Model-based Analysis Support for Dependable Complex Systems in CHESS

Alberto Debiasi[2], Felicien Ihirwe[1], Pierluigi Pierini[1], Silvia Mazzini[1] and Stefano Tonetta[2]
[1]*Innovation Technology Service Lab, Intecs Solutions Spa, Pisa, Italy*
[2]*Embedded Systems Unit, Fondazione Bruno Kessler (FBK), Povo, Italy*

Keywords: CHESS, Model Driven Engineering, Complex Systems, Dependability Analysis.

Abstract: The challenges related to dependable complex systems are heterogeneous and involve different aspects of the system. On one hand, the decision-making processes need to take into account many options. On the other hand, the design of the system logical architecture must consider various dependability concerns such as safety, reliability, and security. Moreover, in case of high-assurance systems, the analysis of such concerns must be performed with rigorous methods. In this paper, we present the new development of CHESS, a cross-domain, model-driven, component-based and open-source tool for the development of high-integrity systems. We focus on the new recently distributed version of CHESS, which supports extended model-based development and analyses for safety and security concerns.

1 INTRODUCTION

The ever increasing complexity and dependability issues of systems in various domains, such as transportation, space, energy, health, and industrial production, requires effective design and development methods. The complexity and heterogeneity of components can be addressed with modeling approaches that span different technical disciplines and prove effective in the end-to-end engineering of the products. This implies taking into account various requirements such as quality, performance, cost, safety, security, and reliability. Model-based design technologies enable the user to perform beforehand different assurance-related activities such as physical architecture exploration, system's behavioral analysis, early verification, and validation.

The CHESS toolset (Cicchetti et al., 2012) offers a cross-domain modeling and analysis of high-integrity systems providing an integrated framework that helps the modeler (user) to automate different development phases: from the requirements definition, to the architectural modeling of the system's software and hardware, up to its deployment to an hardware platform (Cicchetti et al., 2012). CHESS follows a component-based approach where the user decouples different functional parts of the system as components that can be modeled, analyzed, verified, stored, reused individually, and be integrated to meet the system's common goals. CHESS supports, among other, schedulability and dependability analysis across the entire project life cycle. The results of the analysis are back-propagated to the model itself so that the modeler can review and fine-tune the model to satisfy real-time and dependability requirements.

CHESS tool recently became a full-fledged open-source project, hosted by The Eclipse Foundation (https://www.eclipse.org/chess/). The code has being developed by various contributors following an open-source approach with public projects for issue tracking, code repository branches, and continuous integration. This paper presents the latest development of CHESS to address software's security and safety analysis of the system exploiting the CHESS error model to represent faults and attacks. Analysis is completed by the integration with back-end tools such as xSAP (Bittner et al., 2016) and Mobius (Courtney et al., 2009) for the minimal cut sets analysis and Monte-Carlo simulation.

CHESS was developed, used and extended in many research projects, by both industrial and academic partners. To list a few, CHESS was involved in international projects such as the homonymous CHESS project[1], CONCERTO[2], and SESAMO[3],

[1]http://www.chess-project.org/
[2]http://www.concerto-project.org/
[3]http://sesamo-project.eu/

Debiasi, A., Ihirwe, F., Pierini, P., Mazzini, S. and Tonetta, S.
Model-based Analysis Support for Dependable Complex Systems in CHESS.
DOI: 10.5220/0010269702620269
In *Proceedings of the 9th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2021)*, pages 262-269
ISBN: 978-989-758-487-9

under the ARTEMIS Joint Undertaking initiative, AMASS[4], AQUAS[5], and MegaM@Rt[6], under the ECSEL Joint Undertaking initiative. CHESS has been applied in different domains such as Avionics (Godard and Nelissen, 2016), Automotive (Bressan et al., 2018), Space (Pace et al., 2014), Telecommunication (Mazzini, 2015), and Petroleum (Gallina et al., 2014)(Montecchi and Gallina, 2017).

The rest of the paper is arranged into five sections. Sections 2 provides an introduction to the CHESS tool building blocks and methodology, Section 3 presents the new major features released under CHESS 1.0.0, Section 4 discusses the related work. In Section 5 we present the envisioned future extension on CHESS and finally Section 6 concludes the paper.

2 CHESS IN A NUTSHELL

The CHESS modeling tool was released under the Eclipse PolarSys project[7] and recently it was moved from the incubation status to the first major release. The CHESSML is an integrated modelling language profiled from OMG standard languages: UML, SysML and MARTE under the Papyrus modeling environment [8] Not all the features from all the three languages were profiled to CHESS but only a specific subsets that suits CHESS's perspective. In particular sub-profiles supporting contract-based and dependability concerns have been defined, while MARTE has been adopted (with minor deviations) for what concern the timing perspective. There are different tools, plugins, and languages that were integrated into CHESS to support model validation, model checking, realtime and dependability analysis.

In this section, we are going to briefly describe the core aspects of CHESS methodology. We will also look in brief at different analysis mechanisms that are being performed in CHESS and how they link together to enhance system correctness.

2.1 Component-based Methodology

The CHESSML language supports a component-based development methodology enabling property-preserving component assembly for real-time and dependable embedded systems. Emphasis is given to *separation of concerns* between the functional and the non-functional dimensions, such as safety, security, reliability, performance, and robustness (Mazzini et al., 2015).

In CHESS, components at design level encompass functional concerns only (i.e., they are devoid of any constructs pertaining to tasking and specific computational model). The specification of non-functional attributes is then used for the automated generation of the container, enforcing the realization of the non-functional attributes declared for the component to be wrapped.

The CHESS methodology follows the "Correctness by Construction" practice which enforces *(1)* the use of formal and precise tools and notations for the development and the verification of all product items; *(2)* say things only once to avoid contradictions and repetitions; *(3)* the design of software components that are easy to verify, by e.g., using safer language subsets, and to implement, by using appropriate coding styles and design patterns (Panunzio and Vardanega, 2014).

2.2 Multi-view Modeling Approach

The CHESS tool provides a set of design views to uphold the "separation of concern", the "correctness by construction" and the other methodological principles introduced before. Six main views are defined to support the CHESS modeling approach. Throughout the development process, each view has its own underlined constraints that enforce its specific privileges on model entities and properties that can be manipulated. Depending on the current stage of the design process, CHESS sub-views are adopted to enhance certain design properties or stages of the process.

Requirement View is used to define system requirements and track their verification, *System view* provides a suitable frame for system-level design activities such as contract-based design, functional and dependability analysis. *Component view* composed of two sub-views, *Functional and Extra-functional views* which supports the design of system components logic and their internal compositions. *Deployment view* wich allows hardware structure modelling and the allocation of the corresponding software component instances. *Analysis view* capture all the analysis activities, diagrams and results. Finally, *Instance view* which is used to visualize and model the Platform Specific Model (PSM) as a joint instance of software to hardware allocated components.

[4]https://www.amass-ecsel.eu/
[5]https://aquas-project.eu/
[6]https://megamart2-ecsel.eu/
[7]https://projects.eclipse.org/projects/polarsys.chess
[8]https://www.eclipse.org/papyrus/

2.3 Model-based Analysis and Verification

CHESS provides the capability to perform several kinds of analysis depending on the specific requirements (functional, timing, dependability). Part of these functionalities have been added or extended to the new release of CHESS and are further explained in the next section.

Functional Verification: by means of model checking is supported by the integrated *nuXmv* model checker (Cavada et al., 2014). System and component properties, derived from requirements, can be formalized into linear temporal logic properties, then they can be verified on top of the system's or component's behavioral models developed using state machines.

Contract-based Analysis: is built on top of the OCRA tool support (Cimatti et al., 2013). Component formal properties are structured in terms of contracts, comprised of an assumption and a guarantee pairs. The assumption is a property to be fulfilled by the component's environment, and the guarantee is a property that must be satisfied by the component implementation - provided that the environment satisfies the assumption (Cimatti and Tonetta, 2015). The contract refinement can be exploited for architectural analysis, compositional functional verification (model checking) and safety analysis (see also Sec. 3).

***Timing Analysis*:** is built on top of the MAST[9] analysis tool. It is invoked to perform analysis such as schedulability and end-to-end response time analysis. Schedulability analysis is performed by taking input from the annotated PSM model and the computed partition schedule on each available processing unit. Then, the response-time analysis calculates the worst-case response time of each task (Godard and Nelissen, 2016) assessing the schedulable tasks complying with the given timing constraints. The end-to-end analysis is done by utilizing the component sequence diagram. Applying MARTE timing stereotypes, the tool evaluates the hardware component's responsiveness. This analysis facilitates "early end-to-end response time verification", giving a sense of any possible refinement of the model before deployment (Mazzini et al., 2015).

Dependability Analysis: include Failure Logic Analysis and State Based Quantitative Dependability Analysis. Failure Logic Analysis (Gallina et al., 2014) builds on top of Failure Propagation and Transformation Calculus (Wallace, 2005) and enables deductive as well as inductive hazards analysis towards semi-automatic generation of artefacts, necessary for arguing about HARA (Hazards Analysis and Risk Assessment). Supported by the DEEM tool (Bondavalli et al., 2000), State-Based Quantitative Dependability Analysis (Montecchi et al., 2013) supports safety engineers in the management of Reliability, Availability, Maintainability and Safety (RAMS) properties, and in the assessment of hazard rate threshold associated with safety integrity levels. Properties modeled using the CHESSML dependability profile are analyzed to obtain the probability of occurrence of specific failure modes and other quantitative metrics involving reliability, availability, and safety.

[9]https://mast.unican.es/

3 NEW SYSTEM-LEVEL ANALYSIS SUPPORT

In this section, we present the new major features released in CHESS 1.0.0. The new release includes extended support for system-level safety and security analysis. CHESSML dependability profile which normally supports different techniques for safety and dependability analysis has been extended to model fault injection and threats. Other new features include contract validations, parameter-based architectures, and document generation. This new release can be accessed at https://www.eclipse.org/chess/

3.1 Contract-based Design Analysis and Model Checking

CHESS supports the specification of component contracts specified in the OCRA contract language. Requirements are formalized into Formal Properties, which contain OCRA assertions, i.e., textual specifications of temporal logic formulas (see Fig. 1 for an example). In the contract-based paradigm, these properties are restricted to the related component interface. A contract is a pair of properties representing an assumption and a guarantee of the component. In addition, CHESS tool supports the contract refinement analysis for composite components. The contract of a composite component is defined by the assumption of the composite component itself and the guarantee ensured by the contracts of its sub-components, considering their interconnection as described by the architecture and and that the assumption of each sub-component is ensured by the contracts of the other sibling sub-components.

The new CHESS release has improved the contract-based analysis aspects by integrating CHESS with V&V tools such as OCRA, nuXmv, and xSAP. In this regard, the new additional analysis includes:

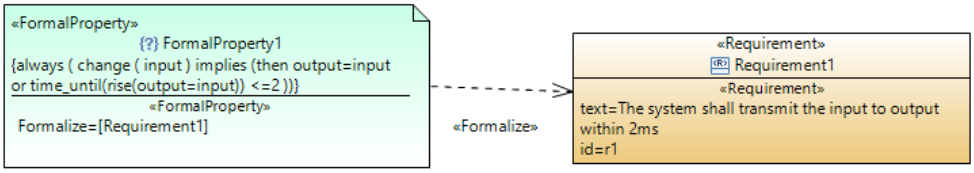

Figure 1: Example of a FormalProperty formalizing a requirement.

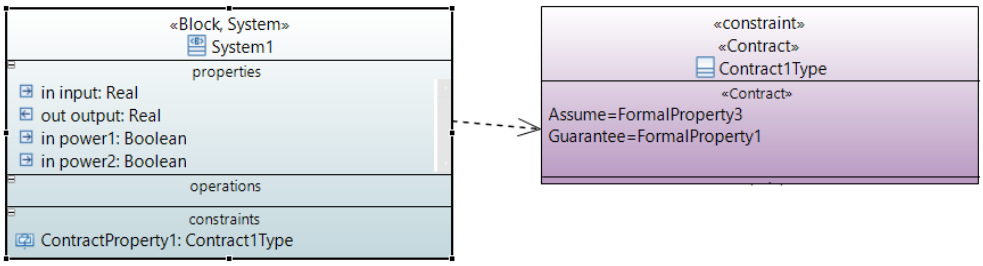

Figure 2: Example of a component contract.

(1) *Model checking*, i.e. the behavioral models, that describe how the internal state of a component and the output ports are updated, can be verified against some formal properties in different temporal logics. The formal properties can represent some requirements (e.g., functional or safety-related requirements) or some validation queries such as the reachability of states.

(2) *Contract-based compositional verification of state machines* is performed on composite components. The local state machine of each subcomponent is verified separately against its local contract. The correctness of the composite component is implicitly derived by the correctness of the contract refinement and the successful verification of local state machines.

(3) *Contract-based safety analysis*, i.e. identify the component failures as the failure of its implementation in satisfying the contract. When the component is composite, its failures can be caused by a failure of one or more sub-components and/or a failure of the environment in satisfying the assumption. As result, the analysis produces a fault tree in which each intermediate event represents the failure of a component or its environment, linked to a Boolean combination of other nodes. The top-level event is the failure of the system component. The basic events are the failures of the leaf sub-components, in addition to the failure of the environment (see (Bozzano et al., 2014) for more details).

3.2 Fault Injection and Safety Analysis

CHESS supports safety analysis based on fault injection thanks to the integration of xSAP(Bittner et al., 2016). The behavioral models of components are extended with faults and the tool automatically generates Fault Trees, showing the combinations of events leading to a failure or an undesired state, and Fault Mode Effect Analysis (FMEA) tables, listing all potential failure modes and their effects on the system (Bozzano and Villafiorita, 2013).

More specifically, once the system model is defined in CHESS, through components definition and their nominal behavioral model, the faulty behavior is expressed through a specific state machine called ”Error Model”. The Error Model extends the nominal state machine with information about the effect upon a property of the component, and consequently on its nominal behavior. Figure 3 represents an example of an error model that, in case of an internal fault, moves the related component in an error state where the property ”energy” is stuck at 0 value. The optional probability assigned to that transition is $5 \cdot 10^{-2}$.

Once the error model is defined, the Fault Tree Analysis (FTA) or FMEA can be done by invoking xSAP through the CHESS environment.

The xSAP approach is based on the library-based fault injection (i.e., an extension of a behavioral model with the definition of faults taken from a library of faults) and the use of model-based routines to generate safety artifacts.

The result of the FTA is the fault tree that is automatically shown in a dedicated panel in the front-end; see Figure 4 for an example. If fault probabil-

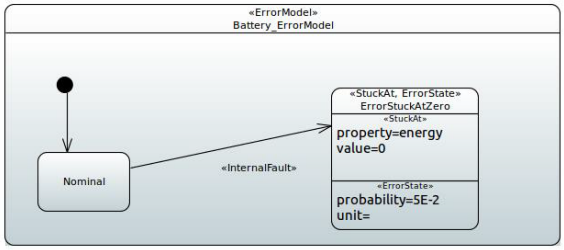

Figure 3: State machine modeling faulty behavior.

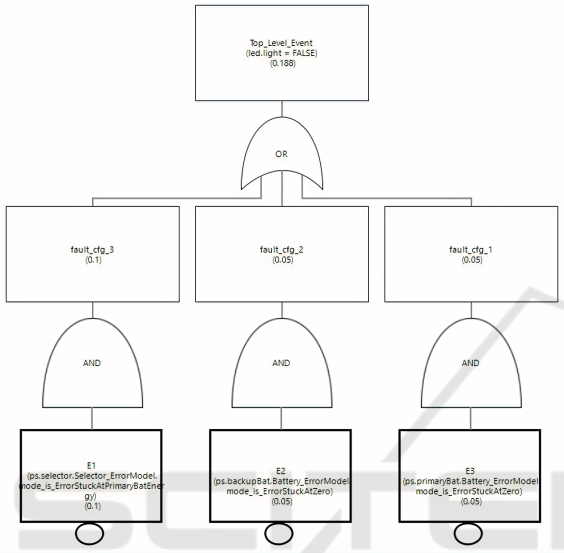

Figure 4: A fault tree visualized in the CHESS Editor View; note the probabilities associated to the top and basic events.

ities have been specified during the configuration of the error model, the fault tree will report their combination. The fault tree shows all minimal cut-sets, identifying the basic fault conditions which can lead to the top-level failure.

This is complementary to other existing analysis techniques supported in CHESS such as Failure Logic Analysis and State Based Quantitative Dependability Analysis, which do not consider the nominal behaviors and fault injection, but explicitly model the faulty behavior and fault propagation.

3.3 Quantitative Reliability Analysis

The CHESS profile for dependability is used to enrich functional models of the system with information regarding the behavior with respect to faults and failures, thus allowing properties like reliability, availability, and safety to be documented and analyzed.

The new release supports the modeling of security concerns which helps in threat identification at the early stages of the development and facilitates the exploiting of the Mobius capabilities for analysis of reliability. **Möbius**[10]**:** is a software tool for modeling the behavior of complex systems, by allowing the study of the reliability, availability, security, and performance for large-scale discrete-event systems (Courtney et al., 2009). Many reliability analysis results can be obtained with probabilistic models built with Mobius using the stochastic activity networks (SAN) formalism, solved via Monte-Carlo simulation[11].

Specific extensions of the dependability profile are related to the modelling of Cyber-Attacks aspects and model transformations from CHESS to the Mobius tool to run the analysis of SANs.

As results the implemented methodology allows modeling of a system security threat and data corruption which may result to service misfortune. An example of a system security threat can be a cyber-security attack, i.e. an unauthorized access of the system, halting services.

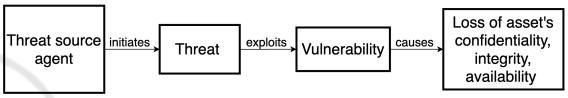

Figure 5: Process of Security breach.

Figure 5 depicts the process of a security breach that leads to the violation of the security-related properties. *A threat* event, initiated by a threat source agent, able to exploit a *vulnerability* of an asset (e.g. a component/system) may result in a loss to the confidentiality, integrity, and/or availability of the asset. Vulnerabilities could be represented as a pre-defined enumeration collected through different sources (e.g. personal competence, standards, results of previous threat analysis, etc.). Finally, the consequences could be modeled using pre-defined effects, which refers to the loss of Confidentiality, Integrity and Availability (CIA).

An <<***ErrorModel***>>-tagged state machine is used when modeling the security breach. The failure, internal fault, and effect are extended to include security threats, vulnerability, and consequences respectively. Figure 6 illustrates an example of an error model, where a cyber-security attack initiates a data corruption threat. The vulnerability was modeled exploiting the value check function which is set to false. In this case, the system transits to an erroneous state leading to component failure. Note that a component could have multiple instances of <<***ErrorModel***>>-tagged state machines, attached to it. Each instance would provide the elaboration of input/output failure behavior addressing a specific concern.

[10]https://www.mobius.illinois.edu/

[11]https://www.investopedia.com/terms/m/montecarlosimulation.asp

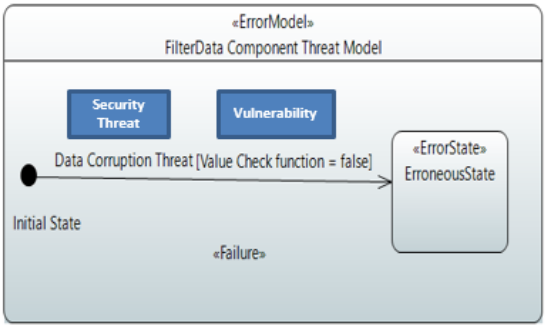

Figure 6: Erroneous state transition due to security threat event and vulnerability.

The generation of the Mobius SAN model process is done by performing an automatic model-to-model transformation from a model instance to the SAN model recognized by Mobius for reliability analysis.

The new reliability analysis is additional to the existing State Based Quantitative Dependability Analysis. It exploits the Mobius powerful and well supported analysis capabilities as engine for safety and security co-engineering, according to the scenario addressed in (Popov, 2017). Editing Mobius models can be non trivial. To this purpose we have extended the CHESS profile, the CHESS modelling language capabilities, and the user-friendly editor front-end to fully support the modelling of system architectures taking into consideration safety and security co-engineering, and by using automatic transformations to SAN model for reliability analysis with MOBIUS.

This extension has been developed in the context of the AQUAS project, as result of a collaborative effort among Intecs and City University of London, applied and evaluated across different use cases, in the ATM and Industrial Drive domain. This approach provides a smooth integration, guarantees the consistency among SysML and SAN models, and largely reduces the effort required to construct an appropriate SAN analysis model.

3.4 Support for Parameterized Architecture and Trade-off Analysis

In a parameterized architecture the number of components, the number of ports, the connections, and the static attributes of components depends on a set of parameters. Parameters are defined along the architectural hierarchy and, thus, the number of parameters themselves can depend on other parameters.

CHESS supports the the modeling of the parameterized architecture as well as its instantiation. In particular, the user can set the values of the parameters, defining the configuration of the architecture, and the tool automatically generates a concrete architecture corresponding to that configuration. Figure 7 shows an example of parameterized architecture.

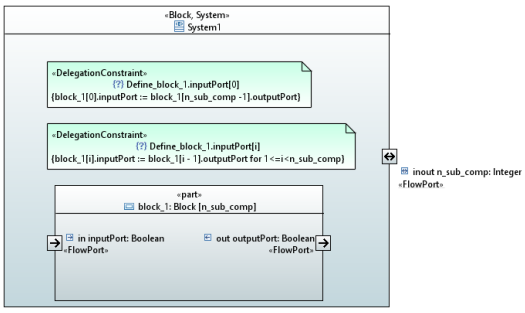

Figure 7: Example of parameterized architecture.

The parameterized architecture is also exploited for trade-off analysis, by performing various analyses on the different instantiations and comparing the results. This makes it easy to visually get an idea of how the intended model instances perform with respect to the selected configurations.

3.5 Automatic Generation of Diagrams and Documentation

The traditional way of editing a model is by adding an element in a diagram but changes made in the model are not reflected in the diagrams. The new CHESS release offers the possibility of generating a diagram from the model which reflects the data in the model on the fly. The supported diagrams are Block Definition Diagram (BDD) and Internal Block diagram (IBD). Multiple diagrams can be generated on a single component in the model. The generated diagram elements will be automatically aligned but the user can rearrange by moving elements manually or by invoking **"layout selection command."**

The new release also supports the generation of the model architecture and the report on various analyses executed on the model in an HTML document or a LaTeX source code. The report is divided into two sections. The first describes the structure of the model which includes diagrams and the associated components while the second includes the report lists of the results of the validation and verification (V&V) analyses results. There are many types of V&V results such as *property validation, assume/guarantee properties results, contract checks results, model checking, FTA, FMEA*, and so on.

4 RELATED WORK

Several commercial tools provide similar functionalities of CHESS. One of the most popular is **Matlab/Simulink**[12]. Although Simulink facilitates the modelling and analysis of complex systems, its simulation efficiency might be an important disadvantage. Being based on a single Model of Computation and Communication (MoCC) is another limitation. **CoFluent**[13] is another commercial tool extended to model IoT systems. Although supporting more interaction models than Matlab/Simulink, it is also limited in the way components may interact among them.

Another tendency is to overcome the UML lacks in semantic content, required in some application domains, towards a proliferation of DSLs (Brambilla et al., 2012). Among the available DSLs, UML/MARTE is the standard language for real-time and embedded systems design, while SysML is the standard language for system modeling. Several modelling environments like **Papyrus**[14] support UML/MARTE. Nevertheless, its flexibility and semantic richness requires the definition of efficient modelling methodologies.

Capella[15]**:** is an open-source comprehensive and extensible Eclipse system modelling tool. It is inspired to the SysML principles and it supports the ***ARCADIA*** methodology that is successfully deployed in a wide variety of industrial contexts (Bonnet et al., 2015). ARCADIA provides architectural descriptions for functional analysis, structural analysis, interfaces and behavior modeling, structured in five perspectives according to major system engineering activities and concerns.

COMPASS (Bozzano et al., 2019)**:** supports model checking, model-based safety, reliability, and performance analysis and shares with CHESS some of the tools used as backend for such analyses. Differently from CHESS, it targets a variant of AADL and does not support traceability and code generation.

MapleSim[16]**:** is a modeling tool for multi-domain engineering systems built on top of ***Modelica*** modeling language (Fritzson, 2015). MapleSim features an integrated environment in which the system equations can be automatically generated and analyzed (Cao and Wu, 2013).

Although we see some approaches able to tackle modeling challenges, no tool or approach has been able to fit in our methodology with such analysis and verification functionalities. Which makes CHESS a novel approach for implementing component-based modeling methodology for real-time and dependable systems by taking care of non-functional properties and enforces the correctness at all the stages of the development process.

5 FUTURE WORK

CHESS is a very huge toolset with more sophisticated and powerful functionality to meet user needs. However, there is still a gap for improvement, to cover more and more domains such as the Internet of Things (IoT) in a more concrete way. Note that we are not concluding that it is not capable to perform some basic modeling of IoT related scenarios but we aspire to make it more IoT specific. This extension will follow CHESS's component-based methodology and it will also follow already existing modeling approaches present in CHESS.

The envisioned approach will be achieved by improving the CHESSML metamodel with a set of specific stereotypes, contracts, communications, and operations profiled for IoT. The new proposed approach will also take in use of already existing dependability analysis infrastructure. We also plan to export IoT models developed with CHESS to external consumers. Finally, we plan to exploit the current CHESS's code generation support for Ada language, integrated with the open-source ThingML framework[17], for IoT code generation.

6 CONCLUSIONS

Dependable complex system design and development present several challenges, the well-known canonical approach is to divide complex systems into smaller chunks (or subsystems), build them separately, and later integrate them. In this paper, we presented the current state of the CHESS tool to tackle design, analysis, and verification of real-time dependable complex systems. We walked through the CHESS tool architecture and we highlighted its component-based and multi-view modeling approaches. We have also presented the new system-level extensions and capabilities of the tool released under the CHESS1.0.0 version.

[12]https://www.mathworks.com/products/simulink.html
[13]https://www.intel.com/content/www/us/en/cofluent/overview.html
[14]https://www.eclipse.org/papyrus/
[15]https://www.eclipse.org/capella/
[16]https://www.maplesoft.com/products/maplesim/
[17]https://github.com/TelluIoT/ThingML

ACKNOWLEDGEMENTS

This work has received funding from the Lowcomote project under European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement n813884. We would like to acknowledge also different projects funding leading to the mature realization of CHESS which include the CHESS[1], CONCERTO[2], SESAMO[3], AMASS[4], and AQUAS[5]. We would like to acknowledge the main contributors to the development of the CHESS toolset, in particular Stefano Puri, Nicholas Pacini, Luca Cristoforetti and Pietro Braghieri. Finally, we would like to acknowledge also Prof. Davide Di Ruscio for the assistance on drafting this paper.

REFERENCES

Bittner, B., Bozzano, M., Cavada, R., Cimatti, A., Gario, M., Griggio, A., Mattarei, C., Micheli, A., and Zampedri, G. (2016). The xSAP Safety Analysis Platform. In *TACAS*, pages 533–539. Springer.

Bondavalli, A., Mura, I., Chiaradonna, S., Filippini, R., Poli, S., and Sandrini, F. (2000). Deem: a tool for the dependability modeling and evaluation of multiple phased systems. In *DSN 2000*, pages 231–236.

Bonnet, S., Voirin, J.-L., Normand, V., and Exertier, D. (2015). Implementing the mbse cultural change: Organization, coaching and lessons learned. *INCOSE International Symposium*, 25(1):508–523.

Bozzano, M., Bruintjes, H., Cimatti, A., Katoen, J., Noll, T., and Tonetta, S. (2019). COMPASS 3.0. In *TACAS*, pages 379–385. Springer.

Bozzano, M., Cimatti, A., Mattarei, C., and Tonetta, S. (2014). Formal safety assessment via contract-based design. In *ATVA*, pages 81–97. Springer.

Bozzano, M. and Villafiorita, A. (2013). Safety critical systems. In *Encyclopedia of Software Engineering*. CRC Press (Taylor & Francis Group).

Brambilla, M., Cabot, J., and Wimmer, M. (2012). *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers.

Bressan, L., de Oliveira, A. L., Montecchi, L., and Gallina, B. (2018). A systematic process for applying the chess methodology in the creation of certifiable evidence. In *EDCC*, pages 49–56.

Cao, J. M. and Wu, T. (2013). Multi-domain modeling simulation and application based on maplesim. In *Mechatronics and Intelligent Materials III*, volume 706, pages 1894–1897. Trans Tech Publications Ltd.

Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., and Tonetta, S. (2014). The nuXmv Symbolic Model Checker. In *CAV*, pages 334–342. Springer.

Cicchetti, A., Ciccozzi, F., Mazzini, S., Puri, S., Panunzio, M., Zovi, A., and Vardanega, T. (2012). CHESS: a model-driven engineering tool environment for aiding the development of complex industrial systems. In *ASE*, pages 362–365.

Cimatti, A., Dorigatti, M., and Tonetta, S. (2013). OCRA: A tool for checking the refinement of temporal contracts. In Denney, E., Bultan, T., and Zeller, A., editors, *ASE*, pages 702–705. IEEE.

Cimatti, A. and Tonetta, S. (2015). Contracts-refinement proof system for component-based embedded systems. *Sci. Comput. Program.*, 97:333–348.

Courtney, T., Gaonkar, S., Keefe, K., Rozier, E. W. D., and Sanders, W. H. (2009). Möbius 2.3: An extensible tool for dependability, security, and performance evaluation of large and complex system models. In *DSN*, pages 353–358.

Fritzson, P. (2015). *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. Wiley-IEEE Press, 2 edition.

Gallina, B., Sefer, E., and Refsdal, A. (2014). Towards safety risk assessment of socio-technical systems via failure logic analysis. In *ISSRE Workshops*, pages 287–292.

Gallina, B., Sefer, E., and Refsdal, A. (2014). Towards safety risk assessment of socio-technical systems via failure logic analysis. In *ISSRE Workshops*, pages 287–292.

Godard, W. and Nelissen, G. (2016). Model-based design and schedulability analysis for avionic applications on multicore platforms. *Ada User Journal*, 37:157–163.

Mazzini, S. (2015). The concerto project: An open source methodology for designing, deploying, and operating reliable and safe cps systems. *Ada User Journal*, 36:264–267.

Mazzini, S., Favaro, J., and Baracchi, L. (2015). A model-based approach across the IoT lifecycle for scalable and distributed smart applications. In *ITSC*, pages 149–154.

Montecchi, L. and Gallina, B. (2017). Safeconcert: A metamodel for a concerted safety modeling of socio-technical systems. In *MBSA*, pages 129–144, Cham. Springer International Publishing.

Montecchi, L., Lollini, P., and Bondavalli, A. (2013). A reusable modular toolchain for automated dependability evaluation. In *VALUETOOLS*, page 298–303. ICST.

Pace, L., Pasquinelli, M., Gerbaz, D., Fuchs, J., Basso, V., Mazzini, S., Baracchi, L., Puri, S., Lassalle, M., and Viitaniemi, J. (2014). Model-based approach for the verification enhancement across the lifecycle of a space system. In *INCOSE CIISE2014*.

Panunzio, M. and Vardanega, T. (2014). A component-based process with separation of concerns for the development of embedded real-time software systems. *Journal of Systems and Software*, 96:105 – 121.

Popov, P. (2017). Models of reliability of fault-tolerant software under cyber-attacks. In *ISSRE*, pages 228–239.

Wallace, M. (2005). Modular architectural representation and analysis of fault propagation and transformation. *Electronic Notes in Theoretical Computer Science*, 141(3):53 – 71. FESCA 2005.