From Models to Code and Back: Correct-by-Construction Code from UML and ALF

Federico Ciccozzi
MRTC - Mälardalen University, Västerås (Sweden)
federico.ciccozzi@mdh.se

Abstract—Ever increasing complexity of modern software systems demands new powerful development mechanisms. Model-driven engineering (MDE) can ease the development process through problem abstraction and automated code generation from models. In order for MDE solutions to be trusted, such generation should preserve the system’s properties defined at modelling level, both functional and extra-functional, all the way down to the target code. The outcome of our research is an approach that aids the preservation of system’s properties in MDE of embedded systems. More specifically, we provide generation of full source code from design models defined using the CHESS-ML, monitoring of selected extra-functional properties at code level, and back-propagation of observed values to design models. The approach is validated against industrial case-studies in the telecommunications applicative domain.

I. PROBLEM AND MOTIVATION

The intricacy of complex software systems demands proper development mechanisms able to effectively deal with it. Towards this purpose, Model-Driven Engineering (MDE) aims to (i) aid system development by exploiting models that provide abstractions of real phenomena [1], and (ii) provide automation in the process by means of model transformations [2].

A major goal of MDE is the provision of automated code generation from design models; however, this goal is too often seen as the very final step of an MDE approach [3]. On the one hand, preservation of both functional (FP) and extra-functional properties (EFP) throughout the development process by means of appropriate description and verification is of paramount importance since it allows to reduce final product verification and validation effort and costs by providing correctness-by-construction. Note that for correctness-by-construction it is intended the adherence of generated code to related design models both functionally as well as extra-functionally, once the generation process has been validated [4]. On the other hand, certain EFPs (e.g., "worst-case execution time" for a code-block to be executed on a modern hardware platform with accelerator features) specified at modelling level cannot be accurately determined without code execution due to, e.g., resources limitation, common characteristic of embedded systems [5]. That is the reason for which these properties need to be measured at code level through monitoring and analysis activities [6].

In order to evaluate expected against observed EFP values, the abstraction gap between models and code must be bridged; this can be achieved by propagating the EFP values, observed by monitoring the execution of generated code, back to the design models. Hence, we claim that the generation of code and its execution on target platforms should be rather seen as a possibly transitional step in the development; the results coming from the execution would be utilised as an enrichment of the design models for a thorough extra-functional evaluation. Possible refinement of models can then be performed in order to generate code with preservation of the desired EFPs. Instead of manually tune the code, the prescribed manner to perform modifications is to edit the models and re-generate it.

This work proposes an automated round-trip support mechanism that: generates full source code (i.e., generated code that can be executed without manual tuning) entailing the creation of trace links between models and code, monitors code execution, and appropriately back-propagates observed values to modelling level.

II. BACKGROUND AND RELATED WORK

Exploitation of MDE paradigms can enable efficient modelling of EFPs and automation for their management throughout the development process [7]. In our approach, for the definition of design models, as well as EFPs, we exploit the CHESS modelling language (CHESS-ML) [8], a UML-profile that includes a subset of the SysML and MARTE profiles, and supports the component-based design pattern. Moreover, the behaviour specification is modelled in terms of UML state-machine diagrams enriched with Action Language for Foundational UML (ALF) [9]. The EFPs considered in this research work are: (1) throughput, mean time between failures (MTBF) and overall CPU usage, measured at system level, and (2) execution time and memory allocation calculated at subsystem and function level.

Efforts in dealing with EFPs related to composability can be found in the Web Service domain [10, 11]; domain-specific languages and UML profiles have been defined to model EFPs [12, 13]. Navabi et al. in [14] in the early 90’s, and some years later Mahadevan and Armstrong in [15], came up with different approaches for back-annotating behavioural descriptions with timing information; however, both operate only at modelling level thus not entailing code artefacts. A similar approach to ours is described by Guerra et al. in [16] where back-propagation of analysis results to the original model by means of triple graphical patterns is described. Yet, the approach only operates at modelling level with propagation of data among models. While, dealing with embedded systems, our approach focuses on vertically propagating analysis results...
observed at code level back to design models for better evaluating those EFPs that cannot be accurately predicted at higher levels of abstraction.

III. APPROACH AND UNIQUENESS

The proposed approach exploits the expressiveness of UML state-machines and ALF to generate full code in a completely automated manner. Moreover, generated code can be analysed and its execution monitored; results are meaningfully propagated back to the CHESS-ML models for evaluating the preservation of EFPs. The solution is composed by a set of model transformations that iteratively and progressively manipulates the CHESS-ML model and transform it to an intermediate representation from which full implementation code in C++ is generated. More specifically, two generators are currently provided: (1) C++ for Windows/Linux and (2) C++ for both single- and multi-process on OSE [17].

A set of model-to-model (M2M) transformations acts on the CHESS-ML model to generate three different intermediate artefacts: (a) Instance model, representing component instances and links among them via port instances [18], (b) Intermediate model, that is the main intermediate artefact in the process and contains all the information needed to generate code [19], and (c) Back-propagation model which contains explicit traceability links between model elements and their mapping to code units [19]. More specifically, an explicit trace link can be seen as a quadruple \(<ME, EU, MEP, V>\) where \(ME\) is a model element, \(EU\) is an executable unit, \(MEP\) is an EFP defined for \(ME\), and \(V\) is the value of \(MEP\) which is calculated during monitoring activities and propagated back to the related extra-functional annotation’s placeholder of \(ME\) in the design model.

The intermediate model is enriched with behavioural information defined in ALF by means of parsing operations and in-place M2M transformations. At this point, the C++ implementation code is generated through model-to-text (M2T) transformations taking as input the intermediate model. Once the code is generated, its execution can be monitored through appropriate routines on top of OSE. For instance, to calculate execution and response times, \(\text{swap}\_\text{in}\) and \(\text{swap}\_\text{out}\) handlers of OSE, that are respectively invoked each time a process gets CPU to execute and when CPU is taken from a process, have been modified.

The monitoring activities give a textual description of the gathered values as output. Through a text-to-model (T2M) transformation, monitoring results are manipulated in order to be stored in the correct placeholders held in the back-propagation model that will contain all the information needed to perform the actual back-propagation to the CHESS-ML model. Such activity is performed through an in-place M2M transformation that enriches the CHESS-ML model with the observed values stored in the back-propagation model.

M2M and T2M transformations are defined by means of QVTo [20], while M2T transformations are defined through Xpand [21]. Thanks to the multi-step nature of the approach, generation of other target languages can be enabled by defining ad-hoc M2T transformations from the intermediate model. Thanks to full automation, the process can be iterated at will until the designer is satisfied with the result.

The combination of full code generation together with the novel cross-cutting back-propagation capabilities makes the contribution of this research work unique. In fact, a process combining the features provided by the solution we propose has not been fully exploited yet in the current state-of-the-art.

IV. RESULTS AND CONTRIBUTIONS

The solution proposed in this work has been successfully validated against an industrial case-study in collaboration with Ericsson Nikola Tesla (Zagreb, Croatia) and Ericsson AB (Stockholm, Sweden). Concerning scalability, we tested several design model sizes on a machine running a 2.6GHz CPU and 8GB RAM, observing that the proposed solution resulted very scalable up to \(10^3\) (≈ 120 sec) component and port instances while degrading going toward \(10^4\) (> 850 sec).

Regarding the time needed for monitoring activities, it heavily depends on the duration of the code execution since the measurements themselves are mostly performed by monitoring processes during the execution. Nevertheless, the calculation of more complex EFPs could introduce additional complexity and therefore require additional computation time. Generally, the number of iterations for reaching desired EFPs depends on the accuracy of measurements as well as the modeller’s ability in both modelling the system and effectively exploit the back-propagated values to tune the models accordingly. In aid to the modeller, model-based analysis techniques could be exploited (as proposed in CHESS [8]) to minimise the number of iterations.

A first sketch of the code generation process was defined and presented in [22]. The whole round-trip approach was presented in [23] with focus on the traceability artefacts and mechanisms as well as on the feasibility of our novel back-propagation step; code generation and monitoring were considered as black boxes. A full implementation of the round-trip approach, including code generator and monitoring features, was developed and validated on top of the CHESS framework [8] and described in [19]. Definition and implementation of the proposed approach resulted in the author’s pre-doctoral licentiate thesis [24]. The problem of component instances and links generation has been presented in [18] as exploitation of the notion of semantic variation points in UML. Planned enhancements of the code generation process to consider complex deployment configurations were introduced in [25] and preliminary results are currently under submission. Based on that, a further evaluation of the whole approach toward deployment optimisation is currently under submission too.

The presented research work, both regarding implementation and publication of results, has been carried out by the author as main driver and contributor, under the oversight of his supervisors.
REFERENCES


