable (only one heater)</td>
</tr>
<tr>
<td>Single</td>
<td>Recoverable</td>
<td>Not recoverable</td>
</tr>
<tr>
<td>Double</td>
<td>Recoverable</td>
<td>Recoverable</td>
</tr>
</tbody>
</table>

C. Assessment

The COMPASS toolset has been evaluated on several case studies and has provided valuable results. The SLIM language is sufficiently expressive to describe (for discrete representation) most of the platform subsystems of a satellite, and their interactions. The automata-paradigm for behavior specification using modes and transitions encourages abstraction, yet allowing a concise and complete behaviour of all the on-board equipments. This is desired, as having an overly detailed model is not effective for the underlying model checkers. The basic functionalities (fault tree analysis, FMEA, correctness checking) can be considered as mature enough to be adopted by the industry and the more novel functions are promising. Further efforts are needed to integrate it in the overall space engineering process and existing in-production tools. Translation from current engineering formalisms to SLIM and from the COMPASS analyses back to a form comprehensible for system engineers. These translation will be the key steps towards an industrial adoption of the COMPASS approach.

V. Current Activities

A. Telecom/Navigation Case Study

We are currently conducting an internal ESA study where we employ the COMPASS toolset on an in-development Telecom/Navigation satellite. The objective is to model and analyze the satellite platform and its FDIR operations. The activities of the study are running in parallel with the development of the satellite itself. This allows us to assess the benefits of formal modeling and analysis in the early design phases, where the mission outline is determined, but the systems architecture and detailed requirements are still under development. The final report of the study is expected in summer 2011. Preliminary results indicate that a SLIM model of the system level provides coherence for many disciplines and can even be used as a source of traceability. The FMEA and fault tree generation facilities in COMPASS speed up the acquisition of these artifacts immensely as in current industrial practice these artifacts are still manually constructed.

As for future guidelines, we noticed that particular attention has to be paid to the level of abstraction. The technical design documents become increasingly detailed during development, and it is prudent to discretize values of variables to ranges, like nominal and non-nominal values. At system level, the precise values are not always of interest. Discretization leads to more efficient models which can be analyzed faster. We also believe that the definition of
architectural design patterns for formal verification is useful in order to exploit model checkers effectively. They capture expert experience and provide novice users direction on initial modeling. It also appears that formal tools should become more transparent in their operations. Verbose user feedback is expected about the progress of the analysis and the intermediate internal computations so that the user can validate the outcome of the analysis. Such transparency adds to the trust and acceptance of the results from formal analyses. A full set of recommendations and directions for future activities will be part of the final report.

B. COMPASS Graphical Modeling Tool

The system models to be analyzed by the COMPASS toolset currently have to be provided in the SLIM textual format, which is based on the syntax of AADL [4] and its Error Model Annex [5]. This requires users to perform text-based system modeling. However, experience shows that the usability and adoption of the toolset would greatly benefit from the possibility of graphical modeling, allowing engineers to specify systems in a graphical notation that eases the understanding of their architecture and behavior. ESA has therefore initiated a follow-up project to COMPASS with the aim to define a graphical notation that covers the full SLIM language as defined in the COMPASS project [7], to develop a front-end tool that allows to edit, load, and store models in the graphical notation and have it integrated in the current COMPASS toolset and evaluated in an industrial context. The definition of the graphical notation is currently under development. It is geared towards the graphical AADL notation as defined in [13], extended by support for transition diagrams that are used in SLIM to specify both the nominal and the faulty behavior of components. This activity is planned to finish in autumn 2011.

C. Contribution to AADL Standardization

On the basis of the experiences that have been gained from the design and use of the SLIM language, some possible improvements of AADL and, in particular, its Error Model Annex have been suggested to the AADL Committee. Two of the proposed modifications, namely observability attributes for diagnosability analysis and internal error events, are currently under discussion and will presumably become part of the upcoming new release of the AADL Error Model Annex.

D. FMEA Reduction

An additional line of work aims at improving the readability of automatically generated FMEA tables. FMEA is an inductive technique that, starting from a set of identified faults (and combinations thereof), assesses their effects on a set of properties. In its basic form, an FMEA table is given by a set of entries relating (combinations of) faults with the properties which are invalidated by their occurrence. Our activity focuses on the removal of the entries that are redundant from an engineering perspective, i.e., those entries that, intuitively speaking, contain faults that do not contribute to the invalidation of the corresponding property. Note that in general, for a monotonic system, given a combination of faults \( F \) invalidating property \( P \), any superset of \( F \) still invalidates \( P \), regardless of whether the additional faults may be possible explanations for the invalidation or not. This causes an unnecessary increase in the number of entries of an FMEA table, cluttering its presentation and complicating its comprehension. As part of this activity, the notion of redundancy was formally defined, on the basis of user expectations. The definition applies to FMEA tables of arbitrary cardinality, and is given by induction on the cardinality of the entries – the general case also takes into account entries such that the corresponding combinations of faults are not disjoint. This approach has been implemented in the COMPASS toolset, and it has been successfully validated and experimentally evaluated by TAS. See the table below. The first column shows the name of the function and the given maximal cardinality (for instance, C1 means only single faults). The following two columns give the number of elements in each form of the FMEA for the given model.

<table>
<thead>
<tr>
<th>Model</th>
<th>#Classical</th>
<th>#Compact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thermal regulation (C1)</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>Thermal regulation (C2)</td>
<td>67</td>
<td>32</td>
</tr>
<tr>
<td>Acquisition (C2)</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Acquisition (C3)</td>
<td>31</td>
<td>4</td>
</tr>
<tr>
<td>Command (C1)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Command (C2)</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>Control and Monitoring (C1)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Control and Monitoring (C2)</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>Heating (C1)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Heating (C2)</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>Passive units (C1)</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Passive units (C2)</td>
<td>42</td>
<td>10</td>
</tr>
</tbody>
</table>

E. Slicing

Slicing is an automated model-transformation technique for reducing the size of the model while preserving its behavior. A sliced model thus can be analyzed faster than a full model. Key to slicing is the requirement under verification. A particular requirement usually covers only a part of the model. It takes the whole requirements specification to reflect a full model and its details. Hence, to verify a requirement, the full model can be sliced to only the behavior that is necessary to validate or to invalidate the requirement.

Our initial algorithm handles untimed SLIM models and requires no user intervention. It computes the control- and data-flow dependencies in the SLIM model and accounts for intricacies like dynamic reconfiguration and divergent behavior. Experiments on a small adder case study with
two integer generators \( a \) and \( b \) and three properties showed that, dependent on the property, the sliced model can be substantially smaller than the full model. The benefits are seen in the verification time and the amount of explored states, as shown in the table below.

<table>
<thead>
<tr>
<th>Property</th>
<th>#States</th>
<th>Memory (MB)</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsliced</td>
<td>1,676,026</td>
<td>272</td>
<td>7.3</td>
</tr>
<tr>
<td>( 0 \leq a \leq 60 )</td>
<td>1,437,691</td>
<td>211</td>
<td>5.4</td>
</tr>
<tr>
<td>( 0 \leq a, b \leq 30 )</td>
<td>533,553</td>
<td>84</td>
<td>1.4</td>
</tr>
<tr>
<td>( 0 \leq a \leq 30 )</td>
<td>9,379</td>
<td>33</td>
<td>0.1</td>
</tr>
</tbody>
</table>

The slicing algorithm was also used on the thermal regulation case (see Section IV), observing benefits of the same degree. These results have been published at NFM 2010 [14]. The extension to timed and hybrid models is an item for future work.

VI. FUTURE ACTIVITIES AND DIRECTIONS

A. Synthesis of Observability Requirements and FDIR

As part of our future work, we plan to further investigate the problem of observability requirements for FDIR. We intend to study the problem of automatically synthesizing observability requirements, and the FDIR subsystem itself. This problem generalizes that of diagnosability. In particular, diagnosability formally verifies whether a set of observables are always sufficient to satisfy a set of given diagnosis objectives, assuming the set of observables is given, whereas the problem of synthesis consists in automatically generating the set of observables, starting from a formal model of the system and the diagnosis objectives. Preliminary results [15] show the feasibility of this approach, and discuss two alternative algorithms for this purpose. Moreover, both algorithms can be used to synthesize minimal (in terms of set inclusion) or minimum (with respect to a user-defined cost function) sets of observables. Future directions include extending this framework to hybrid systems, and analyzing robustness with respect to sensor faults. A more difficult problem consists in synthesizing the FDIR subsystem itself. Future work includes the design and implementation of a general framework for FDIR synthesis, covering the FDIR development process lifecycle, and compliant with current FDIR architectures and strategies. This framework will allow for an early and more effective design and development of FDIR, taking into account both software dependability requirements and system architectural design, with shorter iterations that use automated techniques. Technically, we intend to leverage the use of model-based planning routines for synthesis, and use pre-defined library components as building blocks of the FDIR architecture.

B. Compositional Reasoning

The current practice of model checking is a strongly monolithic approach: given the overall model, compute the overall state space and then (or on-the-fly) check whether the requirement is satisfied. It does not exploit the hierarchical and component-oriented structure of space systems at a systems level. It would be beneficial in terms of computing resources to reason over each component separately and then combine the reasoned results in a semantically coherent manner.

In the past two decades, several of these compositional approaches emerged. The most promising are the assume-guarantee paradigm [16] and the Owicki-Gries reasoning [17]. The former considers everything outside the scope of a component as an environment. The assumptions of the environment are described as temporal formulae. If the assumptions hold, then the respective component satisfies a set of guarantees, which are also described as temporal formulae. The assumptions and guarantees can be automatically generated using automata learning algorithms (such as L*). Using proof rules, one can verify whether a set of components along with its assumptions and guarantees satisfy a global property. Owicki-Gries reasoning is instead invariant-based. It computes an approximative abstraction of the environment as invariants and then checks whether a property holds on a component in conjunction with the environment. The outcome could be indecisive, depending on whether the invariants expressing the environment are approximated with sufficient information. If this is not the case, the invariants can be refined using the state space computed so far. Both approaches are under investigation to support the reasoning over SLIM components.

C. Extending the Capability of Formal Analyses

One of the main shortcomings of the current dependability and performability analysis techniques as implemented in the COMPASS toolset is the mismatch between the model classes that can be analyzed, and those that can be specified in SLIM. For example, the faithful modeling of safety-critical systems requires both nondeterminism and continuous-time probabilities. Nondeterminism is ubiquitous in nominal system behavior, and naturally models, e.g., external effects (like system inputs) or the concurrent behavior of components. Errors, such as component failures, are subject to random phenomena and occur according to a continuous probability (like a Poisson) distribution. The combination of nominal and error behavior thus entails the investigation of models that exhibit both features. However, available analysis methods for such systems are very limited.

Similar problems arise when the combination of (nominal) hybrid and (faulty) probabilistic behavior is to be considered, or when continuous-time modeling of errors is to be completed by discrete-time transitions. In conclusion, this means that a significant set of SLIM models—and thus a substantial set of real systems—cannot be checked for their reliability and performance.
VII. CONCLUSIONS

The COMPASS project was initiated by the European Space Agency in order to support system-software co-engineering of real-time embedded systems, in particular aerospace systems. Within this framework, a component-based modeling approach was developed that is centered around the standardized AADL framework. To this aim, a significant subset of AADL, incorporating its recent Error Model Annex for modeling faults and repairs, has been extended to a System-Level Integrated Modeling (SLIM) language by adding features that support the detailed specification of the operational behavior of components, in particular covering the aspect of hybridity. Altogether the major distinguishing aspects of this component-based approach are the possibility to describe nominal hardware and software operations, hybrid (and timing) aspects, as well as probabilistic faults and their propagation and recovery. Moreover, it supports dynamic (i.e., on-the-fly) reconfiguration of components and inter-component connections.

Given a SLIM system specification, correctness properties, safety guarantees, and performance and dependability requirements can automatically be checked using the graphical COMPASS toolset. System requirements are formalized using property patterns which act as parameterized “templates” to the engineers and thus offer a comprehensible and easy-to-use framework for requirement specification. Instantiated properties are checked on the SLIM specification using state-of-the-art formal analysis techniques such as SAT-based and BDD-based symbolic model checking, and probabilistic variants thereof. The precise nature of these techniques together with the formal SLIM semantics yield a trustworthy modeling and analysis framework for system and software engineers supporting, among others, automated derivation of dynamic fault trees, FMEA tables, assessment of FDIR, and automated analysis of observability requirements.

The COMPASS toolset has been evaluated by TAS on several case studies and has provided valuable and usable results. The basic functionalities of the tool were found to be mature enough to be adopted by industry. In summary, the technology is promising and further efforts are needed to have it integrated in the overall model-based space engineering process.

REFERENCES