LET for Legacy and Model-based Applications

Andreas Naderlinger, Stefan Resmerita, Wolfgang Pree

Department of Computer Sciences, University of Salzburg
firstname.lastname@cs.uni-salzburg.at

***Abstract*—The Logical Execution Time (LET) paradigm has recently been recognized as a promising candidate to facilitate the migration to multi-core architectures in automotive real-time software systems. We outline several findings regarding the application of the LET paradigm that corroborate this perception. Our work in this respect deals with LET for legacy systems and LET in the context of model-based development (e.g., in MATLAB/Simulink). Furthermore, we present open issues and highlight implications on the development process when using LET as a synchronization mechanism.**

I. Introduction

Since its initial introduction in the Giotto project [1] almost two decades ago, several research groups have been working on the Logical Execution Time (LET) paradigm which has by now been the foundation for several programming languages and run-time systems [2]. While the promised advantages, such as time- and value-determinism, do sound desirable for safety-critical real-time systems, the approach has long been met with skepticism. Also, with a few exceptions (e.g., [3], [4]), industry has been reluctant in the trial, let alone the adoption of LET. With the emergence of multi-core architectures in automotive system this seems to change as LET could play a key role for obtaining predictable behavior when parallelizing control software. As a consequence, it recently experienced an increase in attention from both academia and industry (e.g., [5]). Amongst other benefits, LET shall provide deterministic inter-task communication across multiple cores on automotive multi-core architectures. Central questions that need to be dealt with involve, for example, how to reconcile performance-dominated requirements of control systems with the additional memory and computational costs that come with LET, how to apply this primarily top-down and correct-by-construction approach to legacy systems that may not satisfy all the initial assumptions, and also how and where to best introduce the LET concept in a development process that is no longer centered around code but on models specified, e.g., in MATLAB/Simulink. This abstract presents two active lines of work in our group dealing with these questions: (1) LET applied to legacy automotive systems including multi-core architectures, and (2) LET in the context of a model-based development with simulation in MATLAB/Simulink.

II. LET for Legacy Systems

A substantial amount of legacy code is used in many embedded system domains, in particular in the automotive industry. When carried over to a new hardware platform, data consistency issues arise and provisions must be made to maintain proper behavior along cause-effect chains. Our first work on applying LET to an industrial engine controller reaches back to 2010 [3], where the imposed restriction of limiting code changes to top-level functions, lead to a considerable increase in memory requirements (both RAM and ROM). Abandoning this restriction leads to a drastic reduction in run-time and memory overhead [6]. Both dimensions of overhead are largely dependent on the particular application and are depending also on the degree of freedom for choosing the exact LET [7], especially for multi-core targets. There is an enormous potential for optimizations when migrating to multi-cores using LET. Naturally, different optimizations are difficult to harmonize. For example, a strategy that reduces buffers and leads to less total run-time overhead could still lead to bulky and unacceptable copy-operations at a particular LET boundary. Also, the question is how far optimality of a certain setting (in whichever respect) impacts extensibility or changeability of the software and the potential validation effort that goes along with it. In [8], we propose a transformation process from single-core legacy software to LET-based versions that can be safely run on a multi-core. It is a process that can be applied incrementally and that is centered around a static buffer requirement analysis, which can be applied at different levels of abstraction. The most abstract level determines a minimal set of buffers for a given LET specification that is independent of the underlying platform configuration (including task priorities, scheduling, and function-to-core mapping). Being a minimal upper bound, this set can be further reduced in more refined abstraction layers where restrictions and details are incorporated into the analysis. For example, we describe an optimal buffering strategy w.r.t. the number of required buffers for a known multi-core platform configuration under fixed-priority preemptive scheduling.

At run-time, automotive applications change functionality to adapt computational demands according to the crank angle in order to avoid system overload, for example, at high engine speeds. This variation in physical execution times must be reflected also in the logical timing domain, e.g. using a multi-modal specification (as already supported by Giotto). So far, to the best of our knowledge, support for multiple modes in the context of LET and multi-core has not been addressed. It is unclear how this will increase the complexity of the analysis and the run-time system that ensures the LET semantics, and it is also unclear what the exact semantics should even be in the case of a mode switch and how this goes together with AUTOSAR modes.

III. LET in Model-based Development

Model-based design has become an established development approach in the field of embedded real-time systems. Clearly, LET should be an established fixture already in the modeling and simulation phase of the development. The predominant environment for modeling and simulating automotive control systems is MATLAB/Simulink, which is based on the synchronous block diagram (SBD) formalism. Being built on the synchronous reactive programming paradigm, SBD is also suited to realize LET behavior. However as we outlined in [9], in the presence of cyclic data-dependencies as found between AUTOSAR runnables, for example, care must be taken to comply with limitations implied by the simulation engine such that a valid execution order of the blocks can be found. In [10], we present a Simulink implementation of a run-time system (E-machine) for a multi-mode multi-rate LET specification involving potentially cyclic data dependencies. The simulated control algorithms may be implemented as Simulink/Stateflow blocks or in the programming language C.

Originating from a purely control-engineering oriented view, since at least the introduction of AUTOSAR support, Simulink models realign to more and more software-centric perspectives. It is not clear how a clean transition from platform-independent to platform-dependent models that support push-button code generation can be achieved. In any case, for obtaining highly optimized code with a minimal number of additional LET buffer variables, for example, the code generation for a particular runnable must not be considered in isolation. Timing and data-flow dependencies of the whole application must be taken into account.

The need for considering data-flow dependencies is not only an issue of optimization. In a *classic* LET-based specification, the LET interval of an individual task (or function) was mainly driven by physical requirements (expressed in the period) and inevitably by properties of the hardware/system (implied by worst-case execution/reaction times). Since in the multi-core setting LET is used as a synchronization mechanism, LET intervals must be harmonized across multiple cores and thus cannot be decided individually on task/function-level. This has implications on the whole development process (and also on the mode-switch issue discussed in the previous section). Despite this, the development process might benefit from using LET as a design contract between control and embedded software engineers as outlined in [11].

In the standard LET model, where a task's LET equals the period, end-to-end latencies are a major concern. However, when the LET is contained in the period, this issue is considerably relaxed [12].

An open issue, for example, is robustness w.r.t. the impact of a model change (e.g., adding a new data-dependency) on the generated code and how the attempt to minimize code changes relates to the resulting run-time efficiency.

IV. Conclusion

This abstract touched on aspects of LET related to its application to automotive software systems, especially for a single- to multi-core migration. We hereby covered legacy and model-based applications and gave examples of open issues in this respect.

References

[1] T. A. Henzinger, B. Horowitz, and C. Kirsch, "Giotto: A time-triggered language for embedded programming," *Proceedings of the IEEE*, vol. 91, pp. 84–99, January 2003.

[2] C. M. Kirsch and A. Sokolova, "The logical execution time paradigm." in *Advances in Real-Time Systems*, S. Chakraborty and J. Eberspächer, Eds. Springer, 2012, pp. 103–120.

[3] S. Resmerita, K. Butts, P. Derler, A. Naderlinger, and W. Pree, "Migration of legacy software towards correct-by-construction timing behavior." in *Monterey Workshop*, 2010, pp. 55–76.

[4] V. Belau, H. von Hasseln, and M. Simons, "Coordinating AUTOSAR runnable entities using giotto - first concepts," 2012, poster presented at DEPCP 2012, Dresden, Germany.

[5] A. Hamann, D. Dasari, S. Kramer, M. Pressler, F. Wurst, and D. Ziegenbein, "Waters industrial challenge 2017," 2017, (and 8th International Workshop on Analysis Tools and Methodologies for Embedded Real-time Systems, WATERS 2017).

[6] S. Resmerita, A. Naderlinger, M. Huber, K. Butts, and W. Pree, "Applying real-time programming to legacy embedded control software," in *2015 IEEE 18th International Symposium on Real-Time Distributed Computing*, April 2015, pp. 1–8.

[7] J. Hennig, H. von Hasseln, H. Mohammad, S. Resmerita, S. Lukesch, and A. Naderlinger, "Towards parallelizing legacy embedded control software using the LET programming paradigm," in *Proc. of WiP Papers of the 22nd IEEE Real-Time and Embedded Technology and Apllications Symposium*, ser. RTAS '16, 2016.

[8] S. Resmerita, A. Naderlinger, and S. Lukesch, "Efficient realization of logical execution times in legacy embedded software," in *Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design, MEMOCODE 2017, Vienna, Austria, September 29 - October 02, 2017*, 2017, pp. 36–45.

[9] A. Naderlinger, J. Templ, and W. Pree, "Simulating real-time software components based on logical execution time," in *Proceedings of the 2009 Summer Computer Simulation Conference*, ser. SCSC '09. Vista, CA: Society for Modeling & Simulation International, 2009, pp. 148–155.

[10] J. Templ, A. Naderlinger, P. Derler, P. Hintenaus, W. Pree, and S. Resmerita, *Real-Time Simulation Technologies: Principles, Methodologies, and Applications (Computational Analysis, Synthesis, and Design of Dynamic Systems)*. CRC Press, April 2016, ch. Modeling and Simulation of Timing Behavior with the Timing Definition Language, pp. 159–178.

[11] P. Derler, E. A. Lee, M. Törngren, and S. Tripakis, "Cyber-physical system design contracts," in *2013 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, April 2013, pp. 109–118.

[12] W. Pree, J. Templ, P. Hintenaus, A. Naderlinger, and J. Pletzer, "TDL - steps beyond giotto: A case for automated software construction," *Int. J. Software and Informatics*, vol. 5, no. 1-2, pp. 335–354, 2011.