Mixed-criticality in Railway Systems: A Case Study on Signaling Application

Albert Cohen[†], Valentin Perrelle[*], Dumitru Potop-Butucaru[†], Elie Soubiran[‡*], Zhen Zhang[*]

[*]Technological Research Institute SystemX
{valentin.perrelle, elie.soubiran, zhen.zhang}@irt-systemx.fr

[†]INRIA, France
{albert.cohen, dumitru.potop_butucaru}@inria.fr

[‡]Alstom Transport
{elie.soubiran}@transport.alstom.com

I. Introduction

Since the early 2000's almost every new metro project in the world make use of a standardized railway signalling system called Communication Based Train Control (CBTC) (IEEE 1474). Previously to CBTC, conventional signalling train control systems were relying almost exclusively on track circuits, wayside signals and operating procedures to ensure train protection and operation. In order to ensure better operational performance (e.g. effective utilization of the transit infrastructure), CBTC systems rest on three pillars: "*Automatic train control (ATC) based on high-resolution train location determination, independent of track circuits*"; "*high-capacity and bidirectional train-to-wayside data communications*"; and "*train-borne and wayside computing units that execute vital functions*". Functions are classified within three families that are: Automatic Train Protection (ATP), Automatic Train Operation (ATO) and Automatic Train Supervision (ATS). The level of criticality differs from a family to another and without loss of generality, one can state that ATP functions are mostly safety critical functions (SIL4 regarding CENELEC 50126), whereas ATO and ATS gather functions of low criticality (from SIL0 to SIL2). As a matter of fact, CBTC systems are in essence Mixed-critical systems. Furthermore, the mainstream evolution of those systems tends toward more functional integration on more powerful computing units. ATP and ATO functions that were traditionally distributed on different computing units (both on wayside and train-borne) tends now to be deployed on the same computing units and thus sharing resources.

FSF[1] is an IRT SystemX project positioned on two topics, the first one is about the conception of signalling applications (typically ATO/ATP application) that contain both critical and non-critical parts and the second one is on execution platforms that execute those applications while offering high guarantee of safety and availability. Industrial expectations around the execution platform include the use of multi-core COTS, the use of modern RTOS that offer spatial and temporal isolation, the use of safety and availability architectural patterns (e.g. voting and redundancy), and the whole being finally hidden behind a "system abstraction layer". On top of this platform, a tooled framework is prototyped and allows one to develop, verify and deploy component based applications where components may arbitrary contains both vital (SIL4) and non-vital (SIL0) code. The project has started in May 2013, the aim of this communication is to propose a first return of experience and a positioning on how mixed critical issues will be addressed in FSF.

[1]FSF is a French project name acronym standing for "safe and reliable embedded systems"

Alstom Transport has defined an applicative case study that, while being limited to one single ATC function, is representative of the complexity in term of vital/non-vital code interweaving, operational performance and availability constraints. The system function is called "Passenger Exchange" (PE). This function takes control on the train when this one is safely docked at a station; it organizes the exchange of passengers (train and station doors opening/closing) while protecting them from any untimely train movement or non-aligned doors opening and finally gives the departure authorization when all safety conditions are met. The functional specification is made of more than 300 requirements (natural language + SysML), and the functional structure is made of about twenty sub functions.

PE is designed as a system component containing both vital and non-vital parts. At this level a component is roughly a packaging unit that exposes to the exterior world a set of ports (in or out) and that is characterized by a set of behaviours that depend on the operational environment. This component is then broken down into a set of atomic software components which are this time exclusively vital or non-vital. An important remark is that there are no restrictive design constraints on data dependency between the vital and non-vital atomic components.

To illustrate this fact, let us consider a simplified example from the case study, depicted in Fig. 1. The vital components, in red, are in charge of controlling operations. This can be summarized by computing which doors are safe to open (e.g. because they are not aligned) and by preventing train departure if safety conditions are not met (e.g. the doors are open or opening). The non-vital components are in charge of operating doors with respect to a mission protocol and a time table. In this example we can identify two occurrences of vital to non-vital communications. First, in order to ensure that doors commands (non-vital) do not lead to an accident, they must be checked against the enabled set of doors (vital). This is the role of the "Process output" atomic component. Second, the departure authorization (vital) must be computed regarding

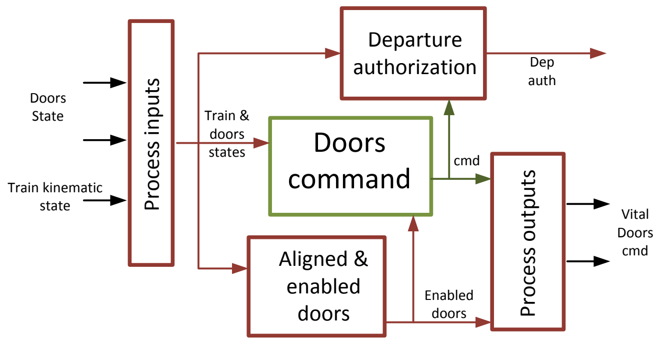

Figure 1. Simplified view of the component breakdown in PE.

door commands to ensure that no opening commands will be executed after the authorization has been given.

This is a simplified example. Such communication patterns are quite common in the complete case study.

II. Synchronous approaches

A. Synchronous languages

Data-flow synchronous languages, such as LUSTRE [1] or SIGNAL [2] have been designed in the 80's for program real-time safety critical embedded systems. Since then, they have been widely used in industrial applications [3]. These languages emphasize a correct-by-construction approach, ensuring bounded memory and execution time. Moreover, they are praised for their predictable behaviour and formally defined semantics.

Recently, the problem of scheduling multi-rate, mixed-critical synchronous programs have been addressed, at first for uni-processor [4] then for multi-processors [5]. Outside the scope of mixed-criticality there were also several attempts to distribute synchronous data-flow languages [6], [7], [8]. Recent work have been done to develop these languages to target multi-core platforms through the programming of parallelism [9]. This work introduces futures in LUSTRE-like languages giving the guarantee that the sequential semantics is preserved.

B. Automatic allocation, partitioning, and scheduling

Due to their use in the avionics industry, synchronous languages have been considered early on as an input formalism for the automatic or semi-automatic synthesis of real-time implementations. Most significant in this direction are previous results by previous work by Sorel *et al.* [10] on the AAA/SynDEx methodology and tool for distributed, but not time-triggered, real-time implementation of multi-periodic synchronous specifications, previous work by Caspi *et al.* on the use of Lustre/Scade in the real-time implementation of Simulink over multi-processor platforms based on the time-triggered partitioned bus TTA [11], and previous work by Forget *et al.* [12] on the specification and implementation of multi-periodic applications over a time-triggered platform using the Prelude language.

But none of these approaches allow us to take into account all the characteristics of our case study in order to allow automatic mapping. In particular, none of them has support for ensuring the time and space separation between application parts with different *criticalities*.

This is why we considered in this project a new tool, named LoPhT [13], [14], which allows the automatic mapping of applications onto platforms following the ARINC 653 time and space partitioning mechanisms. The LoPhT tool takes as input deterministic functional specifications provided by means of synchronous data-flow models with multiple modes and multiple relative periods. (Cf. the LoPhT part of Fig. 2) These specifications are extended to include a real-time characterization defining task periods, release dates, and deadlines. Task deadlines can be longer than the period to allow a faithful representation of complex end-to-end flow requirements. The specifications are also extended with allocation constraints and partitioning information meant to represent the criticality of the various tasks, as well as information on the preemptability of the various tasks. Starting from such specifications, the LoPhT tool performs a fully automatic allocation and off-line scheduling onto partitioned time-triggered architectures. Allocation of time slots/windows to partitions can be fully or partially provided, or synthesized by LoPhT. The mapping algorithms of LoPhT take into account the communication costs. The off-line mapping algorithms of LoPhT use advanced mapping techniques such as software pipelining and pre-computed preemption to improve schedulability and minimize the number of context switches.

III. Case study

The PE case study has been implemented and a first demonstrator has been produced. The challenge for this first demonstrator was to propose a framework for on the one hand the design and implementation of components and on the other hand the design of signalling application its partitioning and scheduling.

Choice of software modelling language. We chose to use the language HEPTAGON, very similar to LUSTRE and featuring novel constructions and novel optimizations. Two criteria have influenced the choice of the language. First, the functional specification defined at system level and allocated to software components have been written in a reactive and mostly equational way. It was thus very natural to implement it in a synchronous data-flow language. Second, the normative referential (CENELEC 50128) recommends the use of formal methods for the development of vital software while making no restrictive assumption on the language used for the non vital part. Synchronous languages are a good trade-off since they enable the use of formal methods (for instance model checking or abstract interpretation) while providing a sufficient power of expression to implement non-vital components. Finally, having a single language to develop both vital and non-vital components allows not only the early simulation of functional behaviour without integration effort but also the rationalization of competence in the software development team.

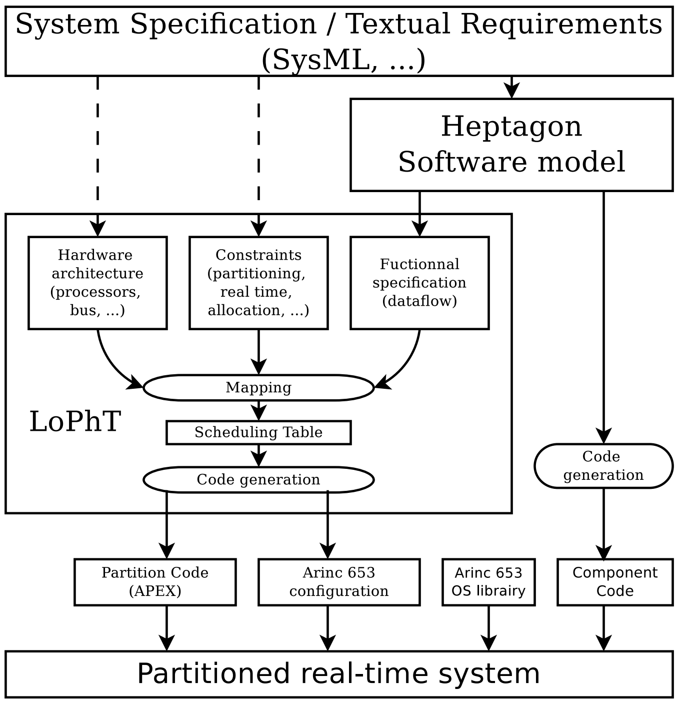
System Specification / Textual Requirements (SysML, ...)
Heptagon Software model
Hardware architecture (processors, bus, ...)
Constraints (partitioning, real time, allocation, ...)
Fuctionnal specification (dataflow)
Mapping
LoPhT
Scheduling Table
Code generation
Code generation
Partition Code (APEX)
Arinc 653 configuration
Arinc 653 OS librairy
Component Code
Partitioned real-time system

Figure 2. The global flow of the use case.

Technical realization. We developed the Passenger Exchange sub-components following a five step process depicted in Fig. 2:

1) In a SysML environment, we produced a component design that realizes the Passenger Exchange function. System requirements are traced and refined to define atomic components that correspond to software components and that are either vital or non-vital.
2) We mapped every atomic component to a HEPTAGON node realizing its functional behaviour.
3) Depending on the SIL of the component, verification activities have been conducted, but these are outside the scope of this communication.
4) We built a small signaling application composing multiple components into a realistic full-system scenario. These components include the PE itself, train/station interfaces, and a simulation of other system functions (such as train driving and passenger behavior). In HEPTAGON, the application is an assembly of nodes. At this stage, a first executable code is produced to simulate the application behaviour, however no insurance is given on spatial isolation.
5) In LOPHT, the atomic components are allocated to three partitions, which are "P0: vital", "P1: non-vital" and "P2: environment". Meanwhile the execution durations are given. Five windows are created. The scheduling result, presented in the Fig. 3, is consistent with the block diagram presented in Fig. 1. Using LOPHT, ARINC 653 dependent code is generated and linked to the component code generated by HEPTAGON. The resulting application is executed on POK OS [15].

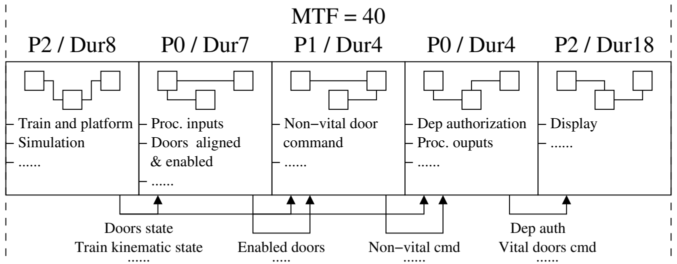

Figure 3. The partitional scheduling result of LOPHT.

IV. CONCLUSION

We presented the work conducted in the FSF project to handle mixed criticality. We used a synchronous design framework to implement a simplified signaling application and to deploy it on a partitioned OS.

We are continuously working towards a better integration of the tools composing the framework.

In the passenger exchange use case, mixed criticality resides at the application level, or even at function level, rather than the system level. On the other hand, the approach proposed in IMA and ARINC meets the needs of a system integrator. The main constraint highlighted by this case study is that there may be, even within a single system function, many communications between its vital and non-vital subcomponents. When generalized to the whole set of system functions, this pattern induces a large number of communications between the vital and non-vital parts. Furthermore, if we want to preserve the synchronous semantics (e.g. no additional delay) the number of windows may explode. The overall cost of communications and context-switch become prohibitive for systems global performance. Executing mixed-critical signaling applications on the same platform remains a challenging problem considering the state of the art in real-time operating systems.

Finally, the vital/non-vital dichotomy traditionally used in signaling application proved to be insufficient with respect to the operational availability of the system. It would be more appropriate to consider at least three levels, safety-critical, mission-critical, and non-critical, and to exploit this information in the partitioning and scheduling.

ACKNOWLEDGMENT

This research work has been carried out under the leadership of the Technological Research Institute SystemX, with partial support from the French Program "Investissements d'Avenir". The research leading to these results also received funding from the ARTEMIS Joint Undertaking project EMC2 (grant agreement 621429) and from the French Ministry of Industry.

REFERENCES

[1] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice, "Lustre: A declarative language for real-time programming," in *Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, ser. POPL '87. New York, NY, USA: ACM, 1987, pp. 178–188. [Online]. Available: http://doi.acm.org/10.1145/41625.41641

[2] A. Benveniste, P. Le Guernic, and C. Jacquemot, “Synchronous programming with events and relations: The signal language and its semantics,” *Sci. Comput. Program.*, vol. 16, no. 2, pp. 103–149, Sep. 1991. [Online]. Available: http://dx.doi.org/10.1016/0167-6423(91)90001-E

[3] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. L. Guernic, Robert, and D. Simone, “The synchronous languages 12 years later,” in *Proceedings of The IEEE*, 2003, pp. 64–83.

[4] S. Baruah, “Semantics-preserving implementation of multirate mixed-criticality synchronous programs,” in *Proceedings of the 20th International Conference on Real-Time and Network Systems*, ser. RTNS ’12. New York, NY, USA: ACM, 2012, pp. 11–19. [Online]. Available: http://doi.acm.org/10.1145/2392987.2392989

[5] E. Yip, M. M. Y. Kuo, P. S. Roop, and D. Broman, “Relaxing the synchronous approach for mixed-criticality systems,” in *Proceedings of the Work in Progress Session of the 20th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS’14 WiP)*. IEEE, 2014.

[6] P. Aubry and P. Le Guernic, “On the desynchronization of synchronous applications,” in *11th International Conference on Systems Engineering, ICSE*, vol. 96. Citeseer, 1996.

[7] P. Caspi, A. Curic, A. Maignan, C. Sofronis, S. Tripakis, and P. Niebert, “From simulink to scade/lustre to tta: A layered approach for distributed embedded applications,” in *Proceedings of the 2003 ACM SIGPLAN Conference on Language, Compiler, and Tool for Embedded Systems*, ser. LCTES ’03. New York, NY, USA: ACM, 2003, pp. 153–162. [Online]. Available: http://doi.acm.org/10.1145/780732.780754

[8] G. Delaval, A. Girault, and M. Pouzet, “A type system for the automatic distribution of higher-order synchronous dataflow programs,” *SIGPLAN Not.*, vol. 43, no. 7, pp. 101–110, Jun. 2008. [Online]. Available: http://doi.acm.org/10.1145/1379023.1375672

[9] A. Cohen, L. Gérard, and M. Pouzet, “Programming parallelism with futures in lustre,” in *Proceedings of the Tenth ACM International Conference on Embedded Software*, ser. EMSOFT ’12. New York, NY, USA: ACM, 2012, pp. 197–206. [Online]. Available: http://doi.acm.org/10.1145/2380356.2380394

[10] M. Marouf, L. George, and Y. Sorel, “Schedulability analysis for a combination of non-preemptive strict periodic tasks and preemptive sporadic tasks,” in *Proceedings ETFA’12*, Kraków, Poland, Sep. 2012.

[11] P. Caspi, A. Curic, A. Magnan, C. Sofronis, S. Tripakis, and P. Niebert, “From Simulink to SCADE/Lustre to TTA: a layered approach for distributed embedded applications,” in *Proceedings LCTES*, San Diego, CA, USA, June 2003.

[12] C. Pagetti, J. Forget, F. Boniol, M. Cordovilla, and D. Lesens, “Multi-task implementation of multi-periodic synchronous programs,” *Discrete Event Dynamic Systems*, vol. 21, no. 3, pp. 307–338, 2011.

[13] T. Carle, D. Potop-Butucaru, Y. Sorel, and D. Lesens, “From dataflow specification to multiprocessor partitioned time-triggered real-time implementation,” INRIA, Rapport de recherche RR-8109, Oct. 2012. [Online]. Available: http://hal.inria.fr/hal-00742908

[14] T. Carle and D. Potop-Butucaru, “Predicate-aware, makespan-preserving software pipelining of scheduling tables,” *ACM Trans. Archit. Code Optim.*, vol. 11, no. 1, pp. 12:1–12:26, Feb. 2014.

[15] J. Delange, L. Pautet, and P. Feiler, “Validating safety and security requirements for partitioned architectures,” in *Proceedings of the 14th Ada-Europe International Conference on Reliable Software Technologies*, ser. Ada-Europe ’09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 30–43.