CHESS: an open source methodology and toolset for the development of critical systems

Silvia Mazzini, John Favaro, Stefano Puri, Laura Baracchi

Intecs S.p.A.
Pisa, Italy
silvia.mazzini@intecs.it

Abstract. This paper presents the CHESS open source methodology and toolset, aiming to improve MDE practices and technologies to better address safety, reliability, performance, robustness and other non-functional concerns, while guaranteeing correctness of component development and composition for critical embedded systems.

Keywords: Model-based, component-based, correctness-by-construction, separation of concerns, model transformation, contract-based, formal methods, real-time analysis, systems/software co-engineering, open-source.

1 Introduction

The speedup of technological progress and of time to market have caused all phases of systems development to be compressed and accelerated. At the same time designing and building complex systems involves many different roles and expertise (for example, software, hardware and dependability engineering), with a consequent need for systematic and disciplined development paradigms.

A model driven engineering (MDE) approach is theoretically the ideal solution, providing formal and semantically grounded support for the design of the system, capable of capturing the overall characteristics as well as detailed properties of all its composing parts.

When designing software, MDE can exploit the unique opportunity that arises thanks to the fact that software models are software themselves. This introduces the possibility to generate a software product through a sequence of automated model transformations: if the model in input provides all the required information and model transformations are proved correct, the final software product is guaranteed to reflect the properties of the model, thus implementing a *correct-by-construction* development process.

Despite its theoretical credentials and academic acknowledgement, however, industrial level tools may be inadequate or too expensive, and MDE is still often perceived by the industry as an extra burden, so that, in our experience, more traditional approaches are often pursued.

With the CHESS methodology and supporting toolset (originally developed in the CHESS project [4] and then enhanced in the CONCERTO project [3] focusing on the development of multi-core systems and on the extensions for a wider domain coverage) we aimed to improve MDE practices and technologies to better address safety, reliability, performance, robustness and other non-functional concerns, while guaranteeing correctness of component development and composition for embedded systems.

CHESS was developed as an open source project, mainly to improve its visibility, usability and standardization. This approach is fundamental for enabling the most fruitful collaboration between research and technology providers, allowing wide exploitation of prototypes and thus an optimal basis for tool maturation. Moreover, in the area of embedded critical systems targeted by CHESS, commercial off the shelf tools tend to be extremely costly and somewhat rigid, whereas an open source technology has the competitive advantage of its zero/low cost, while still supporting a feasible business model, based on the providers' offer of customizations, support, consulting and training.

2 The CHESS Component Model

The CHESS methodology relies on the CHESS Component Model, which is built around the concepts of components, containers and connectors. It supports the *separation of concerns* principle, strictly separating the functional aspects of a component from the non-functional ones.

According to the CHESS Component Model, a component represents a purely functional unit, whereas the non-functional aspects are in charge of the component's infrastructure and delegated to the *container* and *connectors* (Figure 1).

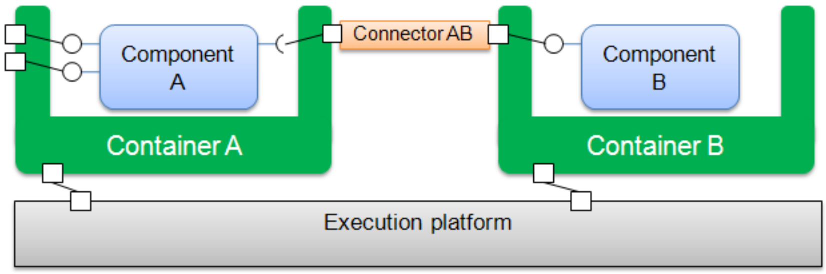

Figure 1: Component, Container and Connector

The container [20] can be regarded as a wrapper enveloping the user's component, which is responsible for the realization of all non-functional properties that are specified for the component that it embeds. The container also mediates the access of the component to the executive services it needs from the execution platform. The connector [17] is responsible for the interaction between components; it allows to decouple interaction concerns from functional concerns.

From the interaction perspective, components are considered as black boxes that expose only their provided and required interfaces. Non-functional attributes are specified by decorating the component's interfaces with non-functional properties; e.g. regarding real-time concerns the activation pattern (e.g. sporadic or cyclic) can be specified for each component's provided operations.

The declarative specification of non-functional attributes of a component, together with its communication concerns, is used in CHESS for the automated generation of the containers and connectors that embody the system's infrastructure. In particular, when a component is assigned to a processing unit, we can generate the container within which the component is going to be deployed on the execution platform of the processing unit. Indeed, the internal structure of containers depends on the non-functional attributes required for the components they may embed. Deterministic rules need to exist for containers to be automatically generated from the attributes set on the model. For instance, for every computational model, execution platform pair, the set of allowable containers realizing internal threads and its protected objects can be defined and factored in a library of code archetypes, which can then be later used to simplifies automatic code generation [21].

In principle, there is a default 1:1 correspondence between a component and its container. However, if – e.g., for reasons of local optimization – selected operations of distinct components should be allocated to the same thread, then multiple components could be allocated to a single container.

The key properties of the CHESS component model are *compositionality* and *composability*. Compositionality is achieved when the properties of the system as a whole can be determined as a function of the properties of the constituting components and the execution environment. Composability, rather, is achieved when individual components' properties are preserved on component composition, deployment on target and execution.

Compositionality and composability are guaranteed in CHESS not only for functional properties, but also for non-functional properties, such as real-time and dependability. This way, the ambitious goal of composition with guarantees [7] is achieved, implementing the correctness by construction [8] theory.

3 The CHESS Design Flow

Following the CHESS methodology, the user specifies the system's components, declaring their functional and non-functional properties, thus providing a Platform Independent Model (PIM) to represent the solution to the problem, independent of any specific implementation. Then the modeler complements the PIM with information on the target platform and the deployment plan. By using a dedicate profile language, analysis about failure propagation is performed at PIM level, for system, SW and platform specification, to allow early dependability analysis.

Automated model transformation produces a Platform Specific Model (PSM) from the user PIM and platform specification; in particular the containers and connectors entities are created in the PSM. The PSM is read-only: this way the implementation product is guaranteed to be deterministic.

Real-time analysis, such as schedulability analysis, end-to-end response time analysis and analysis of different scheduling algorithms for multicore deployments, is performed on the PSM, with back propagation of results to the PSM, PIM platform and deployment models. The modeler can iterate these steps as many times as necessary until satisfactory analysis results are obtained.

At this point, the implementation is deployed to the HW, with run-time verification support if needed. Run-time monitoring is activated to collect live data for run-time monitoring analyses and back propagation of results.

The CHESS methodology enables early verification, as possible inconsistencies and integration issues will surface at the earliest stages of the process. It also supports *system-software co-engineering* as a seamless process, by keeping traceability between system level entities and requirements on one side and the corresponding software and hardware level entities on the other side.

4 Contract-based Modeling Extensions to CHESS

Contract-based reasoning was first envisaged as an extension to CHESS in the ESA funded FoReVer study [6] and further elaborated within the SafeCer project [5]. Component properties are formalized in terms of *contracts,* composed of an *assumption* and a *guarantee* models as formal properties, where the assumption is a constraint on the component's environment or usage, and the guarantee is a property that must be satisfied by the component - provided that the environment satisfies the assumption.

The CHESS extended methodology introduced *stepwise refinement*, where the decomposition of a component is accompanied by the decomposition of its contracts, as a central activity in the development process. Stepwise refinement is subject to formal verification and is a key point in the overall verification process as in [9].

Support for modeling contracts and for stepwise refinement is provided in the extended CHESS toolset. Formal verification of the contract refinement is performed by OCRA (Othello Contracts Refinement Analysis) [11] by Fondazione Bruno Kessler for the verification of logic-based contracts refinement for embedded systems [12], which is integrated in the CHESS extension.

This extended methodology can be exploited at its best if a library of standard qualified components with associated contracts is available. In the top-down modeling process, a library of components represents a bottom-up driver to ensure convergence to a feasible solution based on the reuse of possibly certified components.

CHESS is currently the subject of extension and adaptation in the context of the AMASS ECSEL project [10]. The goal of AMASS [13] is to create an open tool platform, ecosystem, and self-sustainable community for assurance and certification of Cyber-Physical Systems for different domains of interest. In particular, the project will investigate how the usage of CHESS, that is, its contract-based component model, verification and code generation features, can enable architecture-driven assurance support.

5 The CHESS Toolset

The CHESS toolset [1] provides an integrated framework to support the CHESS methodology. It assists the modeler throughout the whole development process, following the CHESS methodology, from the definition of requirements, to the modeling of the system's architecture, down to the software design and its deployment to hardware components. It also offers support for the analysis of selected real-time and dependability features (in particular, failure propagation and state-based) as well as code generation functionality to automatically generate the infrastructure code needed to implement the non-functional properties defined in the model. Generation of the infrastructure code for Ada is currently supported; of course other target languages can be addressed as well.

The CHESS toolset was developed as a set of Eclipse plugins based on MDT Papyrus (the Eclipse UML editor) and on the CHESS Modeling Language, which was defined as an extension of the UML, SysML and MARTE modeling languages [19].

We decided to rely mainly on SysML for the modeling of requirements and for the system level design, on UML for modeling software aspects of the system, and on MARTE for describing the real-time aspects, staying as close as possible to the standard modeling languages. In particular a profile has been defined on top of UML to model failures definition and their intra/inter-components propagation, while SysML has been extended to offer support for contract based design.

MARTE has been used and extended to be able to model real-time properties for component instance interfaces; indeed, MARTE support which allows to specify real-time property for component's operations exposed through ports (through the RtSpecification entity), cannot be used at component instance level, which is the most appropriate level where real-time properties must be provided (e.g. the periodic activation of an operation can be different for two instances of the same component providing the given operation).

A specific profile was also developed for the avionics domain to allow modeling and analysis of ARINC 653 architectures [17]. This way the CHESS toolset provides an open framework to accommodate the widest possible set of users from different domains.

6 The CHESS Open Source Project

CHESS results are included in the PolarSys[1] initiative, an industrial group for promoting open source tools for embedded systems: the CHESS core technology is available in PolarSys as open source project [2], and CHESS interfaces are published to enable other platform and tool providers to develop additional features for integration with CHESS and to exploit new CHESS functionalities as they become available. The CHESS open source initiative has received valuable input from several academic partners. Since the initial contribution provided by Intecs and University of Padua, new

[1] https://www.polarsys.org/

contributors have joined the CHESS Polarsys project; in particular the Mälardalen University and the University of Florence provided extensions for the modelling and analysis of dependability properties of interest in their research. A proposal about extension of the current support for contract based analysis is also currently under evaluation.

Industrial parties have expressed interest in the CHESS project, also suggesting desired improvements (e.g. C code generation support). Although usage of CHESS in the industry is nascent, very positive results from case studies performed in several research projects have demonstrated that the CHESS approach and toolset can offer valuable support for the development of cyber-physical systems.

The CHESS modelling environment is based upon the open source project Papyrus, which is one of the most appreciated open source tools in the industry; in particular, recently the Papyrus Industry Consortium[2] has been created to support a model-based engineering platform based on the domain specific and modeling capabilities of the Eclipse Papyrus family of products. We think that having Papyrus as the baseline editor can foster the interest around CHESS.

Use of other open source resources has permitted us to make valuable extensions. For example, real-time analysis is performed in the CHESS toolset thanks to its integration with an extension to the MAST engine [14], making it possible to perform schedulability analysis and end-to-end response time analysis for multi-core architectures. Another example is the dependability analysis support CHESS provides: quantitative state based analysis is performed via integration with the DEEM server [15] [18] (while qualitative dependability failure logic analysis to calculate system level failure behavior given the failure behavior of the individual components established in isolation [16] is directly integrated in CHESS).

7 Discussion and Conclusions

The usage of the CHESS methodology and toolset has been experimented with in the context of several research projects where use cases from different domains (e.g. telecom, automotive, avionics, space, industrial automation and petroleum plants) provided an interesting testbed for validation of the process and for providing domain specific extensions to better accommodate specific needs and standards.

In some occasions, when collaborating with industrial users to validate the tool, we found that widely acknowledged commercial tools allow a higher degree of freedom to the user and may be easier to use in a traditional development process, if compared to the strictly disciplined and almost guided modeling process supported by CHESS. This can be considered as a drawback, as it requires users to have a solid academic background in modeling and imposes a slow learning curve at the beginning. However, the higher level of freedom allowed by some commercial tools comes at the cost of producing models for which feasibility analysis cannot always be performed in a sound and deterministic manner.

[2] https://www.polarsys.org/ic/papyrus

By following the systematic and rigorous design process prescribed by CHESS, supported by its correct model transformations, the semantic meaning of each analysis artefact and analysis operation is guaranteed to correspond to the semantic meaning of the modelling artefact and decoration attribute in the user model. The user model is therefore guaranteed, by construction, to be statically analyzable for feasibility.

The CHESS methodology and toolset are in an advanced prototypical stage and may need to be engineered, but we strongly believe that, being available as open source, CHESS provides an important opportunity for the future of the development of complex critical systems.

8 Acknowledgements

This work has been partially supported by the ARTEMIS-JU projects: CONCERTO [3], CHESS [4] and SafeCer [5], the ECSEL project AMASS [14] and by the ESA funded project FoReVer [6].

9 References

1. A. Cicchetti, F. Ciccozzi, S. Mazzini, S. Puri, M. Panunzio, A. Zovi and T. Vardanega,“CHESS: A Model-Driven Engineering Tool Environment for Aiding the Development of Complex Industrial Systems”, Proceedings of Automated Software Engineering (ASE) International Conference, Essen, July 2012.
2. CHESS PolarSys project, [Online], Available: https://www.polarsys.org/projects/polarsys.chess [Accessed: July 15, 2016].
3. CONCERTO project: “Guaranteed Component Assembly with Round Trip Analysis for Energy Efficient High-integrity Multi-core Systems”, Artemis Call 2012 333053, [Online], Available: http://www.concerto-project.org/ [Accessed: July 15, 2016] .
4. CHESS project: “Composition with guarantees for high-integrity embedded software components assembly”, [Online], Available: http://www.chess-project.org/ [Accessed: July 15, 2016].
5. SafeCer project: “Safety Certification of Software-Intensive Systems with Reusable Components”, [Online], Available: http://safecer.eu/ [Accessed: July 15, 2016].
6. FoReVer project: “Functional Requirements and Verification Techniques for the Software Reference Architecture”, ESA funded project, [Online], Available: https://es-static.fbk.eu/projects/forever/ [Accessed: July 15, 2016]
7. T. Vardanega, “Property Preservation and Composition with Guarantees: From ASSERT to CHESS”, in: Proc. of the 12th IEEE International Symposium on Object/Component/Service Oriented Real-Time Distributed Computing, 2009, 125 – 132.
8. R. Chapman, “Correctness by Construction: a Manifesto for High Integrity Software”, Proceedings of the 10th Australian workshop on Safety critical systems and software - Volume 55, Pages 43-46, 2006.
9. L. Baracchi, S. Mazzini, G. Garcia, A. Cimatti and S. Tonetta, “The FOREVER Methodology: a MBSE framework for Formal Verification”, Proceedings of DASIA Conference, Porto, May 2013.

10. AMASS project: “Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems”, [Online], Available: http://www.amass-ecsel.eu/ [Accessed: July 15, 2016].
11. OCRA: “a command-line tool for the verification of logic-based contract refinement for embedded systems”, [Online], Available: https://es-static.fbk.eu/tools/ocra/ [Accessed: July 15, 2016].
12. A. Cimatti and S. Tonetta, “A Property-Based Proof System for Contract-Based Design”. EUROMICRO-SEAA 2012: 21-28.
13. A. Ruiz, B. Gallina, J.L. de la Vara, S. Mazzini and H. Espinoza, “AMASS: Architecture-driven, Multi-concern, Seamless, Reuse-Oriented Assurance and Certification of CPSs”. 5th International Workshop on Next Generation of System Assurance Approaches for Safety-Critical Systems (SASSUR). SafeComp, International conference on computer safety, reliability and security, Trondheim, September 2016.
14. MAST: “Modeling and Analysis Suite for Real-Time Applications”, [Online], Available: http://mast.unican.es/ [Accessed: July 15, 2016].
15. DEEM: “DEpendability Modeling and Evaluation of Multiple Phased Systems”, [Online], Available: http://rcl.dsi.unifi.it/projects/tools [Accessed: July 15, 2016].
16. B. Gallina and E. Sefer, “Towards Safety Risk Assessment of Socio-technical Systems via Failure Logic Analysis” submitted to RISK 2014.
17. A. Baldovin, A. Zovi, G. Nelissen, S. Puri, “The CONCERTO Methodology for Model-Based Development of Avionics Software” Chapter Reliable Software Technologies – Ada-Europe 2015 Volume 9111 of the series Lecture Notes in Computer Science pp 131-145, June 2015.
18. “A Reusable Modular Toolchain for Automated Dependability Evaluation”, Proceeding ValueTools '13 Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools pp 298-303, December 2013.
19. “CHESSML profile”, [Online], Available: https://www.polarsys.org/chess/publis/CHESSMLprofile.pdf [Accessed: July 15, 2016].
20. N. R. Mehta, N. Medvidovic, and S. Phadke. Towards a taxonomy of software connectors. In Pro. Of the 22nd International Conference on Software Engineering, pages 178–187, 2000.
21. M. Bordin and T. Vardanega. Atomated model-based Generation of Ravenscar-Compliant Source Code. In Proc. If the 17th Euromicro Conference on Real-Time Systems 2005.