Automatic implementation of TTEthernet-based time-triggered avionics applications

Raul Adrian Gorcitz, Thomas Carle, David Lesens, David Monchaux, Dumitru Potop-Butucaru, Yves Sorel

Automatic implementation of TTEthernet-based time-triggered avionics applications

Raul Adrian Gorcitz*, Thomas Carle†, David Lesens‡, David Monchaux*, Dumitru Potop-Butucaru†, Yves Sorel†
*CNES - Centre National d'Études Spatiales
†INRIA - Institut National de Recherche en Informatique et en Automatique
‡Airbus Defence and Space

***Abstract*—The design of safety-critical embedded systems such as those used in avionics still involves largely manual phases. But in avionics the definition of standard interfaces embodied in standards such as ARINC 653 or TTEthernet should allow the definition of fully automatic code generation flows that reduce the costs while improving the quality of the generated code, much like compilers have done when replacing manual assembly coding. In this paper, we briefly present such a fully automatic implementation tool, called Lopht, for ARINC653-based time-triggered systems, and then explain how it is currently extended to include support for TTEthernet networks.**

I. Introduction

The implementation of complex embedded software relies on two fundamental and complementary engineering disciplines: *real-time scheduling* and *compilation*. Real-time scheduling covers[1] the upper abstraction levels of the implementation process, which determine how the *functional specification* is transformed into a set of *tasks* and then determine how the tasks must be *allocated* and *scheduled* onto the *resources* of the execution platform in a way that ensures *functional correctness* and the respect of *non-functional requirements*. By comparison, compilation covers the low-level code generation process, where each task (a piece of sequential code written in C, Ada, *etc.*) is transformed into machine code, allowing actual execution.

In the early days of embedded systems design, both high-level and low-level implementation activities were largely manual. However, this is no longer the case in the low level, where manual assembly coding has been almost completely replaced by the use of languages such as C or Ada and compilers [1]. This shift towards high-level languages and compilation allowed a significant *productivity gain* by ensuring that source code is *safer* and *more portable*. As compiler technology improved and systems became more complex in both hardware and software, compilation has also approached the *efficiency* of manual assembly coding, and in many cases outperformed it. It is important to note here that the widespread adoption of compilation that we see today was only possible due to the early adoption of standard interfaces that allowed the definition of economically-viable compilation tools (with a large-enough user base). These interfaces include high-level languages such as C, Ada, *etc.*, relatively stable microprocessor instruction set architectures (ISAs), and finally executable code formats like ELF.

[1]Together with other disciplines such as systems engineering, software engineering, *etc.*

The paradigm shift towards fully automated code generation is far from being completed at the system level. Aspects such as the division of the functional specification into tasks, the allocation of tasks to resources, or the configuration of the real-time scheduler are still performed manually for most industrial applications. Furthermore, research in real-time scheduling has largely followed this trend, with most (but not all) effort still invested into verification-based approaches aimed at proving the *schedulability* of a given system (and into the definition of run-time mechanisms improving resource use).

This slow adoption of automatic code generation can be traced back to the slower introduction of standard interfaces allowing the definition of economically-viable compilers. Functional specification languages such as Simulink, LabVIEW, or SCADE have been introduced in the mid-1980s, which allowed the gradual definition of techniques for the synthesis of functionally-correct sequential or even multi-task embedded code (but without real-time guarantees). The next major step came in the mid-1990s, when execution platforms have been standardized in fields such as avionics (IMA/ARINC 653) and automotive (OSEK/VDO, then AUTOSAR). This second wave of standardization already allowed the industrial introduction of automatic tools for the synthesis of processor schedules or network schedules.

The research community went farther and proposed real-time implementation flows that automatically produced running real-time applications [2], [3] where the processor and network schedules are jointly computed using a *global optimization approach* that results in better resource use. Of course, more work is needed to ensure the industrial applicability of such results. For instance, the aforementioned techniques cannot handle all the complexity of IMA avionics systems, which involve functional specifications with multiple execution modes, multi-processor architectures with complex interconnect networks, and complex non-functional requirements including real-time, partitioning, preemptability, allocation, *etc.*

Our research objective is to bring fully automatic implementation to such complex embedded systems, and thus help in improving the safety and efficiency of the resulting systems while also improving the productivity of the design process.

In previous work [4], [5], [6] we have defined an automatic implementation method and tool, called Lopht, which is able to handle all the complexity elements of a realistic case study provided by Airbus DS [4] (which includes all those mentioned above). Our tool uses advanced optimization algorithms

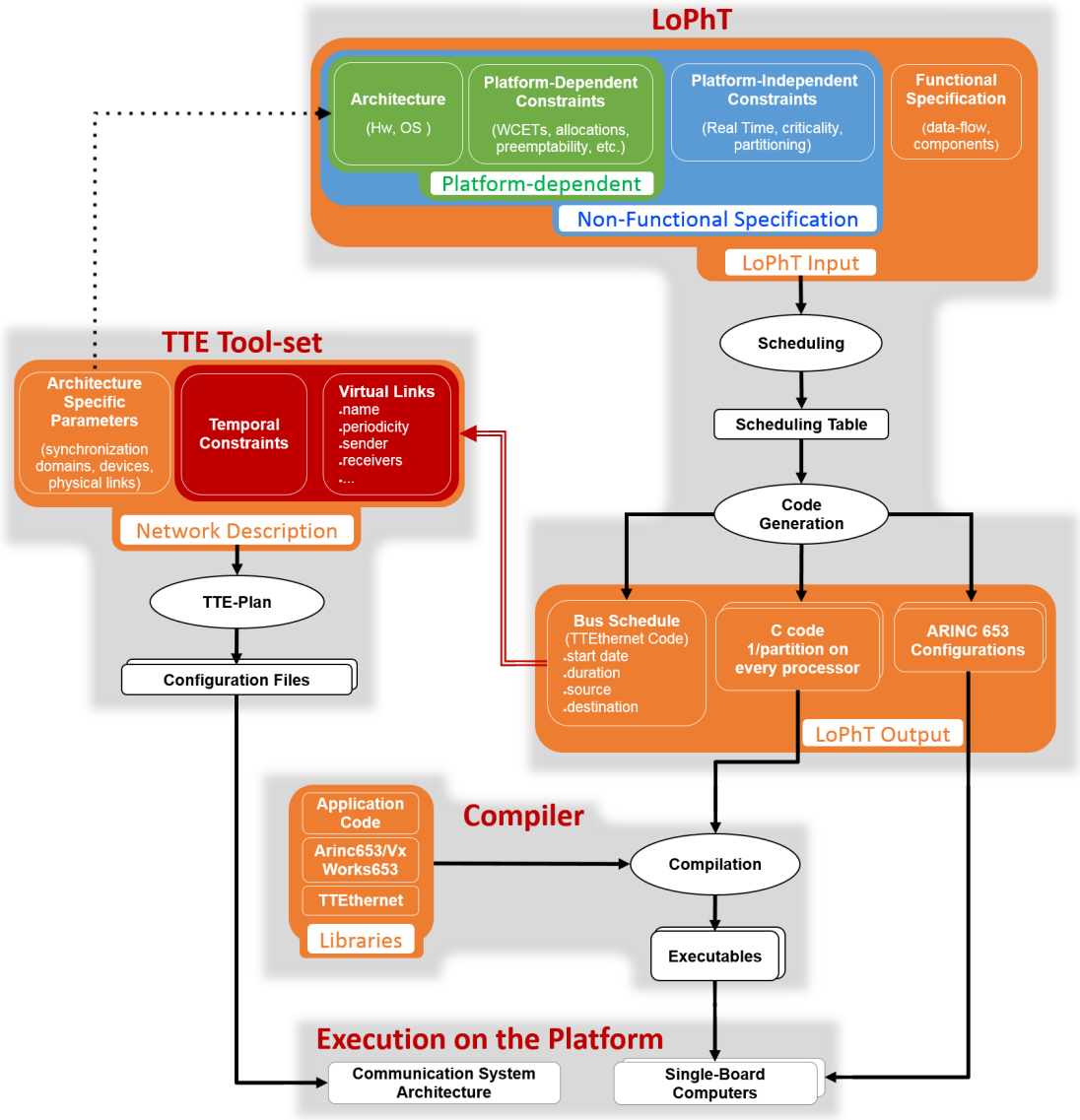

Figure 1. Lopht-based implementation flow for Time-Triggered Ethernet

which result in *high resource use* and *low numbers of partition/context changes*. It fully automates the implementation process, including the *creation of tasks* from the functional specification, the full *synthesis of ARINC 653 code* (C/APEX code for the partitions, including communication code, and OS configuration for each computer), and the *synthesis of communication schedules* for the system bus. Last, but not least, Lopht is fast (like a compiler) and it is easy to trace the cause of implementation errors. While designed to cater for the needs of an avionics application, our tool has been successfully applied to a transportation case study [6].

The purpose of this paper is to describe our ongoing work aimed at extending Lopht to fully take into account one of the avionics execution platforms considered by the Avionic-X project for use in future space launchers [7]. The main difference between this execution platform and the ones already handled by Lopht is the use of a Time-Triggered Ethernet (TTE) communication network instead of a more classical bus. Taking into account TTE amounts to interconnecting the Lopht tool with the TTE network configuration toolset, and thus building the fully automatic implementation flow of Fig. 1.

The remainder of the paper will first introduce Lopht (in Section II) and TTE (in Section III), and then will explain how the principles of our ongoing work on connecting the two.

II. The LoPhT automatic implementation tool

Lopht is an off-line mapping tool for multi-processor (distributed) systems designed to allow scheduling and code generation for time-triggered platforms where all the processors and communication media share a common time base.

It adopts a compilation-like approach that uses advanced off-line scheduling and compilation techniques designed to improve resource use (and thus schedulability). Such techniques are the use of pre-computed preemption, the exploitation of

execution conditions, the use of software pipelining, and post-scheduling optimizations aiming at reducing the number of partition switches.

As pictured in Fig. 1, Lopht takes as input a functional specification and a non-functional specification. The *functional specification* is a high-level data-flow synchronous (Scade) program. It allows the representation of system tasks, their execution conditions (modes), and their dependencies. These programs are deterministic and have well-defined formal semantics allowing formal analysis.

The *non-functional specification* comprises both platform-independent and platform-dependent aspects. The *platform-independent requirements* are those provided by control engineering[2] and by platform-independent systems engineering analysis. Such constraints concern the real-time behavior of the tasks (*release dates*, *deadlines*, *periods*), or the *partitioning* of the application according to task criticalities. The platform-dependent specification includes the architecture specification (its *resources* and the *interconnect topology*), the worst-case execution durations of the various tasks and communications (WCETs and WCCTs), and other platform-dependent requirements such as task preemptability, allocation constraints, partitioning constraints. In our case, partitioning means ARINC 653 time partitioning, and the non-functional specification may define the major time frame (MTF) of the system and pre-defined allocations of computation windows. Lopht can currently handle only communications over broadcast communication buses.

As pictured in Fig. 1, the first phase of the Lopht flow takes this functional and non-functional specification and performs a multi-processor real-time scheduling phase that produces a scheduling table describing one hyper-period of the execution of the system. This scheduling table defines:

- The allocation of tasks and communications to processors and buses (including source and destinations for each communication).
- The processor time reservations made for each task. Each reservation consists of a start date, a duration, and an execution condition.
- The bus time reservations made for each communication, which are characterized in the same way as those of processors.

From this table, Lopht generates executable code compliant with the APEX API of ARINC 653 (one C file for each partition of each processor) and a full configuration of the real-time aspects of ARINC 653 (one configuration for each processor).

A variant of the Lopht tool exists, named Lopht-manycore for mapping onto many-core platforms using a network-on-chip interconnect [8], [9]. While this variant does not take into account all the non-functional properties described above (no release dates, deadlines, partitioning, or interruptibility), we mention it here because it allows the manipulation of complex network architectures that are very similar to TTE. Indeed, even if TTethernet is often compared (due to its application field) to buses such as CAN [10] or TTA [11], it is technically not a bus, but an interconnection network formed of unidirectional point-to-point links and switches (routers), and where arbitration/scheduling must be separately defined for each link, just as we do in Lopht-manycore.

[2]Obtained through discretization of the continuous-time control specification.

III. The Time-Triggered Ethernet Network

Time-Triggered Ethernet[3] is a communication network standard and product coming from TTTech [12]. On top of a switched Ethernet basis, TTE adds support for real-time and fault tolerant communications. It allows multiple communications of mixed criticalities to share a single physical medium. This is ensured by means of dedicated hardware using a set of configuration files describing the system architecture and behavior. These configurations are synthesized by the proprietary TTEplan tool starting from a global *network description* file. In this paper, our objective is to synthesize this network description based on the output of Lopht.

1) Types of Traffic: TTEthernet networks allow three types of traffic to co-exist on the same physical links:

- Time-triggered (TT) traffic, where data packets are transmitted following a periodic predefined schedule on each network link, thus ensuring strong functional and temporal determinism properties.
- Rate-controlled (RC) traffic that follows an asynchronous AFDX-like traffic control paradigm using bandwidth allocation gaps (BAGs) to offer latency and throughput guarantees.
- Best-effort (BE) traffic that uses classical Ethernet arbitration over the network space not used by TT and RC traffic.

System-wide time synchronization is ensured through the transmission of a specific type of RC traffic, called protocol control frames (PCF), which are generated and consumed by the network infrastructure itself (not by the computers connected to the network).

The work presented in this paper only uses TT traffic. The objective is therefore to use Lopht in order to jointly compute the schedule of both computations on processors and packet transmissions on all the links of the TTE network.

2) Virtual Links and Packets: All TTE traffic is organized into virtual links (VLs). Each VL has a set of characteristics such as sender, receiver(s), etc. The real-time characteristics of a VL depend on the type of the VL (TT, RC, or BE). For instance, TT VLs have a period, a time window for initiating the transmission, and a maximum packet size. During each period of a TT VL only one TT packet is sent, subject to the size limits defined below.

Like in Ethernet, all data transmissions are packetized. Packets must have less than 1499 bytes. PCF traffic requires smaller packets of only 64 bytes. When TT traffic is used, a TTE network must operate at one of two speeds: 100Mbps or 1000Mbps.

[3]The description of this section is based on the TTE documentation that was available when the paper was written. As TTE is actively being developed by TTTech, some of the aspects of this technology might change over time.

3) Time Bases: Like other time-triggered standards, TTE uses a a discretized time base, and various time scales can be used for synchronization at different levels. Our code generation work will need to consider three time scales of TTE:

- The *raster tick* is the finest timing resolution. It is used for fine-grained scheduling of TT packets.
- The *integration cycle* is a multiple of the raster tick. It is the period with which PCF packets are sent on the network to perform clock resynchronization. Shorter integration cycles result in better clock synchronization precision.
- The *cluster cycle* is a multiple of the integration cycle (it can be equal to it). It must also be a multiple of the period of all TT VLs (and thus of their least common multiple). The cluster cycle defines the length of the scheduling table which is periodically repeated during execution. It is the time scale where synchronization with the operating system of the processors (end systems) is performed.

In addition to these three time scales, TTE defines a fourth time-related parameter, called the *raster granularity slot*. If its value is n, then the TTEplan scheduler can schedule at most one TT frame every n raster ticks. As only one TT packet can be sent on a TT VL during one raster tick it is important to note that this constraint limits the amount of bandwidth available for TT traffic, due to the fact that the maximal TT packet is substantially shorter than the raster tick.

IV. Interfacing Lopht and TTEthernet

Building the integrated design flow pictured in Fig. 1 requires a careful integration of the Lopht and TTEplan tools. This integration requires modifications at several levels in Lopht. The most important of these modifications is that of the formal platform model taken as input by Lopht to allow the representation of TTE networks. Choosing a good platform model is key in ensuring that resources are efficiently allocated by Lopht and that scheduling tables generated by Lopht are implementable using the TTEplan toolset.

Previous work on modeling networks-on-chips (NoCs) in Lopht-manycore [8] provides us with a good starting point, which has the advantage of being readily compatible with the internal scheduling infrastructure of Lopht. However, this model needs to be modified to take into account TTE-specific information, such as the periodicity of TT VLs and the existence of multiple time bases.

One particular issue here is the limited, but still existing scheduling freedom of the TTE arbiters. To facilitate platform description (and also the scheduling algorithms), we make supplementary hypotheses on the way TTE is used, detailed in Section IV-A, which allow us to consider that the TTE network is strictly time-triggered without taking large timing margins.

The second level where modifications are needed is that of the scheduling algorithms, which need to be modified to take into account four important TTE-specific properties: the fact that TTE uses store-and-forward switching (as opposed to the wormhole switching that is common in NoCs), the requirement that message transmission is periodic on all TT VLs, the fact that TTE does not support conditional communication, and the existence of multiple time bases (*e.g.* on the TTE network and on the processors). Again, existing algorithms provide a good staring point, but some adaptations are needed.

Finally, we need to modify the code generator to allow the synthesis of the TTEplan input (the network description file). We shall detail this in Section IV-B.

A. Assumptions on TTE use

Our objective is to allow the precise **off-line** scheduling of TT packets in the time base provided by the raster ticks. To do this, we complete the previous description of TTEthernet with three assumptions facilitating our scheduling work:

- The sending of TT packets on each network link (between one end station and one switch, or between two switches) is scheduled on raster tick start points.
- All TT packets are assumed to have the maximal size fixed by Ethernet (no optimization is attempted dependent on the size of various packets).
- PCF traffic and the synchronization precision can be neglected during scheduling.

The first assumption aligns the sending of all packets, at the level of all TTE devices (end stations and switches) on raster tick starts. Of course, on a route involving several devices this results in a loss of efficiency when a packet is buffered after reception until at least the next raster tick.

The raster tick where a TT packet is scheduled should allow it to be scheduled fully under the worst-case influence coming from PCF packets. For instance, in the chosen configuration (100Mbps network speed), the recommended raster tick is 200µs. By comparison, the transmission of the maximal TT data packet takes 120µs, and a PCF packet takes 5µs to be transmitted. Under the assumption that the TT frame is scheduled at the beginning of the raster tick, we assume that the remainder of raster tick is sufficient to accommodate the worst-case load coming from PCF packets regardless of the TTE network topology and other network setting. Furthermore, we assume that the remainder of the raster tick can also accommodate the worst-case of the clock difference between the sender and receiver.

The last two assumptions imply that in the time base provided by the raster tick we can reason about scheduling as if PCF traffic did not exist. Furthermore, we can also assume that the transmission of a TT packet will always start and complete inside the raster tick where it has been scheduled. *Under these hypotheses, we can perform scheduling in the discrete time base of the raster tick, and all scheduling constraints for communications (release dates, deadlines) can be formulated in this time base.*

B. Synthesis of TTEplan input

The configuration of a TTE network is realized using the proprietary TTEplan tool [13]. This tool takes as input a single network description file which specifies the architecture of the communication system, the VLs of the system, and their

attributes. In our case, this input file will only contain TT VL definitions which must be (partly) synthesized from the output of Lopht. The following informations need to be provided for each VL:

1) *the communication name and the VL id* - Identifier of the virtual link.
2) *the maximum payload* - The maximum payload carried by a frame of this VL.
3) *the type of the link* - The type of traffic (TT).
4) *the period of the communication* - The transmission periodicity of this VL.
5) *the source system* - The device initiating this communication.
6) *the set of destination systems* - The intended receivers of the transmitted data.
7) *datalink* - Interface with the data link layer.

In addition to the VL definition, network configuration files can also include real-time constraints on the VLs. TTE-Plan accepts three types of constraints:

- **send time** - Defines the earliest point in time where the transmission can be initiated.
- **receive time** - Defines a deadline for the end of a transmission reception.
- **transmission duration** - Declares a maximum allowed communication time.

It is important to note that, to our knowledge, the network description file does **not** allow the specification of communication paths or the exact time-triggered scheduling of packet sends at specific at specific points in the network.

This poses us a significant problem, as the basic principle of *LoPhT* is to perform a global scheduling of both computations and communications. The output of *LoPhT* is a scheduling table specifying the time-triggered scheduling of each resource (link) of the TTE network. More precisely, the output of *LoPhT* comes under the form of a set of time reservations over the resources of the platform, each reservation defining two temporal attributes: the start time and the duration of the time reservation.

Translating such scheduling tables into TTEplan input can only be done by losing information. The simplest code generation, which we will implement first, synthesizes one VL for each data-flow arc of the functional specification, after the hyper-period expansion detailed in [4]. Hyper-period expansion has the disadvantage of increasing the number of VLs, and the advantage of improving schedulability.[4] Under this code generation approach, each VL has a period equal to the hyper-period, and we associate to it one send time constraint and one receive time constraint using the dates prescribed by the scheduling table. These constraints are provided in the raster granularity time base.

The resulting network configuration file is provided to TTEplan, and two outcomes are possible:

- TTEplan finds a solution. This solution will not be necessarily the same one that *LoPhT* found, but the way it was constructed means that it can be used instead to build the running implementation.
- TTEplan does not find a solution even though one exists, as determined by *LoPhT*. To handle such cases, three solutions can be envisioned:
  - Modifying the output of *LoPhT* to provide the maximum freedom of scheduling to TTEplan, but without compromising the global schedulability determined by *LoPhT*.
  - Enriching the constraint language of TTEplan to allow the specification of routing and scheduling at link level.
  - Directly building the TTE configuration files, thus bypassing TTEplan.

We do not expect this last point to be blocking given the network loads of our case studies.

[4]Ongoing work aims at dealing with multi-periodic specifications without performing a hype-period expansion.

REFERENCES

[1] Embedded.com, "2009 embedded market study," Online, Jan 2009, http://www.embedded.com/electronics-blogs/embedded-market-surveys/4405221/2009-Embedded-Market-Survey.

[2] T. Grandpierre and Y. Sorel, "From algorithm and architecture specification to automatic generation of distributed real-time executives," in *Proceedings MEMOCODE*, Mont St Michel, France, 2003.

[3] S. S. Craciunas and R. S. Oliver, "Smt-based task- and network-level static schedule generation for time-triggered networked systems," in *Proceedings RTNS*, Versailles, France, October 2014.

[4] T. Carle, D. Potop-Butucaru, Y. Sorel, and D. Lesens, "From dataflow specification to multiprocessor partitioned time-triggered real-time implementation," http://hal.inria.fr/hal-00742908/PDF/RR-8109.pdf, INRIA, Research Report RR-8109, Oct. 2012.

[5] T. Carle and D. Potop-Butucaru, "Predicate-aware, makespan-preserving software pipelining of scheduling tables," *ACM Transactions on Architecture and Code Optimization*, vol. 11, no. 1, 2014.

[6] A. Cohen, V. Perrelle, D. Potop-Butucaru, E. Soubiran, and Z. Zhan, "Mixed-criticality in railway systems: A case study on signalling application," in *WMCIS Proceedings of the Workshop on Mixed Criticality for Industrial Systems*, 2014.

[7] D. Monchaux, P. Gast, and J. Sangare, "Avionic-x: A demonstrator for the next generation launcher avionics," in *Proceedings ERTS²*, Toulouse, France, February 2012.

[8] T. Carle, M. Djemal, D. Potop-Butucaru, R. D. Simone, and Z. Zhang, "Static mapping of real-time applications onto massively parallel processor arrays," in *Proceedings ACSD*, Tunis, Tunisia, June 2014.

[9] T. Carle, M. Djemal, D. Genius, F. Pêcheux, D. Potop-Butucaru, R. de Simone, F. Wajsbürt, and Z. Zhang, "Reconciling performance and predictability on a many-core through off-line mapping," in *Proceedings RTNS*, Montpellier, France, May 2014.

[10] "CAN (iso 11898): Controller area network," http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=33422.

[11] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 138–144, January 2003.

[12] "SAE AS 6802: Time-Triggered Ethernet," http://standards.sae.org/as6802/, 2011.

[13] *TTE-Plan: The TTEthernet Scheduler*, 4.1 ed. TTTech, 2013.