A model-based approach across the IoT lifecycle for scalable and distributed smart applications

Silvia Mazzini, John Favaro, Laura Baracchi
Intecs S.p.A.
Pisa, Italy
silvia.mazzini@intecs.it

Abstract—**A model-based methodology is presented for providing effective and efficient property-preserving mechanisms for non-functional properties such as real-time, safety, security and performance for Internet of Things (IoT) system components. The presented methodology and related supporting tool chain provide a means to model, develop, analyze, verify, operate, manage and monitor heterogeneous mission-critical smart applications with distributed scalable deployments of IoT components, such as the data distribution services of Intelligent Transport Systems, which are increasingly being deployed in loosely coupled federations of nodes.**

Keywords—model-based; contract-based; modeling; IoT; smart systems; formal verification; requirements engineering; co-engineering.

I. Introduction

A smart application implies a variety of systems that must cooperate, directly or indirectly. The involved systems belong to the Internet of Things (IoT), displaying independent sensing and actuating capabilities as well as computation, collaboration and control activities resulting in possibly critical behavior (e.g. control functions on board of a vehicle). Together they form a system of systems, whose goal is to embody an emergent behavior capable of improving important issues, such as transportation issues in a smart mobility application.

Much progress has been made in recent years in the modeling of systems typical of the IoT, whose main characteristics include extreme heterogeneity, inherent concurrency, dynamic adaptability, and high levels of interaction with the physical world through sensors and actuators. A good example of such systems are the backbone data distribution systems of Intelligent Transport Systems, which are beginning to move away from centralized hierarchical architectures to more loosely coupled federations of nodes that can be scaled to the size and density of a region.

Among the most challenging aspects related to the development of a smart application, is the need to integrate complex IoT systems that are developed by separate engineering teams/companies, following independent lifecycles and timing schedules. It is of paramount importance to be able to evaluate beforehand, right from the initial phases, the combined behavior of the overall system, including critical aspects related to safety, security and real-time requirements.

Other important aspects of a smart application are the dynamic nature of configuration in the IoT and its intrinsically distributed nature. These aspects make the integration challenge more difficult, exacerbate the infrastructural costs of system verification and validation, and require effective management and monitoring of the system to be carried out during its operation.

It is in this perspective that the adoption of a model-based approach across the whole development life cycle offers its greatest advantages.

II. A Model-Based Approach

Model-based system engineering (MBSE) is the application of modeling to support system requirements, design, analysis, verification and validation activities starting from the initial engineering phases of the system life cycle and continuing in their manufacturing, operation and maintenance, until disposal.

MBSE aims to provide a common core modeling language and an integrated methodological framework for system engineering to provide consistency, integration and control over the engineering (and the related modeling) of the different involved disciplines, possibly adopting an adequate level of abstraction in modeling.

One of the basic advantages in the adoption of a model-based development methodology, with respect to a document-centric approach, is that it fosters the systematic formalization of relevant information on the system, its components and properties, in terms of specific model elements whose semantics can be understood by a person, but can also be automatically processed by engines for formal verification, analysis, simulation or code generation.

The systematic adoption of the MBSE approach introduces the opportunity of performing *early verification* on high-level models of the system, thus reducing possible late - and thus expensive and time consuming - discovery of inconsistencies for instance during the integration phase. This is a particular problem in the complex, scalable reactive systems that are becoming common in the IoT, in which verification sometimes occurs as late as after deployment in the field.

Another important aspect of MBSE is that requirements are part of the model themselves and play a central role in the system development lifecycle: system elements are associated to the technical requirements they satisfy, which are in turn traced to higher level requirements, up to system level requirements. This way the impact of changes can be better evaluated and consistent model verification evidence with respect to the requirements can be provided throughout the development process.

Thanks to the clear definition of interfaces in the models, the perimeter and decomposition of responsibilities for the system's components become self-evident, easing considerably the whole system's lifecycle management. This is well synchronized with current interface standardization efforts in various subfields of the Internet of Things.

Finally the adoption of an MBSE methodology allows a smooth iterative and incremental development process, encouraging a collaborative attitude among the involved stakeholders, allowing discussions at all levels to be firmly anchored to the models that represent the development status and target requirements.

III. The CHESS methodology and toolset

CHESS is a methodology and supporting toolset which is the principal result of Intecs' participation in several R&D projects, starting from the original CHESS project [1], to provide a model-based solution to address the challenges of developing critical real-time and embedded systems, by adopting a component-based approach.

CHESS results are included in the PolarSys initiative, an industrial group for promoting open source tools for embedded systems: the CHESS core technology is available in PolarSys as open source [2], and CHESS tools 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.

CHESS promotes a *component-based* development process in which particular emphasis is given to the ability to specify the non-functional properties of components, including critical properties such as time predictability, isolation, transparency and other real-time and dependability related characteristics.

The CHESS toolset [3] provides an integrated framework to support the modeler through 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 *real-time and dependability analysis* as well as *code generation* functionality to automatically generate the infrastructure code (currently Ada) needed to implement the non-functional properties defined in the model.

CHESS relies on the concept of "Model Driven Engineering" [4], [5]: an extension of model-based engineering that exploits 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 automated model transformation.

In extreme synthesis the CHESS toolset is the core modeling and analysis environment: an Eclipse-based framework delivered as open source within the PolarSys initiative.

The CHESS toolset accompanies the developer through every step of the development process, from presenting clean, uncluttered, single-concern design spaces, to providing constant, informative verification feedback to modeling actions, it encourages a disciplined, productive development experience in the highly complex world of dependable real-time embedded systems and through a correct-by-construction derivation approach, the work of the developer results at the end in concrete, platform-specific implementations with high-integrity runtime guarantees.

Real-time analysis is performed in CHESS thanks to the integration [1] with an extension to the MAST engine [6], allowing to perform schedulability analysis and end-to-end response time analysis, based on the modeled scenarios and on the non-functional properties used to decorate the models. End-to-end analysis is important in Smart Systems because often the individual components are provided only with guarantees of local response time, but it is extremely difficult to understand response times across the breadth of the entire Smart System without deployment (which is very late in the lifecycle). CHESS provides an alternative for "early end-to-end response time verification".

Moreover CHESS provides *dependability analysis* support: quantitative state based analysis is performed via integration[2] with the DEEM server [7], while qualitative dependability failure logic analysis [8] is directly integrated in CHESS.

Real-time and dependability analysis results are back-propagated onto the model itself, enabling the modeler to perform some tuning on the model to satisfy real-time and dependability requirements.

A. *Component-based modeling with guarantees*

Separation of concerns and *correctness by construction* are the two main theoretical principles around which CHESS was conceived and developed.

Separation of concerns is first implemented in CHESS by the adoption of a *multi-view* approach, where the system is designed using a single model with dedicated consistent views, which are specialized projections of the system in specific dimensions of interest. This corresponds to the "Conceptual Frameworks" systems engineering approach, based on the identification of different "viewpoints", where a viewpoint identifies a set of concerns, representations and modeling techniques that are specific to a stakeholder. A "view" is the result of applying a viewpoint to a particular system.

For this purpose the CHESS modeling framework organizes the modeling activities in separate *views*: starting

[1] integration with MAST is implemented thanks to model transformations, developed by Intecs in collaboration with the University of Padova, applied on the PSM.
2 integration with DEEM is implemented thanks to model transformations, developed by Intecs in collaboration with the University of Florence, applied on the PSM.

from the definition of requirements and the description of the system with its hierarchical decomposition, using a SysML profile respectively in the Requirements View and in the System View, continuing with the association of requirements to components, the definition of hardware and software components using a UML/MARTE profile in the Component View, down to the Analysis View and the Deployment View.

The CHESS *component model* [9] further supports the separation of concerns principle: the functional aspects of a component being allocated to the component's internals, whereas all non-functional aspects are in charge to the component's infrastructure. From the interaction perspective, components are considered as black boxes that only expose provided and required interfaces. Non-functional attributes (fundamental in case of critical components) that need to be set are specified by decorating the component's interfaces with non-functional properties.

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 container and connectors, depicted in Fig. 1. As a consequence, in the CHESS approach components at design level encompass functional concerns only: in particular, they are devoid of any constructs pertaining to tasking and specific computational model concerns (e.g. a real time activation pattern for an operation), or interaction between components (e. g. safety protocols to be adopted in the communication).

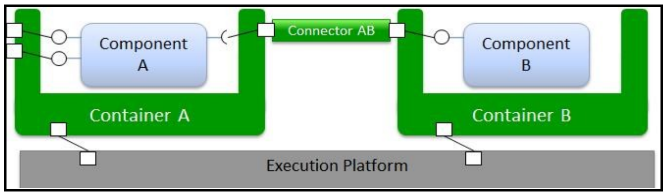

Fig. 1 The CHESS Component Model

The CHESS component specification, as defined at user level in terms of functional behavior and non-functional property specification, is therefore completely platform-independent: it represents the Platform Independent Model or PIM. Containers and connectors, instead, are platform specific and form the Platform Specific Model or PSM.

The automatic generation of the PSM from the PIM is achieved in CHESS thanks to the application of the CHESS component model and to the implementation of model-to-model transformations, resulting from the combined effort of Intecs and University of Padova.

The key properties of the CHESS *component model* are *compositionality* and *composability*.

Compositionality is an important concern at system level: it 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, instead, is achieved when individual components' properties are preserved on component composition, deployment on target and execution.

Compositionality and composability can be referred in CHESS not only to functional properties, but also to non-functional properties (e.g. real-time, and dependability) which are of fundamental importance when dealing with the behavior of critical systems.

The specification of non-functional behavior is super-imposed on component interfaces in a way that preserves their original functional semantics and enriches it with non-functional semantics separately realized by the component infrastructure, as depicted in Fig. 1.

Based on the characteristics of compositionality and composability, CHESS provides a process where components' non-functional properties can be modeled, assured by analysis, guaranteed throughout the development process from design to implementation, and actively preserved at run time. This way the ambitious goal of *composition with guarantees* [10] is achieved, implementing the correctness by construction [11] theory.

Compositionality and composability are desirable properties of IoT systems because "Smart Things" (e.g. Smart Homes, Smart Cities) are so often constructed as assemblies of separately developed components which may well exhibit undesired feature interaction at the system level. The developer of individual Smart Components can strive for composability, and the integrator of Smart Systems can benefit for compositionality.

IV. CHESS EXTENSIONS

The CHESS methodology and toolset have been further developed and extended in several EU and ESA funded R&D projects to better cover different aspects and needs in the IoT systems' development lifecycle.

The CONCERTO [12] project is the logical continuation of the original CHESS project: it provides enhancements for the modeling and real-time analysis of multicore systems, support for partitioning, improvements to the code generation capabilities for distributed nodes and beyond Ada, as well as modeling and safety analysis solutions for addressing mixed criticality issues.

The FoReVer [13], SafeCer [14] and SESAMO [15] projects, on the other hand, provided important system-oriented extensions to the CHESS methodology and tool chain, mainly in the perspective of a systematic formalization of the process with a strong focus on early verification, safety and security aspects.

A. System level modeling

CHESS extended its concerns from concentrating mainly on the software level to embracing also the system level design phase.

During the whole modeling process the system is designed in terms of architectural components formally described with their well-defined interfaces and related properties. Such components are considered as black boxes until they are decomposed into their lower level components. System level entities are initially designed at a higher level of abstraction and then hierarchically decomposed applying a "divide et impera" approach in order to be able to focus separately on the different composing parts of the system. This way a deeper understanding of the parts can be achieved, allowing to add details to the parts, refining their models, and thus implementing an *incremental* process.

When system decomposition has been performed to a suitable extent, the system elements are linked in the model to the corresponding software level entities by means of the SysML «allocate» dependency: this way *system-software co-engineering* is implemented as a seamless process. Particular emphasis is dedicated in the CHESS approach to ensure adequate support to the system-software co-engineering phase, keeping consistency between system level entities and requirements on one side and the corresponding software and hardware level entities on the other side. This leads to an *iterative* development process: it is hardly ever a pure waterfall process and often software level decisions have an impact also on the higher system level, requiring an iteration loop (it is not infrequent that the system's decomposition itself must be adjusted based on the envisaged software organization).

B. *A contract-based approach*

Another interesting new perspective introduced in the CHESS extended methodology is that it follows a *contract-based* approach, an emerging approach to address the increasing complexity of industrial systems. Contracts can be used everywhere and at all stages of system design, from early requirements capture, to any level of system modeling and detailed design involving software and hardware. They can be considered formalizations of the conditions for correctness of element integration, for lower levels of abstraction to be consistent with the higher ones, and for abstractions of available components to be reused.

The introduction of a contract-based approach relies on the fact that the definition of functional interfaces does not provide sufficient information on a component for its reuse: assumptions on the environment where the component can be integrated are discriminant for its reusability, in particular for critical components, where non-functional properties such as safety and real-time constraints are crucial. This situation arises often in Smart IoT systems in the automotive, industrial automation, and medical areas, for example.

According to the CHESS extended methodology components' properties are formalized in terms of *contracts* and thus composed of an *assumption* and a *guarantee* modeled as formal properties, where the assumption is a constraint on the component's environment or usage, while the guarantee is a property that must be satisfied by the component - provided that the environment satisfies the assumption.

The whole development process envisaged in the CHESS extended methodology runs through different conceptual models: from the higher level *functional architecture* and *logical architecture* - with functions identified in the functional architecture *allocated* to the entities of the logical architecture - down to the lower level *physical architecture* - with specification of software and hardware entities and their deployment, i.e. allocation of software to hardware.

Passing from one architectural level to the next, requires a change in perspective and often implies also a change in the involved responsibility and professional profiles. Mapping leaf elements from the higher level architecture onto elements in the lower level architecture is a valid support to ensure coherence and continuity in the development process, allowing full *traceability* of design decisions across different design phases and thus highlighting the impact of design changes in different phases.

According to the CHESS extended methodology, within each conceptual level, the modeler applies a *step-wise refinement* process: alongside with the decomposition of a component into subcomponents, the component's contracts are *decomposed* into a collection of contracts over its subcomponents.

Step-wise refinement is subject to *formal verification* and is a key-point in the overall verification process as in [16], [17] according to the approach first envisaged in the ESA funded FoReVer study [13] and further elaborated within the SafeCer project [14]. The core of the formal verification reasoning can be summarized in the following statement: *if the refinement steps are proven correct by formal verification, any implementation of the leaf components that satisfies the components contracts can be used to implement the system satisfying the system's contracts*. This provides a powerful aid in the model-based system development process and strongly encourages the use of standard qualified components.

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

The resulting CHESS extended methodology [17] is a component-based model-based incremental iterative development process that relies on the concept of contracts for the modeling and analysis of real-time and embedded systems where formal verification is carried out from the earliest development loops implementing an enhanced version of the traditional V-model development process as depicted in Fig. 2.

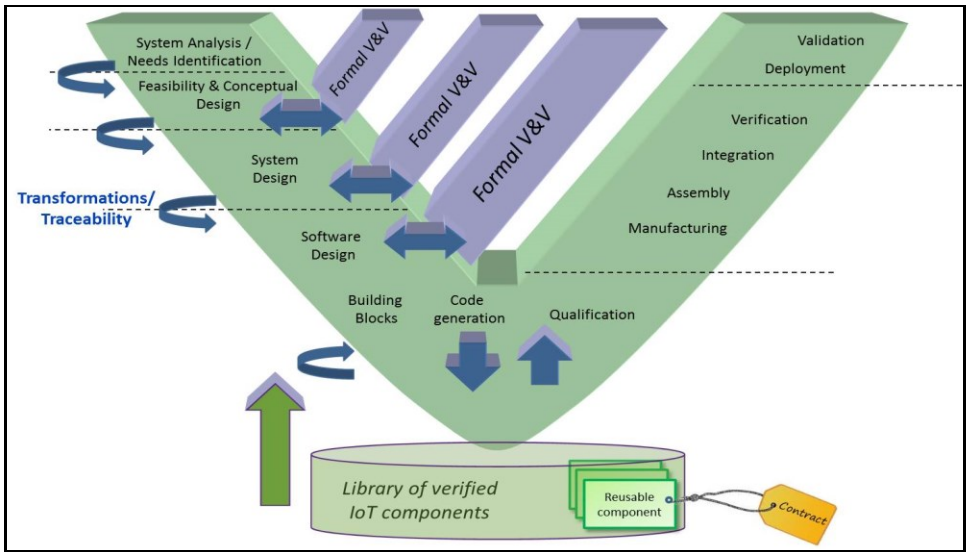

Fig. 2 The CHESS enhanced V-model development process

Indeed *formal analysis techniques*, applied *early* in the development process, have the potential to reduce the overall development, verification and maintenance costs, to increase the effectiveness of the engineering of the system, increase the level of consistency of models and the level of automation in the system lifecycle.

Support for modeling contracts and for step-wise refinement, is provided in the extended CHESS toolset. Step-wise refinement is subject to formal verification performed by OCRA (Othello Contracts Refinement Analysis) [18] by Fondazione Bruno Kessler, a tool built on top of NuSMV3 for the verification of logic-based contracts refinement for embedded systems [19], which is integrated[3] in the CHESS extension.

C. *Modeling requirements*

Another important aspect of the CHESS extended methodology is related to the role and modeling of *requirements.* Requirements have a central role in the CHESS extended methodology, being an integral part of the model, together with their associated traceability links to other model items, right from the initial development phases, according to the MBSE approach. Requirements are supposed to be written following appropriate domain specific guidelines, using suitable glossaries of terms, and if possible relying on a semantic wiki [20].

Integration of requirements in the model is brought to a further extent, thanks to the *formalization of requirements*: an approach that was thoroughly explored in the MARVELS project [21], but not yet integrated in CHESS. Requirements are formalized by writing them to match a predefined grammar and linking their constraints to the corresponding model elements and related properties: this allows verification activities to be performed directly on the model.

[3] integration with OCRA is implemented via model transformations developed by Intecs, applied on the System View and on the PIM.

Formalization of requirements implies not only the formalization of the constraints, but also the formalization of the related assumptions (e.g. environmental conditions, operational activities, durations, or boundary conditions in general). Smart IoT systems are generally extremely sensitive to environmental conditions because of the deployment of sensors "close to the ground." The formalization of such requirements brings out inconsistent and missing assumptions on these conditions.

Constraints and assumptions are then associated with the verification activities that shall verify them on a specific model component. Once the verification is successfully performed, the verified constraints, according to specific assumptions, are considered as guarantees, justified by the performed verification as illustrated in Fig. 3. The qualified or accepted element is therefore associated with a "contract" which is the collection of guarantees and assumptions (as defined in the FoReVer study), together with the performed verification activities and form a basis for reuse of system components and their models [21].

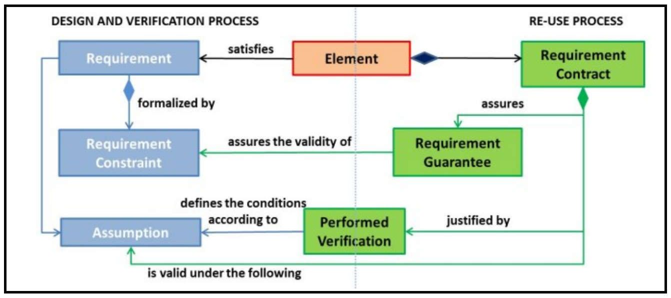

Fig. 3 Requirements, Architectural Elements and their reuse

V. CHESS IN THE AUTOMOTIVE DOMAIN

CHESS has been extended to address specific safety and automotive aspects in the frame of the SafeCer [14] and the SESAMO [15] project.

In the frame of the SESAMO project Intecs integrated CHESS with *medini analyze* [22], an industrial toolset by KPIT supporting the safety analysis and design for software controlled safety related functions in cars as prescribed by the ISO 26262 automotive safety standard. System models defined and detailed in medini analyze, associated with safety goals/requirements characterized by their related ASILs, can be imported (via automated model-to-model transformations) [23] to the CHESS Requirements and System Views, where it is possible to continue the modeling process designing the software components, associating them to the system level components and performing real-time and dependability analysis. The modeling and analysis process in CHESS may result in the definition of new requirements modeled in the CHESS environment, which can then be imported back into medini analyze. This opens the door to accommodating the growing set of Smart Services in the automotive industry, particularly in the area of active safety functions that are gradually introducing increasing degrees of autonomy that will require reliable, verifiable, and qualifiable analyses before being allowed on the road.

An important enhancement to CHESS aiming to support the AUTOSAR standard (with its ever-increasing integration of capabilities for accommodating smart functionalities) and development flow is currently being carried out in the CONCERTO project [12].

CHESS is now being applied to issues of verifying data distribution systems in Intelligent Transport Systems, with their scalable and heterogeneous deployments that introduce uncertainties that can be well addressed with the modeling capabilities and tool support of CHESS.

VI. CONCLUSION

CHESS offers a holistic solution to the development of the IoT, providing an approach to tackle several problems such as emergent behavior.

Relying on models from the earliest phases of requirements definition and high-level system design, down to the actual component's implementation, configuration, run-time operation, management and monitoring, allows to achieve simultaneously a meaningful global vision of the system as a whole and a clear view of the interfaces among the composing parts.

ACKNOWLEDGMENT

This work has been partially supported by the ARTEMIS-JU projects: CONCERTO [12], CHESS [1], SafeCer [14] and SESAMO [15] and by the ESA funded projects FoReVer [9] and MARVELS [21].

REFERENCES

[1] CHESS project: "Composition with guarantees for high-integrity embedded software components assembly", [Online], Available: http://www.chess-project.org/ [Accessed: May 5, 2015].

[2] CHESS PolarSys project, [Online], Available: https://www.polarsys.org/projects/polarsys.chess [Accessed: May 5, 2015].

[3] 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.

[4] D. C. Schmidt, (2006). Guest Editor's Introduction: "Model-Driven Engineering", *Computer*, vol. 39, no. 2, 25–31, February 2006. [Online], Available: http://dx.doi.org/10.1109/MC.2006.58 [Accessed: May 14, 2015].

[5] R. France and B. Rumpe, "Model-driven Development of Complex Software: A Research Roadmap", in Future of Software Engineering, 2007, FOSE '07, IEEE Computer Society, 2007, pp. 37–54, [Online], Available: http://dx.doi.org/10.1109/FOSE.2007.14 [Accessed: May 14, 2015].

[6] MAST: "Modeling and Analysis Suite for Real-Time Applications", [Online], Available: http://mast.unican.es/ [Accessed: May 5, 2015].

[7] DEEM: "DEpendability Modeling and Evaluation of Multiple Phased Systems", [Online], Available: http://rcl.dsi.unifi.it/projects/tools [Accessed: May 14, 2015].

[8] B. Gallina and E. Sefer, "Towards Safety Risk Assessment of Socio-technical Systems via Failure Logic Analysis" submitted to RISK 2014.

[9] CHESS Consortium, "CHESS Modeling Language and Editor" V1.0.2, Project Deliverable, 31 March 2010.

[10] 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.

[11] 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.

[12] 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 May 5, 2015].

[13] 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: May 5, 2015].

[14] SafeCer project: "Safety Certification of Software-Intensive Systems with Reusable Components", [Online], Available: http://safecer.eu/ [Accessed: May 5, 2015].

[15] SESAMO project: "Security and Safety Modeling", [Online], Available: http://sesamo-project.eu/ [Accessed May 5, 2015].

[16] 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.

[17] L. Baracchi, A. Cimatti, G. Garcia, S. Mazzini, S. Puri and S. Tonetta, "Requirements refinement and component reuse: the FoReVer contract-based approach", in A. Bagnato, I. R. Quadri, M. Rossi and l. S. Indrusiak, Editors Industry and Research Perspectives on Embedded System Design, IGI Global, Hershey PA, USA 2014.

[18] 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: May 11, 2015].

[19] A. Cimatti and S. Tonetta, "A Property-Based Proof System for Contract-Based Design". EUROMICRO-SEAA 2012: 21-28.

[20] S. Mazzini, J. Favaro, R. Schreiner, and H.P. de Koning, "Next Generation Requirements Engineering", Proceedings of INCOSE International Conference, Roma, July 2012.

[21] M. Pasquinelli, D. Gerbaz, J. Fuchs, V. Basso, S. Mazzini, L. Baracchi, S. Puri, L. Pace, M. Lassalle, and J. Viitaniemi, "Model-based approach for the verification enhancement across the lifecycle of a space system", CIISE Conference, Rome, November 2014.

[22] Medini analyze product presentation, [Online], Available: www.ikv.de/medinianalyze [Accessed: May 5, 2015].

[23] S. Mazzini, J. Favaro, A. Martelli, and L. Baracchi "Security and Safety Modelling in Embedded Systems" ERTS Conference, Toulouse, Feb 2014.