# AutoFOCUS 3: Tooling Concepts for Seamless, Model-based Development of Embedded Systems

Vincent Aravantinos, Sebastian Voss, Sabine Teufl, Florian Hölzl, Bernhard Schätz
fortiss GmbH
Guerickestr. 25
80805 Munich, Germany
Email: hoelzl@fortiss.org

***Abstract*—This paper presents tooling concepts in AUTOFOCUS 3 supporting the development of software-intensive embedded system design. AUTOFOCUS 3 is a highly integrated model-based tool covering the complete development process from requirements elicitation, deployment, the modelling of the hardware platform to code generation. This is achieved thanks to precise static *and* dynamic semantics based on the FOCUS theory [1]. Models are used for requirements, for the software architecture (SA), for the hardware platform *and for relations* between those different viewpoints: traces from requirements to the SA, refinements between SAs, and deployments of the SA to the platform. This holistic usage of models allows the provision of a wide range of analysis and synthesis techniques such as testing, model checking and deployment and scheduling generation.**

**In this paper, we demonstrate how tooling concepts on different steps in the development process look like, based on these integrated models and implemented in AUTOFOCUS 3.**

***Index Terms*—AutoFOCUS3, Seamless MBD, Model-Based Development, Embedded Systems, Tooling Concept, Tool**

## I. INTRODUCTION

Embedded systems are increasingly developed in a model-based fashion facilitating constructive and analytic quality assurance via abstract component-models of the system under development. A variety of different tools claim to support a model-based approach; however, these tools mostly cover certain parts of the development process. In this paper, we demonstrate the advantage of integrated models and provide tooling concepts for various design steps. Both together leverage the benefits of a seamless model-based approach based on well-defined semantics. The objective of this paper is to present such tooling concepts in an entire tool (AUTOFOCUS 3) in its current advanced feature state, which is open source and freely available at `http://af3.fortiss.org`. The main contribution of this paper is to demonstrate the seamless integration of the integrated models.

AUTOFOCUS 3 is built on a *system model* based on the FOCUS theory [1] that allows to precisely describe a system, its interface, behavior, and decomposition in component systems on different levels of abstraction. To manage the complexity of describing such a system, different *views* are used for these aspects, providing dedicated description techniques for different structural aspects (like the decomposition of a component in a network of subcomponents, or the hardware platform the system is implemented on) as well as behavioral aspects (like an exemplary execution of a system, or its complete behavior). Since all of these views are projections of the underlying system model, these views are directly integrated. Furthermore, linking views allow an additional integration (like the mapping of a component behavior to a hardware element, or a required partial execution to a completely defined behavior). Besides allowing a manageable precise description of a system under development, the system model also enables different analysis and synthesis mechanisms (like the compatibility analysis of a partial and complete behavior or the deployment synthesis of components to a hardware platform). To support the different tasks in a development process, the views are furthermore organized in *viewpoints*. A viewpoint serves as a construct for managing the artifacts related to the different stakeholders of the development process [2, Chapter 3]. The AUTOFOCUS 3 viewpoints focus on the definition of the system requirements in a requirements analysis, the design of the software as a network of communicating components in form of a software architecture, and the realization of the system as scheduled tasks executed on networked processors in form of a hardware architecture.

The importance of integrated *models*, *views* and *viewpoints* is widely recognized and influenced the definition of many methods and frameworks in systems, software and enterprise engineering [2], [3] and provides the basis for this paper.

The objective of AUTOFOCUS 3 is to implement *tooling concepts* which demonstrate that such an approach is indeed feasible through a ready-to-use, open-source implementation in a pre-series quality. The current AUTOFOCUS 3 is a revised and improved version of earlier prototypes [4], [5] (the oldest dating back to 1996). Previous papers either report on particular aspects of the tool [6], [7], [8], [9], or on its use in the context of industrial case studies [10], [11], [12]. The underlying ideas of the current AUTOFOCUS 3 incarnation are presented in [13].

AUTOFOCUS 3 is not tied to a specific development process, but most developments done with AUTOFOCUS 3 would typically follow the following process or variations thereof:

1) **Requirements Analysis.** Requirements are elicited, documented as structured text, organized, analyzed and refined, and incrementally formalized. Test suites with coverage criteria can be generated from high-level specifications.

2) **Software Architecture.** The system is designed with a component-based language specifying the application software architecture and behavior of the system. The design is validated using *simulation*, *testing* (which can come as refinements from high-level generated tests) as well as *formal verification* based on model-checking.
3) **Hardware Architecture.** The software components are (possibly automatically) deployed on the platform, w.r.t. certain system requirements. Schedules optimizing one or more criteria are generated with the help of *SMT solvers*.

The code is *completely* generated out of the previous models, according to the deployment and chosen schedule. Furthermore, **Safety Cases** [14], which are documented bodies of evidence that provide a convincing and valid argument that a system is adequately safe for a given application in a given environment, can be modelled. A Safety Case may contain complex arguments that can be decomposed, corresponding to modular system artifacts which are generally *dependent on artifacts from different viewpoints*.

The paper is organized as follows: Section II presents briefly the main modelling viewpoints that are offered by AUTOFOCUS 3 (requirements, software architecture, and hardware architecture). Section III presents the transversal viewpoints which facilitate to make the connections between the main viewpoints and thus yield a seamless integration. We also present the benefits resulting of this integration: formal analysis and verification, scheduling, hardware-specific code generation. Section IV presents related work.

## II. MODEL-BASED TOOLING CONCEPTS IN THE DEVELOPMENT PROCESS

In this section we present shortly the three modelling *viewpoints* supporting the process mentioned in the previous section.

### A. Requirements

In AUTOFOCUS 3, requirements are specified model-based: requirements are not just documented as plain text; the tool provides templates with named fields to define, for instance, the title of a requirement, its author, a description, a potential rationale or a review status (see Fig. 1).

Furthermore, requirement sources and glossaries can be defined. Whenever they are referenced in a textual description of a requirement, the entries are automatically highlighted and the definition can be read in a pop-up. Requirements can be hierarchically grouped by packages and organized by trace links. Templates for scenarios and interface behavior help to detail requirements further.

Requirements can not only be documented as text, but also formalized and represented by machine-processable models. Message sequence charts (MSC), see Fig. 2, can be used to describe desired or unwanted interactions of actors.

Temporal logic expressions can be used to express desired and unwanted behavior of the system under development.

**Requirement**

| | |
|---|---|
| ID | 1.2 |
| Type | Requirement |
| Title | Safety requirement: accidents prev |
| Description | Pedestrians and cars should NOT b |
| Rationale | Prevent traffic accidents. |
| Author | Dan |
| Source | System architect Christopher Pike |
| Document Reference | |
| Status | Analyzed |
| Priority | Normal - Satisfier |

Fig. 1. Example structured requirement

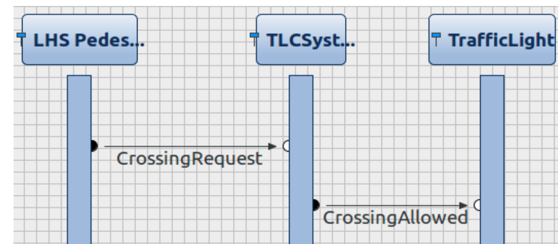


Fig. 2. Message sequence charts

As shown in Fig. 3, AUTOFOCUS 3 provides user-friendly templates (see also [15]) to specify temporal logic expressions.

Guarantees '!(ctrlOutPedestrianSignal == Walk( ) && ctrlOutTrafficSignal == Yellow( ))'.

Fig. 3. Temporal logic expression

Once the requirements analysis is sufficiently advanced, it is possible to express formalized behaviors, e.g., in the form of state automata. A state automaton typically covers a set of requirements rather than a single requirement.

**Statistics**

Number of contained elements

Requirements and use cases: 8
Requirements: 5
Use cases: 3

**Filter**

**Requirements Overview**

Overview list

| Type | Status | Name |
|---|---|---|
| Requirements package | - | 1 - High-level Requiren |
| Requirements package | - | 2 - Low-level Requirem |
| Requirement | In Analysis | 1.3 - Pedestrian light ti |
| Requirement | Analyzed | 1.2 - Safety requiremer |
| Requirement | Analyzed | 2.2 - Safety requiremer |
| Requirement | In Analysis | 2.3 - Pedestrian light ti |

Fig. 4. Requirements statistics and reports

**Tooling Support.** AUTOFOCUS 3 supports the user in *analyzing* the requirements, for example through reports on the review status or statistics (Fig. 4). Simple queries on the requirements identify for example empty fields, duplicates and inconsistent status of requirements and their trace links. A report can be generated from AUTOFOCUS 3 that can be used for the *validation* of the requirements by the stakeholders of the system under development. State automata can be simulated; this is typically used in both the analysis and validation of requirements.

### B. Software Architecture

The software architecture of a system under development can be described using a classical component-based language with a formal (execution) semantics based on the FOCUS theory [1]: components execute in parallel according to a global, synchronous, and discrete time clock.

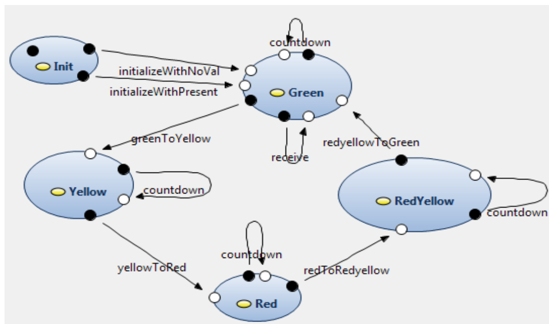


Fig. 5. State automaton

The behavior of atomic components can be defined by state automata (Fig. 5), tables (Fig. 6) or simple imperative code (Fig. 7). Components interact with each other through typed input and output ports which are connected by fixed channels (Fig. 8).

| | Input | | | Output |
|---|---|---|---|---|
| | mergeInButtonA | mergeInButtonB | true | ctrlInRequest |
| - | Present() | * | mergeInB...resent() | Pressed() |
| - | * | Present() | mergeInB...resent() | Pressed() |
| - | Present() | Present() | * | Pressed() |
| - | NoVal | NoVal | * | Released( ) |
| + | Click to add a new rule... | | | |

Fig. 6. Table

```
if (behaviorInState == 0) {
    behaviorOutTrafficSignal = Green( );
    behaviorOutPedestrianSignal = Stop( );
    behaviorOutIndicatorSignal = Off( );
    behaviorOutTime = -1;
    behaviorOutState = 1;
    return;
}
```

Fig. 7. Simple imperative code

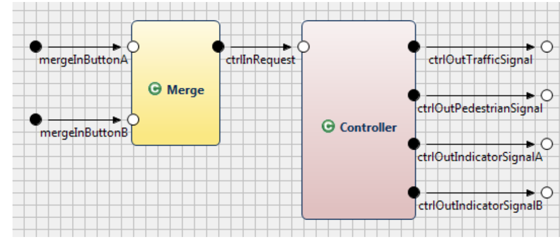


Fig. 8. Components and channels

Finally, components can be decomposed into sub-components to allow a hierarchically structured architecture.

**Tooling Support.** Due to the executable formal semantics of the component-based modeling language, AUTOFOCUS 3 facilitates the simulation of the software architecture at all levels, of a single state automaton (Fig. 9) as well as of composite components (Fig. 10) providing Model-in-the-Loop simulations. Test cases can be created and simulated (Fig. 11).

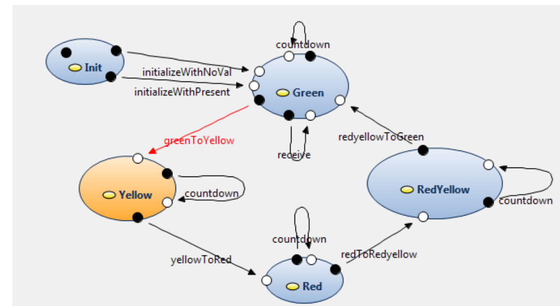


Fig. 9. Simulation of state automata

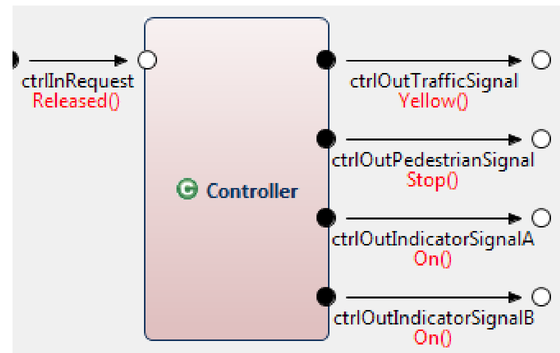


Fig. 10. Simulation of composite components

Furthermore, formal analyses like reachability analysis (see Fig. 12), non-determinism check, variable bounds checking and the verification of temporal logic patterns (see Fig. 13) are available: due to the formal semantics of FOCUS, AUTO-FOCUS 3 can embed a precise translation of the software architecture into the language of the NuSMV/nuXmv [16] model checker. Note that this is all done in a complete transparent way: the translation and call to the model checker are all done in the background to provide user-friendliness

| | LHSButton? | RHSButton? | ctrlOutTrafficSignal! |
|---|---|---|---|
| Test Case 0 | | | |
| Step 0 | NoVal | NoVal | NoVal |
| Step 1 | NoVal | Present( ) | NoVal |
| Step 2 | NoVal | NoVal | Green( ) |
| Step 3 | NoVal | NoVal | NoVal |
| Step 4 | NoVal | NoVal | NoVal |
| Step 5 | NoVal | NoVal | Yellow( ) |
| Step 6 | NoVal | NoVal | NoVal |

Fig. 11. Test suites

workflow. The results of the model checker, namely their counterexamples, are also translated back in the same way e.g., a counter example to a temporal logic property can be graphically simulated.

| Name | yellowToRed |
|---|---|
| Guard | time == 0 && time != 0 |

Guard on transition RedYellow to Red

States unreachable in 25 steps found!

| Unreachable States |
|---|
| Behavior.RootState.RedYellow |
| Behavior.RootState.Red |

Analysis result

Fig. 12. Unreachable state

Guarantees '!(ctrlOutPedestrianSignal == Walk( ) && ctrlOutTrafficSignal == Yellow( ))'. TLCSystem Fri Jul 10 15:11:... SUCCESS

Fig. 13. Verification result

### C. Hardware Architecture

AUTOFOCUS 3 allows to model the hardware: processors (or, in the automotive domain, ECUs – Engine Control Units), buses, actuators and sensors can explicitly be modeled as well as their connections (Fig. 14). Multi-core platforms with

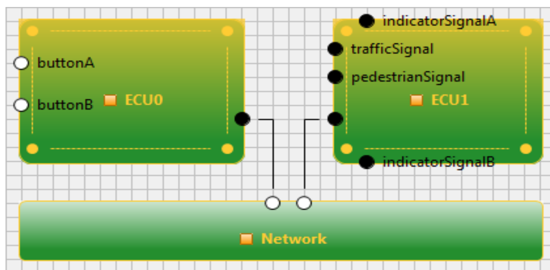


Fig. 14. Hardware architecture for generic ECUs

shared memory are available (Fig. 15), as well as specific *domain specific* hardware e.g., a *pacemaker* platform was built specifically for building and deploying models of a pacemaker. Similarly, automotive-specific hardware is supported via FIBEX import/export (an XML-based standardized format used for representing TDMA-networks).

Hardware architecture models actually deal with more than just hardware: they typically include a *platform* architecture

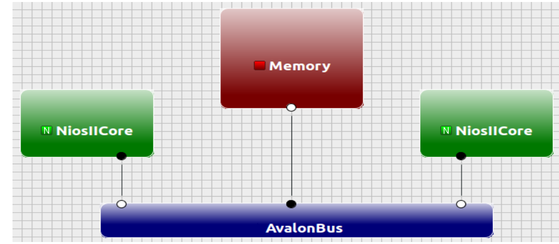


Fig. 15. Hardware architecture for multicore, with shared memory

which encompasses execution environments from bare metal hardware (e.g., chips and wires) over operating system environments up to higher-level runtime environments (e.g., Java virtual machine with remote method invocation mechanism). For instance, the aforementioned hardware model for FIBEX comes also with an implementation for the Flexray protocol ([17]), a time-triggered communication protocol in the automotive domain.

## III. SEAMLESS INTEGRATION

An integration of all of the previously mentioned viewpoints in an integrated model, resp. tooling environment is an important asset for the user: it avoids the effort to integrate tools, defining their interfaces and dealing with conflicting, missing, or badly documented tool-interfaces, semantics and standards. However, if these viewpoints remain completely independent or *if the dependency between them remains informal* as it is the case with most existing tools, then only very few benefits result from its integration. Instead, AUTOFOCUS 3 allows for model integration leading to a consistent, integrated system design. In the following, we present these models as well as analysis and synthesis techniques that demonstrate the benefits of AUTOFOCUS 3 resulting from this strong integration.

### A. Tracing Requirements to the Software Architecture

**Traces.** The integration of requirements (Section II-A) and the software architecture (Section II-B) can be achieved in various ways. A first simple integration is the use of informal traces: for each requirement, traces to components of the software architecture can be added (see Fig. 16). These traces

Traces to architecture

| | Status | Author | ---> | Target |
|---|---|---|---|---|
| | New | | | Behavior |

Fig. 16. Traces to the software architecture

indicate that the component(s) linked to the requirement shall fulfill the requirement. Such traces are automatically visible (and navigable) at the level of the component architecture (Fig. 17). These traces can be used to display information to the user. For example, a global visualization of the traces, both between requirements and between requirements and elements of the software architecture, is available and allows the user

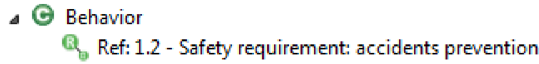


Fig. 17. Traces, seen *from* the software architecture

to have an overall picture of the intra- and inter-viewpoint relations (Fig. 18).

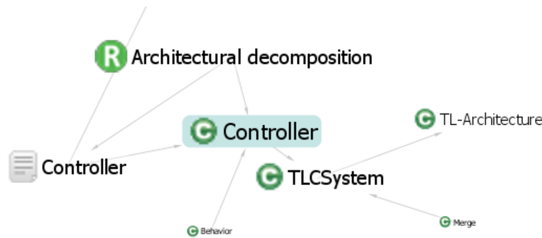


Fig. 18. Traces visualization

**Refinements.** Traces provide informal connections between model elements, which is to be expected since most requirements are (at least at first, or partly) informal. However, as explained in Section II-A, requirements elicitation can go far enough that a formalized behavior is obtained, e.g., that a state automaton is given. In such cases, traces to the software architecture can be enhanced into *refinements* describing not only a mere connection, but even expressing a formal relation between the requirements-level behavior description and a software-level implementation of it. A refinement is simply defined by expressing how values at the requirement level shall be transformed into values at the software level and vice versa (Fig. 19). Such refinements can then be used

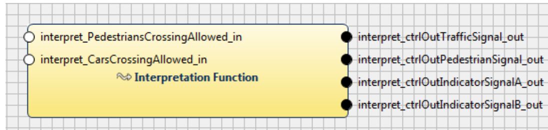


Fig. 19. Refinement definition

to automatically derive implementation-level test suites from requirements-level ones [18], [19] (which can be automatically generated according to some coverage criteria) or to verify by model checking that a component indeed implements a functionality.

**Connecting MSCs to the Software Architecture.** MSCs can be used in requirements in a semi-formal setting, i.e., the MSC entities represent actors identified in the requirements. Or they can be used in a completely formal setting: when the requirement elicitation is advanced enough, MSC entities can refer to components and the messages between entities can denote signals exchanged through channels. This can be expressed directly in the MSC editor, e.g., Fig. 20 shows how the properties of an MSC entity named "Merge Entity" refers to a component named "MergeComponent". The same

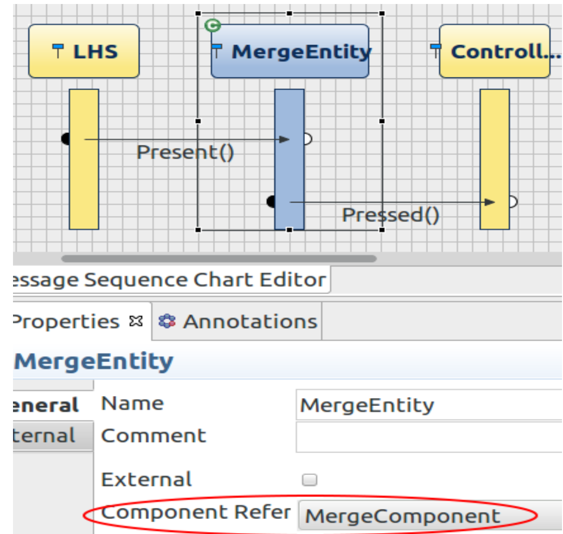


Fig. 20. Connecting MSCs to components

holds for the messages which can be connected to ports of the corresponding components.

Once these connections are provided, AUTOFOCUS 3 allows to verify, by translating the MSC into a temporal logic formula and by using model checking (in the style of [20]), that the given MSC is indeed feasible in the software architecture with the referenced components and ports. When feasible, the resulting run can be simulated in AUTOFOCUS 3.

### B. Deployment of the Software Architecture on the Platform

**Deployment of Software to Hardware Components.** The integration of the software (Section II-B) and hardware architecture (Section II-C) is done by using deployment *models* that describe the integration between different viewpoints. Such deployments map components from the software viewpoint to the hardware viewpoint. Furthermore, the deployment model not only contains the mapping of components but also the allocation of logical ports of a software component to their corresponding hardware "ports", namely hardware *sensors* for sensor values and hardware *actuators* for actuator outputs. Fig. 21 illustrates this deployment. Note that the hierarchical structure of components makes it impossible to have such a simple GUI for deployments in AUTOFOCUS 3, making the actual interface different than the one presented in this figure.

**Design Space Exploration for Model Synthesis.** Once a deployment is defined, Design Space Exploration methods can be applied to define the *scheduling* of the software components on their respective hardware components. AUTOFOCUS 3 provides support to achieve this step by *automated synthesis*. Actually, even deployments can be automatically synthesized as well as complete hardware architectures, according to various system requirements, e.g. timing, safety, etc. To do so we use Design Space Exploration (DSE) techniques. In [21], we demonstrate how such a joint generation of deployments and schedules can be efficiently done for shared-memory multicore architectures. Each solution of such a synthesis process already

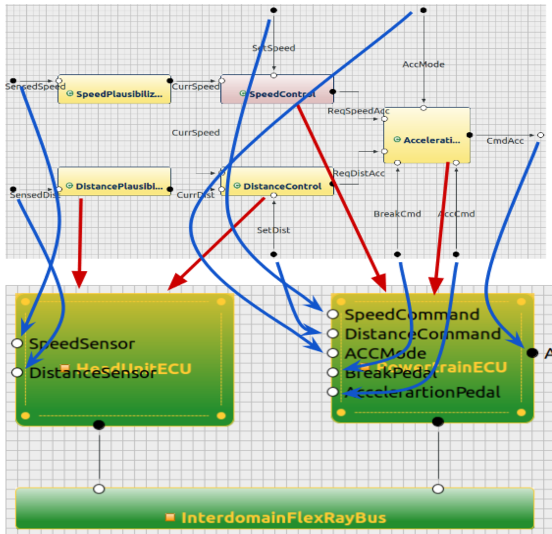


Fig. 21. Deployment (illustration)

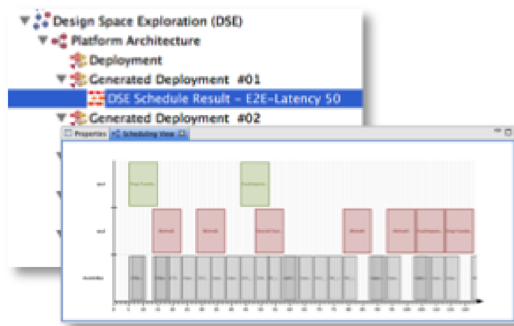


Fig. 22. Design Space Exploration results

contains a possible deployment, which in turn already contains a valid schedule (cf. Fig. **??**). This reduces the effort and complexity in a the workflow for the identification of valid system designs.

Our approach relies on a symbolic encoding scheme, which enables to generate the desired models. The symbolic encoding is done by defining a precedence graph of components based on the software architecture as a set of tasks and messages and their connections including further information concerning predefined allocations to hardware architecture.

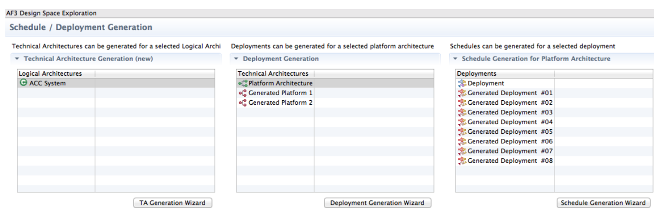


Fig. 23. Design Space Exploration Workflow

The proposed approach has proven to perform in a scalable fashion for practical sizes ([21]), as it relies on a symbolic formalization encoding the deployment synthesis as a satisfiability problem over Boolean formulas and linear arithmetic constraints. A state-of-the-art *satisfiability modulo theories (SMT)* solver, namely Z3 [22], is used to compute these solutions. Using Design Space Exploration techniques during system development involves the software engineer/designer itself. The system designer is often not just interested in an automatically synthesized solution, but even more in various solutions that can be compared. Therefore, visualization techniques [23] are part of a Design Space Exploration approach that leverages to guide the system designer through the solution space.

Furthermore, we propose a tooling concept that includes a Design Space Exploration Workflow (Fig. 23) enabling to use intermediate results for next optimization steps, e.g. a *Generated Deployment* or a *Scheduling Synthesis*.

### *C. Holistic Code Synthesis for Deployed Systems*

Once the software architecture, the platform architecture, and a (manually defined or automatically synthesized) deployment model are defined, AUTOFOCUS 3 provides the possibility to have holistic code generation.

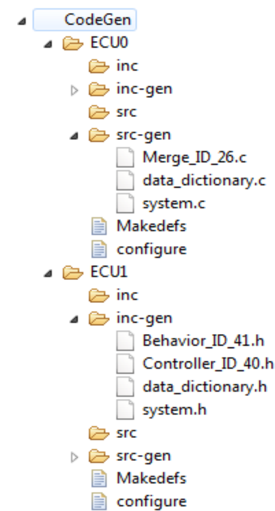


Fig. 24. Generated C code for the deployed system

The input to the generation facility is the mapping of software components to platform execution units. The result of the code generator is a full implementation of the system model including configuration files for the underlying operating system as well as bus message catalogs and compile and build environment configurations (see Fig. 24).

The code generator consists of two parts: the software architecture general purpose generator and the platform-specific target generator. The former translates the different types of software behavior specifications into an intermediate code model. From this intermediate representation the final system code (see Fig. 25) is generated by the ECU specific code generator using the ECU target language (e.g. C, Java, VHDL).

Note that these specific generators can ignore the intermediate implementation in cases the original behavior specification

```
void run_system(){
    read_input( );
    prepare_output_net( );
    perform_step_Merge_ID_26( );
    if (noval_mergeOutRequest_ID_38 == true) {
        set_noval_net_mergeOutRequest( );
    }
    else {
        write_net_mergeOutRequest(mergeOutRequest_ID_38);
    }
    finish_output_net( );
}
```

Fig. 25. Main loop of the ECU running the Merge component

can be implemented more efficiently when applying a transformation to the target language and/or hardware directly (e.g., a state-less computation component might be implemented efficiently on a FPGA sub-unit available to the ECU).

Every platform integrated in AUTOFOCUS 3 must provide its extension to the target generator as well as a justification that it upholds the semantics of the model of computation of the software architecture. Likewise, the software code parts must also be behaviorally equivalent to these formal semantics. Proving such semantic equivalences can be cumbersome [24], but is absolutely necessary in order to avoid breaking functional properties established by earlier validation and verification methods.

### D. *Safety Cases*

To argue about the safety of systems, *Safety Cases* are a proven technique that allows a systematic argumentation. Safety Cases may contain complex arguments that can be decomposed corresponding to modular system artifacts which are generally *dependent on artifacts from different viewpoints*: e.g., requiring redundancy for safety has an impact both on software *and* on hardware architectures. Such assurance cases

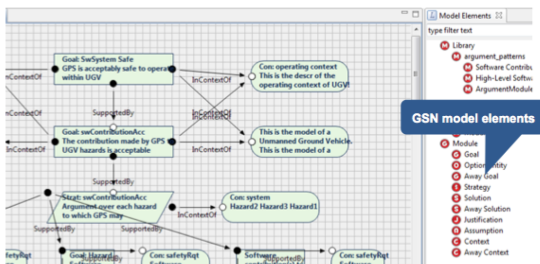


Fig. 26. GSN-based Safety Cases

are generally not well integrated with the different system models, resp. viewpoints. To provide a comprehensible and reproducible argumentation and evidence for argument correctness, we make use of the integrated system model. Since AUTOFOCUS 3 provides such integrated models at its core, it leverages the possibility to tightly connect these system models with safety case artefacts in order to form a comprehensive safety argumentation. In AUTOFOCUS 3 we provide safety case modelling based on Goal Structuring Notation (GSN) [25], as illustrated Fig. 26. Different safety case artifacts can be connected to their corresponding system artifacts (e.g., a safety goal to a requirement from the requirement viewpoint). This – for instance – enables to automatically guide the construction of the system architecture w.r.t. the safety claims, as we demonstrated in [26].

## IV. RELATED WORK

There are many model-based tools which target the development/architecting of embedded systems, but none of them, to our knowledge, presents all the features of AUTOFOCUS 3.

Papyrus [27] with Moka[1] allows the execution of models based on the fUML [28] Semantics. Code generation is, as far as we know, only partly implemented, but considering the fast growth of Papyrus and Moka, this is should only be a question of time. A more significant difference to AUTOFOCUS 3 is that AUTOFOCUS 3 integrates all the modules into a unified software instead of being made of separate modules for diagram editing (Papyrus) and execution semantics (Moka). This has a significant impact on the verification (either testing or formal verification): in AUTOFOCUS 3, the semantics for execution and verification are intrinsically identical; in Papyrus additional work is required to synchronize the semantics of Moka and the verification tool, for example Diversity[2] – a verification tool typically used together with Papyrus.

The widespread commercial tool IBM Rational Rhapsody[3] has been offering for a long time a complete tool chain until code generation. Rhapsody has a precisely defined semantics [29]. It has even been used as a basis to provide integrated formal verification [30]. However, it is not as tightly integrated as AUTOFOCUS 3, not open source and is essentially used for commercial use and not as a platform for research experiments as AUTOFOCUS 3. The design space exploration viewpoint of AUTOFOCUS 3 is a research tooling concept which is a good example of such an experiment which differentiates AUTOFOCUS 3 from Rhapsody. Similar considerations hold for Bridgepoint (or xtUML)[4], LieberLieber Embedded Engineer[5] and the Enterprise Architect (EA)[6] that supports many modeling languages such as UML or SysML. ADORA (Analysis and Description of Requirements and Architecture)[7] is a research tool that supports an object-oriented method and a modeling language also called ADORA [31]. ADORA targets requirements and the software architecture of a system. Hardware is not included.

Ptolemy II [32] is similar to AUTOFOCUS 3 in the sense that it is based on formal semantics and provides code generation. Like AUTOFOCUS 3, it is an open source and academic tool which is used for research. However, Ptolemy targets only the software architecture: neither requirements nor the hardware are integrated. This arises from the fact that the

[1]https://wiki.eclipse.org/Papyrus/UserGuide/ModelExecution
[2]http://projects.eclipse.org/proposals/diversity
[3]www-01.ibm.com/software/awdtools/rhapsody/
[4]https://xtuml.org/
[5]http://www.lieberlieber.com/en/embedded-engineer-for-enterprise-architect/
[6]http://www.sparxsystems.com/products/ea/index.html
[7]http://www.ifi.uzh.ch/rerg/research/adora.html

development of embedded systems is not the main focus of Ptolemy.

The SCADE Suite[8] is a commercial tool well-known in the development of control software, for example in avionics. While the SCADE Suite[9] offers a lot of functionality with respect of simulation, verification and code generation, at the moment it does not provide any support for requirements.

## V. Conclusion

In this paper, we presented AutoFOCUS 3 and the tooling concepts that it supports at different steps in the development process. AutoFOCUS 3 is based on a completely integrated model-based development approach from requirements elicitation to deployment on the platform of code which allows to generate code *completely* (i.e., without further human modification) from the models. Based on well-defined semantics, AutoFOCUS 3 demonstrates how integrated models are enablers for a wide range of analysis and synthesis techniques such as testing, model checking and deployment and scheduling synthesis. Tooling concepts in AutoFOCUS 3 demonstrate how to make use of these techniques in a model-based development process.

## References


[1] M. Broy and K. Stølen, *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. Springer, 2001.

[2] K. Pohl, H. Hönninger, R. Achatz, and M. Broy, *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*. Springer Publishing Company, Incorporated, 2012.

[3] U.S. Office of Management and Budget and U.S. Office of E-Government and IT, "A Common Approach to Federal Enterprise Architecture."

[4] F. Huber, B. Schätz, A. Schmidt, and K. Spies, "AutoFocus - A Tool for Distributed Systems Specification," in *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT)*, ser. LNCS, vol. 1135. Springer Verlag, 1996, pp. 467–470.

[5] F. Hölzl and M. Feilkas, "Autofocus 3: A scientific tool prototype for model-based development of component-based, reactive, distributed systems," in *Proceedings of the 2007 International Dagstuhl Conference on Model-based Engineering of Embedded Real-time Systems*, ser. MBEERTS'07. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 317–322.

[6] S. Teufl, D. Mou, and D. Ratiu, "MIRA: A tooling-framework to experiment with model-based requirements engineering," in *21st IEEE International Requirements Engineering Conference, RE*, Rio de Janeiro-RJ, Brazil, 2013, pp. 330–331.

[7] A. Campetelli, F. Hölzl, and P. Neubeck, "User-friendly model checking integration in model-based development," in *24th International Conference on Computer Applications in Industry and Engineering (CAINE)*, November 2011.

[8] S. Voss, J. Eder, and F. Hölzl, "Design space exploration and its visualization in AUTOFOCUS3," in *Gemeinsamer Tagungsband der Workshops der Tagung Software Engineering*, Kiel, Deutschland, 25.-26. Februar 2014, pp. 57–66.

[9] T. Kelly, C. Carlan, and S. Voss, "Model-based safety cases in autofocus3," in *1st International Workshop on Assurance Cases for Software-intensive Systems (ASSURE)*, 2013, tool demonstration.

[10] M. Feilkas, A. Fleischmann, F. Hölzl, C. Pfaller, K. Scheidemann, M. Spichkova, and D. Trachtenherz, "A top-down methodology for the development of automotive software," Technische Universität München, Tech. Rep. TUM-I0902, January 2009.


[8]http://www.esterel-technologies.com/products/scade-suite/

[9]http://www.esterel-technologies.com/products/scade-suite/


[11] M. Feilkas, F. Hölzl, C. Pfaller, S. Rittmann, B. Schätz, W. Schwitzer, W. Sitou, M. Spichkova, and D. Trachtenherz, "A Refined Top-Down Methodology for the Development of Automotive Software Systems: The KeylessEntry System Case Study," Technische Universität München, Tech. Rep. TUM-I1103, Februar 2011.

[12] W. Böhm, M. Junker, A. Vogelsang, S. Teufl, R. Pinger, and K. Rahn, "A formal systems engineering approach in practice: an experience report," in *1st International Workshop on Software Engineering Research and Industrial Practices, SER&IPs*, Hyderabad, India, June 2014, pp. 34–41.

[13] B. Bernhard Schätz, "Model-based development of software systems: From models to tools." Habilitation Thesis, Technische Universität München, 2009. [Online]. Available: http://www4.in.tum.de/ schaetz/papers/Habiliationsschrift.pdf

[14] P. Bishop and R. Bloomfield, "A methodology for safety case development," in *Safety-Critical Systems Symposium*. Birmingham, UK: Springer-Verlag, ISBN 3-540-76189-6, Feb 1998.

[15] M. Dwyer, G. Avrunin, and J. Corbett, "Patterns in property specifications for finite-state verification," in *ICSE*, 1999.

[16] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta, "The nuxmv symbolic model checker," in *Computer Aided Verification*. Springer International Publishing, 2014, pp. 334–342.

[17] F. Consortium, "Flexray communications system protocol specification, version 2.1, revision A," URL http://www.flexray.com, 2005.

[18] D. Mou and D. Ratiu, "Binding requirements and component architecture by using model-based test-driven development," in *Twin Peaks of Requirements and Architecture (Twin Peaks)*, 2012.

[19] J. O. Blech, D. Mou, and D. Ratiu, "Reusing test-cases on different levels of abstraction in a model based development tool," in *MBT*, 2012, pp. 13–27.

[20] S. Li, S. Balaguer, A. David, K. Larsen, B. Nielsen, and S. Pusinskas, "Scenario-based verification of real-time systems using uppaal," *Formal Methods in System Design*, vol. 37, no. 2-3, pp. 200–264, 2010.

[21] S. Voss and B. Schätz, "Scheduling shared memory multicore architectures in AF3 using Satisfiability Modulo Theories," in *MBEES*, 2012, pp. 49–56.

[22] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340, 2008.

[23] S. Voss, J. Eder, and F. Hölzl, "Design space exploration and its visualization in AUTOFOCUS3," in *Gemeinsamer Tagungsband der Workshops der Tagung Software Engineering*, Kiel, Deutschland, 25.-26. Februar 2014 2014, pp. 57–66. [Online]. Available: http://ceur-ws.org/Vol-1129/paper33.pdf

[24] F. Hölzl, "The AutoFocus 3 C0 Code Generator," Technische Universität München, Tech. Rep. TUM-I0918, 2009.

[25] T. Kelly and R. Weaver, "The goal structuring notation – a safety argument notation," in *Proc. of Dependable Systems and Networks 2004 Workshop on Assurance Cases*, 2004.

[26] S. Voss, C. Cârlan, B. Schätz, and T. Kelly, "Safety case driven model-based systems construction," in *EITEC*, 2015.

[27] S. Gérard, C. Dumoulin, P. Tessier, and B. Selic, "Papyrus: A UML2 tool for domain-specific language modeling," in *Model-Based Engineering of Embedded Real-Time Systems - International Dagstuhl Workshop*, Dagstuhl Castle, Germany, November 4-9 2007, pp. 361–368, revised Selected Papers.

[28] T. O. M. Group, *Semantics of a Foundational Subset for Executable UML Models (FUML)*. Pearson Higher Education, 2013. [Online]. Available: http://www.omg.org/spec/FUML/1.1

[29] D. Harel and H. Kugler, "The rhapsody semantics of statecharts (or, on the executable core of the uml)," in *Integration of Software Specification Techniques for Applications in Engineering*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, vol. 3147, pp. 325–354.

[30] I. Schinz, T. Toben, C. Mrugalla, and B. Westphal, "The rhapsody uml verification environment," in *Proceedings of the Software Engineering and Formal Methods, Second International Conference*, ser. SEFM. Washington, DC, USA: IEEE Computer Society, 2004, pp. 174–183.

[31] M. Glinz, S. Berner, and S. Joos, "Object-oriented modeling with adora," *Inf. Syst.*, vol. 27, no. 6, pp. 425–444, Sep. 2002.

[32] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. R. Sachs, and Y. Xiong, "Taming heterogeneity - the ptolemy approach," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 127–144, 2003.