CISTER
Research Center in
Real-Time & Embedded
Computing Systems

Conference Paper

The CONCERTO methodology for model-based development of avionics SW

Andrea Baldovin
Alessandro Zovi
Geoffrey Nelissen*
Stefano Puri

*CISTER Research Center

CISTER-TR-150309

2015/06/22

The CONCERTO methodology for model-based development of avionics SW

Andrea Baldovin, Alessandro Zovi, Geoffrey Nelissen*, Stefano Puri

*CISTER Research Center

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: baldovin@math.unipd.it, grrpn@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

The development of high-integrity real-time systems, including their certification, is a demanding endeavour in terms of time, skills and effort involved.This is particularly true in application domains such as the avionics, where composable design is to be had to allow subdividing monolithic systems into components of smaller complexity, to be outsourced to developers subcontracted down the supply chain. Moreover, the increasing demand for computational power and the consequent interest in multicore HW architectures complicates system deployment.For these reasons, appropriate methodologies and tools need to be devised to help the industrial stakeholders master the overall system design complexity, while keeping manufacturing costs affordable.

In this paper we present some elements of the CONCERTO platform, a toolset to support the end-to-end system development process from system modelling to analysis and validation, prior to code generation and deployment. The approach taken by CONCERTO is demonstrated for an illustrative avionics setup, however it is general enough to be applied to a number of industrial domains including the space, telecom and automotive. We finally reason about the benefits to an industrial user by comparing to similar initiatives in the research landscape.

The CONCERTO methodology for model-based development of avionics software

Andrea Baldovin[1], Alessandro Zovi[1], Geoffrey Nelissen[2] and Stefano Puri[3]

[1] Department of Mathematics, University of Padua
via Trieste, 63 - 35121 Padua, Italy
{baldovin,azovi}@math.unipd.it

[2] CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto
Rua Dr. António Bernardino de Almeida 431 - 4249-015 Porto, Portugal
grrpn@isep.ipp.pt

[3] INTECS
via Umberto Forti, 5 - 56121 Pisa, Italy
stefano.puri@intecs.it

Abstract. The development of high-integrity real-time systems, including their certification, is a demanding endeavour in terms of time, skills and effort involved. This is particularly true in application domains such as the avionics, where composable design is to be had to allow subdividing monolithic systems into components of smaller complexity, to be outsourced to developers subcontracted down the supply chain. Moreover, the increasing demand for computational power and the consequent interest in multicore HW architectures complicates system deployment. For these reasons, appropriate methodologies and tools need to be devised to help the industrial stakeholders master the overall system design complexity, while keeping manufacturing costs affordable.
In this paper we present some elements of the CONCERTO platform, a toolset to support the end-to-end system development process from system modelling to analysis and validation, prior to code generation and deployment. The approach taken by CONCERTO is demonstrated for an illustrative avionics setup, however it is general enough to be applied to a number of industrial domains including the space, telecom and automotive. We finally reason about the benefits to an industrial user by comparing to similar initiatives in the research landscape.

Keywords: Model-based engineering, CONCERTO, IMA, ARINC 653, partitioned multicore

1 Introduction

The behaviour of real-time software must be proved correct not only in the functional dimension, as it is the case for any software system, but also in the time dimension to ensure proper interaction with the physical world. The requirements that need to be met by those systems extend much beyond *functional behaviour*, to include timeliness, energy consumption, robustness. Those

additional requirements are globally understood to designate *extra-functional properties*. Interestingly, while functional behaviour embodies the logic required to solve a specific problem and lends itself to *reuse*, extra-functional properties are usually strictly contingent on the specific deployment platform and the stipulated execution conditions.

Because of their inherent complexity, the delivery of high-integrity real-time systems is seldom the result of a single player following a strictly sequential development process from requirements to deployment. Rather, it is the result of the collective effort of numerous subcontractors, which eventually needs to be *incrementally* integrated into a coherent system and ascertained to satisfy the applicable certification requirements. To enable the involvement of multiple development parties, the system needs to be designed in a way that favours its decomposition into elementary parts. This buys into the principle of *compositionality*, which considers the system top-down to describe it as a function of its constituting components. Compositionality at system level is facilitated by *composability* at lower levels of the system stack, to guarantee that the behaviour of individual components as determined in isolation stays the same upon composition in the final system.

Unfortunately, while compositionality and composability of functional properties are comparatively easy to achieve, ensuring composable behaviour in the extra-functional dimension is a more challenging problem. This is because of the complexity of precisely characterising the interactions among different entities executing in the HW/SW stack. Consequently, designing and assessing extra-functional properties becomes harder and the resulting systems tend to represent very specific solutions tailored to the problem they address, often based on specific industrial practices. In this situation, dimensioning a system very precisely becomes impractical and over-provisioning emerges as the only solution to meet the wished integrity levels at the expense of manufacturing and operation costs.

Model-driven engineering. Research conducted in the context of the CHESS project[1] suggested that resorting to a *reference software architecture* in combination with a model-based component-oriented development process can be a very effective strategy for the industrial development of real-time embedded software [11]. A reference software architecture can be thought of as a template architectural blueprint defining the methodology and architectural practices that form a common baseline solution for software systems of a specific domain. A reference architecture exhibits at least the following traits, as defined in [10].

1. A component model, to define software building blocks encapsulating pure functional and reusable logic.
2. A computational model, to map the entities in the component model to a framework of analysis techniques.
3. A programming model, to define a limited subset of programming language constructs that can be automatically generated from the component model.

[1] `http://www.chess-project.org/`

4. An execution platform capable of preserving the properties ascertained at the analysis stage and monitoring their possible violation.
5. A development process inspired by Model-Driven Engineering (MDE) [14], to promote a disciplined approach capable of mastering complexity by enforcing separation of concerns [5].
6. The provisioning of domain-specific views on the system to satisfy different stakeholders.

The methodology developed by CONCERTO builds on the foundation of the CHESS research, embracing a model-based component-oriented approach to system design, a model-based analysis framework with back propagation and automatic generation of application code.

The avionics use case. Modern avionics systems are designed around an integrated system architecture known as Integrated Modular Avionics (IMA) [12], where several software subsystems are allowed to share the available physical resources, provided that some constraints on space and time isolation are respected. This is imposed mainly for safety reasons, i.e., to avoid faults in any executing application, called partition, to propagate to the whole system. IMA architectures are hierarchically organised into two layers: at the top level, execution requirements for partitions are considered to determine the schedule of the whole system. That schedule allocates time slots to partitions, within the boundaries of which every partition is guaranteed to execute in isolation from others. Within any partition a local scheduler is in charge of executing application tasks with no knowledge needed about the system-level schedule.

Interestingly, partitioned design naturally promotes composability, which in turn enables incremental development and verification of the system. However, when a reference architecture is considered, some specific requirements from the avionics domain need to be supported.

1. *Hybrid design approach.* The system design process usually proceeds top-down, starting from the definition of higher-level requirements and subsystems and mapping them down to cohesive functional components of finer granularity. However, designing a hierarchical system demands additional support bottom-up to define partitions as aggregates of individual components. The component model needs therefore to support multiple levels of abstraction and aggregation.
2. *Semi-automated schedule design.* As a matter of practice, the generation of the system level schedule is performed by the system integrator taking into consideration the execution requirements of the individual partitions. This risks to be a time-consuming and error-prone activity in the absence of appropriate supporting tools. For example, if the timing requirements of one partition were to change, this potentially affects the whole schedule. However, as discussed later in Section 3.2, this job need not be carried out by hand: rather, a number of execution constraints may be inferred from the execution needs of the individual tasks, and a valid system-level schedule can be semi-automatically built.

3. *Multicore HW architectures.* Although the IMA architecture addresses single-core systems only, the avionics industry has shown some interest for the recent advances in HW architectures and the potential of multicores in particular. Among the possible multiprocessor configurations, the partitioned approach seems to be the most interesting in this context, since inter-processor interference can be contained and controlled to some extent by the system designer. A partitioned system therefore lends itself more easily to incremental verification and certification. However, little industrial experience is available from real-world systems running on multicore processors, which motivates the need for suppport tools.
4. *Model analysis.* Assessing the behaviour of partitioned systems after their deployment may reveal itself a hard job in the absence of suitable techniques. Analysing the system model and verifying its validity early at design time with automated tools brings the advantage of promptly detecting any inconsistencies and identifying the parts of the architecture to be revised.

In the remainder of this paper we present the solutions devised within CONCERTO to meet the challenges above. The paper is organised as follows: Section 2 gives an insight on existing research on modelling tools and execution platforms for partitioned systems supporting the ARINC 653 standard for the avionics [1]. Section 3 presents our methodology and toolset, starting from the CONCERTO component model, then illustrating the added-value tools provided for system configuration, analysis and code generation. Finally in Section 4 our approach is compared to MultiPARTES, a similar research initiative.

2 Background

This section introduces recent research dealing with model-driven methodologies and platforms for avionics software. In a first time we look at MultiPARTES, a research project that addresses model-driven development in the context of partitioned architectures. Then, we discuss an execution platform for IMA that proved to foster time composability in avionics applications.

MultiPARTES. The MultiPARTES research project[2] addressed the development of mixed-criticality embedded systems on heterogeneous multicore platforms. Its approach was developing a model-based software development methodology and the related toolset for building partitioned applications on top of the XtratuM hypervisor [7].

According to the MultiPARTES methodology [13], a system model is first created, later elaborated by a partitioning tool and finally validated by a number of tools. Those tools check the correctness with respect to different criteria, for example response time. The output of the partitioning tool is a deployment model which is amenable to code generation and can be deployed to a dedicated

[2] http://www.multipartes.eu/

platform. The main component of the platform is the XtratuM hypervisor which stands in between the multicore hardware and the partitions. Each of them contains a subsystem which comprises a real-time operating system and the software applications.

The system model consists of three different models: (i) the application model represents the functional description of the system, enriched with non-functional attributes related to criticality, timing and partition configuration; (ii) the platform model represents the hardware description and the OS available for deployment; (iii) the partitioning restriction model represents the relations between the application and platform models and specifies partitioning constraints to be taken into account by the partitioning tool to generate the deployment model. Only the application model conforms to the UML2 metamodel and the MARTE profile [17], whereas the others are defined by non-standard domain-specific languages.

The XtratuM hypervisor uses paravirtualization, which means that its operations are as close to the target hardware as possible. Any OS to be deployed on top of this kind of hypervisor needs to be modified to replace some privileged part of the Hardware Abstraction Layer with the corresponding calls to the hypervisor. The design of XtratuM borrows the concept of partition from IMA, and the cyclic scheduling of partitions and the inter-partition communication mechanisms from ARINC 653. However while an IMA partition is in charge of managing the processes inside it, an Xtratum partition – which is a virtual computer – delegates such management to the virtualised operating system.

PROARTIS_sim and TiCOS. The avionics case study considered in CONCERTO needs to be deployed on a PowerPC (PPC) architecture, which is widely adopted by leading manufacturers in that domain [8]. To this end we consider the execution platform developed within the scope of the EU-FP7 PROARTIS project[3]. This includes (i) PROARTIS_sim, a highly configurable and timely accurate PPC-750 simulator developed by Barcelona Supercomputing Center, and (ii) TiCOS [3], developed at the University of Padova and providing basic OS services and implementing the ARINC 653 API on top of it. PROARTIS_sim can be deployed to build a partitioned multicore configuration as a number of identical single-core replicas executed side-by-side, with no direct communication allowed between applications running on different nodes. TiCOS meets the usual time and space partitioning requirements of IMA systems by implementing a significant part of the ARINC 653 API. At the same time it shows reduced execution jitter and enforces time composability. When running on a partitioned multicore HW configuration, one TiCOS replica is deployed on each core. Those multiple instances are completely unaware of each other and all provided API services have only local effects.

Considering the sources of unpredictability present in typical HW and SW used for embedded systems, [18, 19] showed that the execution stack made up of PROARTIS_sim processor and TiCOS causes much reduced interference on

[3] `http://www.proartis-project.eu/`

the execution of user-level applications and further makes them amenable to probabilistic analysis of worst-case execution time. Those results come in handy in our context to prove that it is possible in fact to preserve the extra-functional properties defined at the modelling level, and execution time guarantees in particular, down to the execution stack, so long as the latter is well-behaved versus the applications running on top of it.

3 The CONCERTO methodology

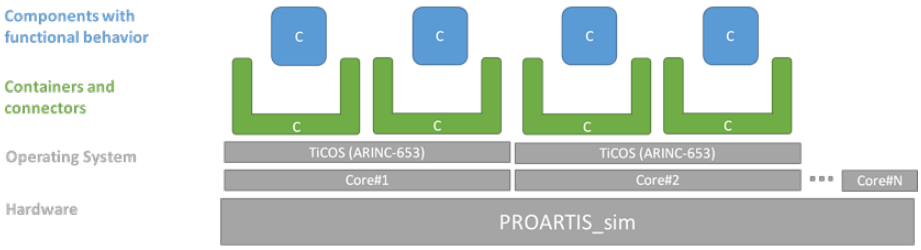

Fig. 1. Decomposition of an avionics system developed with CONCERTO .

In this section we describe how it is possible to model an avionics use case like the one shown in Figure 1 with CONCERTO. In doing this we introduce the constituting parts of the platform starting from the component model (Section 3.1) and the additional plugins assisting system deployment (Section 3.2). We then present the analysis we are able to carry on the system model (Section 3.3) and the technology enabling automatic code generation (Section 3.4).

3.1 Modelling

MARTE is the language chosen by CONCERTO for modelling extra-functional properties, and timing in particular. Although MARTE does not provide explicit guidelines or stereotypes for modelling ARINC 653 partitions, it defines a number of fine-grained stereotypes that can be collectively used to model partitions and their properties. CONCERTO uses those artifacts to model the spatial and temporal isolation requirements of partitions.

Partition modelling. In CONCERTO the spatial isolation property of a partition is modelled by using the «MemoryPartition» MARTE stereotype. The stereotype is defined as a virtual address space ensuring that only resources associated to it can access and modify it. This is compliant with the definition given by ARINC 653. Concerning temporal isolation, the runtime of each partition, i.e. its virtual processor, is modelled by using both the MARTE «SwSchedulableResource» and «ProcessingResource» stereotypes. The former makes it possible to

Fig. 2. Functional Partition in CONCERTO.

represent the virtual processor as a schedulable resource executing in parallel with others. The latter is used to represent the logical entity capable of scheduling and executing the tasks within the partition running on the virtual processor. The virtual processor must then be mapped to a physical processor, annotated by the «HWProcessor» MARTE stereotype.

Beside temporal and spatial isolation, we are also interested in modelling the functional behaviour of partitions. This requires support for modelling hierarchical components, which is provided by UML through composite structure diagrams. The user of CONCERTO is given the possibility to define partitions either prior or after the definition of their owned sub-components, in a top-down or a bottom-up fashion respectively. In practice however, application design influences the partitioning decisions taken at system level. Therefore, from the methodological point of view, partition modelling builds upon the functional specification of applications. For this reason, partitions are initially modelled as empty composite components, acting as wrappers of application-specific subcomponents that implement the required functionality. Partitions further segregate the behaviour of child components that should not be visible to others deployed in different partitions. Instead, the behaviour to be provided to or required from the external world is exposed by the partition through specific interaction points, i.e. UML provided and required ports respectively. Thus, a partition does not provide any behaviour on its own, but rather delegates all the exposed and required functionality to its internal components. All the constraints above are enforced at model level by means of a dedicated UML Component stereotype called «FunctionalPartition», which represents the functional concern of CONCERTO partitions (Figure 2).

In our solution the «MemoryPartition» and «FunctionalPartition» stereotypes are applied together on the same component to induce the spatial isolation property. A possible alternative would be to use the «MemoryPartition» stereotype for the hardware modelling, as a distinct entity associated to the virtual processor. In our current implementation, however we have discarded that choice to keep hardware modelling simpler. Finally, the partition is allocated to a virtual processor through the «Assign» MARTE stereotype (Figure 3).

Partition decoration. For each virtual processor timing information can be specified to enable timing analysis. A table-driven schedule can be associated to a MARTE «HWProcessor» stereotype by attaching a «Scheduler» entity to it. The latter is equipped with (i) a schedule parameter that describes the absolute ordering of the referenced virtual processors, and consequently of the partitions;

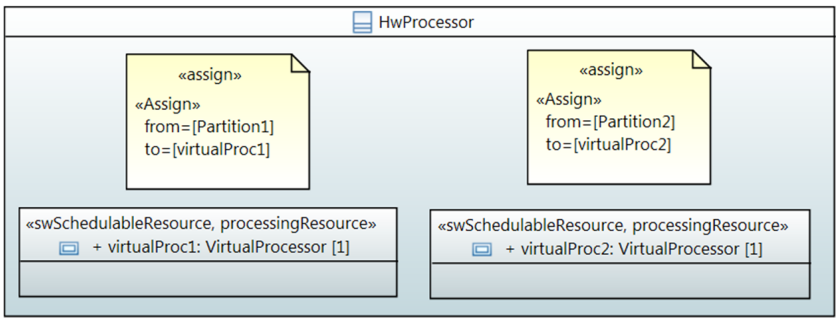

Fig. 3. Virtual Processor in CONCERTO.

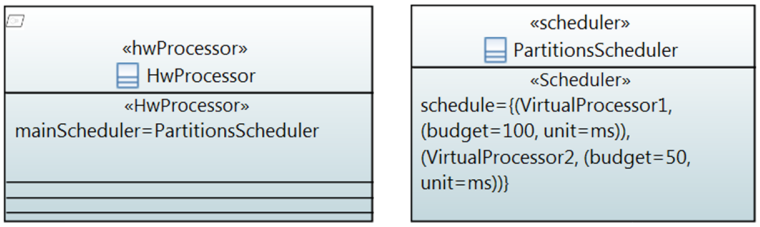

Fig. 4. Partition timing decoration in CONCERTO.

(ii) the budget assigned to each virtual processor, computed as a measure of the processing resource utilisation (Figure 4). The utilisation can be either provided top-down by the system designer or alternatively computed bottom-up by the schedule generation tool described in Section 3.2.

3.2 Deployment

Deployment is of paramount importance to ensure system feasibility and correctness. At this stage the software architecture, as designed in the modelling phase, meets the constraints and the limitations imposed by real target platforms. Moreover, opportunities for optimisation may emerge at this stage, which is fundamental to limit over-provisioning of system resources and contain costs. When considering partitioned multiprocessor avionics systems, decisions have to be taken about where and when partitions should execute. The former is the problem of finding an allocation of partitions to cores, the latter requires to draw a top-level schedule for partitions.

Allocation to cores. By definition, task migration across cores is not allowed on a partitioned multicore system. In our understanding partition migration should be avoided as well, since this would have severe implications on the achievable time and space isolation. In fact task migration is the most notable drawback of global scheduling approaches because of the HW state pollution caused and

the consequent additional execution time overheads introduced. Those undesired effects would be even amplified in a scenario where partitions, as collections of tasks, were allowed to migrate across processors.

Creating the partitioning, i.e. deciding where each task or partition will execute, is known to be a NP-hard problem as an instance of the general graph partitioning problem [6]. In fact, sub-optimal choices at this stage may easily impair system performance and in the worst case even its feasibility. CONCERTO provides the technology to semi-automate this step by proposing a partitioning for the system based on the execution requirements of the partitions defined in the modelling step. This is achieved by implementing a modified version of the worst-fit bin packing heuristic[4]. Modifications to the base heuristic need to be considered to account for the inter-partition communication protocol allowed by ARINC 653 [1]. More precisely, we need to take care of the following implicit constraint emanating from the component model: when two partitions are meant to communicate via sampling or queuing ports, then they must be allocated for execution on the same physical processor. This is enough to generate a valid partitioning, whereas its efficiency, expressed as the minimal number of processing units needed to schedule the system, directly depends on the bin packing technique used and the specific task set. Alternative partitioning choices might have been considered to execute a partition on more than one processor. However such configurations, while promising better theoretical performance, allow concurrent execution of tasks within a partition. Consequently additional contention on shared hardware resources and interference are introduced, eventually breaking the time composability property enforced between the layers of our execution stack. This would make it harder – if not impossible at all – to precisely characterize the execution time of user applications, i.e. partitions, hampering their assembly into the final system. For this reason those setups are not considered in this discussion. Similarly, inter-partition communication across different processors is not considered, as this mandates a careful assessment of how shared memory affects the time analysability of the system. Communication between partitions residing in the same node instead is dealt with as prescribed by IMA for single-core platforms.

Schedule generation. ARINC 653 prescribes a round-robin policy with flexible quanta for the top-level scheduling of partitions, which substantially corresponds to drawing a table-driven schedule offline. This is a critical step, since (i) the number of partitions may grow considerably in a real-world system, (ii) a wrong schedule may easily lead to violations of process deadlines and (iii) any change in the requirements of any partition may potentially impair the validity of a previously computed schedule. CONCERTO provides a plugin to semi-automate the schedule generation process by computing a base schedule for the system, that can be later modified and refined by the system designer according to her specific needs. In fact, the deadline/period and worst-case execution time (WCET) parameters of the individual tasks indirectly constrain the top-level schedule of the

[4] Any user-defined heuristic may be implemented with little impact on the platform.

system and restrict the search space of the problem. The tool assumes the mapping between tasks and partitions is known: this is not unrealistic in fact, since partitions are designed to encapsulate cohesive functional logic and are treated as atomic blocks when their development is outsourced. In case the individual utilisation of partitions has not been specified explicitly by the system designer as explained in Section 3.1, an initial schedule for the system is built as follows. Initially the Major Frame (MAF), i.e. the system hyperperiod in ARINC speak, and the total slack available in the system are computed. Then the execution demand of each partition is calculated at any time instant of the MAF, by considering the request bound function (rbf) of the tasks in the partition. We then look for a couple of values (budget, period) such that their ratio, corresponding to the utilisation of the partition, is minimised. After repeating this process for all partitions, the earliest schedule for the system can be easily drawn by sorting partitions according to the earliest-deadline-first (EDF) policy. However, unless the system is fully utilised, some slack is available in the system and has not been assigned by the algorithm so far. This reserve of computation resources is very useful in practice, to serve as a buffer to accommodate overruns of modest magnitude. Our plugin redistributes the remaining slack uniformly across those partitions for which utilisation was not specified by design. This process starts from the end of the MAF and proceeds backwards until either there is no remaining slack in the system or assigning more slack would make it unfeasible as a consequence of violating some deadlines.

In the case of communication between partitions with blocking semantics, the ARINC 653 specification does not prescribe any specific execution ordering of producer and consumer partitions. Rather, in the event of writes on a full buffer or reads from an empty buffer, execution is blocked and the blocking condition is re-checked on the next scheduling slot. Although inverting the execution ordering of reader and writer would be a more efficient solution, one must be careful to not compromise system feasibility. For this reason this kind of optimisation is left to the user at the moment. More generally, the user is given the chance to modify the proposed schedule to meet any specific need that an automated process can hardly capture. Also, slack distribution can be optimised to allocate extra-time for any special needs, e.g. to partitions performing the most critical operations. Validation of the new schedule is performed in the later step of analysis and back propagation of results to the user model.

3.3 Analysis and back-propagation

Once the deployment of the partitions done and their schedule generated in a semi-automated manner, a timing analysis of the processes mapped on each partition is performed. The timing analysis computes the worst-case response time (WCRT) of each process, which can then be compared against the process deadlines in order to assert their schedulability. If a deadline should not be met, that result is back propagated to the component model. The system integrator can then use this information to revise the component model or tailor the partition deployment in a way that would increase the partition supply and hence

allow all the processes to eventually respect their deadlines. The timing analysis of the system model is thus a critical step in the design of embedded software with timing requirements as its results may initiate multiple iterations on the description of the application model.

The timing analysis is performed using an augmented version of MAST[5], a schedulability and timing analysis tool developed and maintained by the University of Cantabria. CONCERTO extended MAST to model IMA architectures and compute the worst-case response time of processes running in partitions. To that end, the input model of MAST has been improved in order to represent multicore processors as well as partitions, their mapping and their generated schedule. Those inputs are provided to the extended version of MAST directly from the result of the deployment phase by means of a model-to-text transformation based on Acceleo[6], an engine implementing the MOFM2T specification [16] defined by the Object Management Group (OMG).

From an analysis viewpoint, MAST did not support the analysis of processes running in IMA partitions. MAST was thus extended using results of a similar problem, namely the hierarchical scheduling problem, which has been extensively studied in the literature [2, 4]. The state-of-the-art on hierarchical scheduling provides techniques to compute the WCRT of periodic or sporadic tasks to be scheduled using a fixed priority scheduling algorithm within a partition characterised by a given budget and replenishment period. Note however that none of the related works considers periodic tasks with release offsets. Yet, CONCERTO's component model is generic enough to model both periodic and sporadic processes and, in conformity with the ARINC 653 specification, periodic processes can be associated a release offset. Due to the lack of existing timing analysis for IMA systems composed of periodic tasks with offsets, the timing analysis implemented in the extended version of MAST is based on [4] thus assuming all the tasks to be sporadic. Although restrictive and pessimistic, this assumption provides a safe upper bound on the worst-case response times, even for systems that are partially or completely composed of periodic tasks. Yet, in an attempt to improve the accuracy of the results, researches are currently pursued in the CONCERTO project to extend the analysis to IMA partitions composed of a mix of sporadic and periodic processes with or without offsets.

3.4 Code generation and execution

After the user has performed all desired analyses and the model-based description of the system has been validated, most of the conceptual work has been already performed to shape the system and the effort shifts to implementation and deployment on a real target. At this point a number of heterogeneous technologies come into play to take the system blueprint and implement it in the form of human-readable code as a first approximation, then translating it into machine code for execution on a HW platform. Unfortunately, the abstraction

[5] `http://mast.unican.es`
[6] `http://eclipse.org/acceleo/`

gap existing between those architectural layers risks to invalidate the guarantees so hardly obtained on the modelling side and eventually undermines the whole development process. Additionally, another serious obstacle to system development comes from execution on top of unpredictable HW and OS layers, i.e., in the presence of components whose behaviour and interference are hard to characterise and consequently make it difficult to preserve extra-functional properties such as execution timing. Neglecting these considerations early in the development process may inflate the verification process and manufacturing costs consequently, or in the worst case lead to malfunctioning of the delivered system. The methodology defined by CONCERTO suggests how to mitigate both problems, i.e. the preservation of extra-functional properties at lower layers of the architecture and execution on a predictable platform.

For the preservation of timing properties ascertained at the model level our toolset includes an automated code generation facility, following the approach successfully implemented in CHESS and illustrated in [9]. That work demonstrated that it is possible to preserve the semantics of constructs defined at the modelling level, i.e., by the component model, down to implementation and deployment by resorting to a programming model, that is a limited set of code archetypes in the programming language of choice . The automatic code generation approach is useful not only to retain the guarantees obtained at the model level, but it also helps reduce the development effort required in the event of multiple deployments or even of re-targeting to different HW architectures. This is because from one single model of the system any implementation can be automatically generated, provided that the mapping from the component model to a programming model has been specified. Automatic code generation is realised in CONCERTO by means of model-to-text transformations run with the support of Acceleo.

For what concerns predictability of the execution platform, CONCERTO decided to adopt PROARTIS_sim and TiCOS in its execution stack, that proved to cause reduced interference to user applications as explained in Section 2.

4 Discussion

The methods and tools presented in Section 3 address the avionics-specific requirements presented in Section 1. Specifically, hierarchical components enable hybrid top-down and bottom-up design, the schedule generation and partitioning tools support deployment on complex hardware, and timing analysis of the model is made possible by modifications to MAST. We now compare our approach to MultiPARTES, since the latter addresses similar concerns although starting from slightly different premises, i.e. the intent of supporting mixed-criticality systems and heterogeneous multicore architectures.

Methodology and modelling. Not surprisingly the methodologies defined by CONCERTO and MultiPARTES present significant similarities, as a consequence of sharing the common background of CHESS. The approach taken by

MultiPARTES requires an additional step to model in detail the HW configuration of choice via a platform view. In return, support is given to asymmetric multiprocessor (AMP) architectures, although their adoption in real-world avionics systems is still to come. Conversely, in CONCERTO there is no need for this facility since execution on a symmetric multiprocessor (SMP) is assumed. One advantage of CONCERTO is certainly its metamodel, which is fully compliant with MARTE and UML2, whereas MultiPARTES makes use of proprietary entities that may hardly fit together with standard specifications and tools.

Execution stack. Although both define a partitioned architecture, the execution stacks of CONCERTO and MultiPARTES differ significantly because of their different goals. While both are capable of ensuring the required degree of space and time isolation, whether to enforce it by means of a hypervisor or a partitioning kernel needs to be carefully evaluated. One advantage of the hypervisor approach over a standard OS is its capability of encapsulating large subsystems – including user applications and the OS running them – as black boxes, and to make the interactions with the HW platform transparent. Nonetheless, the paravirtualization implementation of XtratuM requires some effort to port the guest OS to a different execution environment, by transforming system calls into hypercalls and redirect them to the hypervisor. Moreover, the introduction of an additional layer in the execution stack is very likely to introduce new overheads and cause more interference to applications at the user level. On the other hand, if a partitioning kernel is deployed, full control can be retained on the deployed HW and system SW, whose interactions are known to highly affect time composability and system analysability in turn.

The XTratuM architecture silently tries to enforce time composability by providing low-jitter hypercalls and allocating partitions to virtual processors statically, thus limiting interference by creating isolated HW/SW silos. These are similar principles to those inspiring the development of TiCOS. However, one may argue that if the design of the guest OS has negative effects on the application side, the introduction of a hypervisor alone is not sufficient to solve those problems but rather moves them one layer below in the architecture. In fact, existing studies [15] confirmed by our research on TiCOS have shown how tight coupling between applications and the OS hopelessly complicates timing analysis. For these reasons putting TiCOS at the centre of the execution stack is a better solution, since it makes it possible to achieve true time composable execution.

Similarly, the possibility of assigning more than one virtual processor to one partition in XtratuM poses no limits on the kind of scheduling policy chosen within partitions. In particular, if global scheduling is chosen, the open issues related to the amount of interference generated by task migrations are still present and moved one step above in the architecture, from the hypervisor layer to the partition internals. The choice of pinning partitions to cores as in TiCOS instead enforces partitioned scheduling of applications, providing a realistic setup for multicore execution with guarantees.

	Target system	Metamodel	Execution stack		
			Architecure	Scheduling	Effects on time composability
CONCERTO	SMP	MARTE/ UML2-compliant	partitioning OS	strictly partitioned (1 partition on 1 core)	time-composable execution stack (TiCOS + PROARTIS_sim)
MultiPARTES	AMP	proprietary	hypervisor	any (1 partition on ≥ 1 vitual cores)	coupling between app and OS + hypervisor overheads

Table 1. Distinguishing elements of CONCERTO and MultiPARTES.

In conclusion, the approach to modelling partitioned systems advocated by CONCERTO and MultiPARTES is very similar, as summarized in Table 1. However, the reasons to prefer a virtualisation architecture need to be justified by specific needs, such as the execution on AMPs or the integration of systems with different criticality levels on the same machine. It is questionable whether these trends will dominate future avionics systems. Yet the known issues concerning time composability and scheduling of applications encountered in traditional OS design need to be addressed in both scenarios.

Acknowledgements. The authors are grateful to the people at Barcelona Supercomputing Center for their supply of PROARTIS_sim. This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and when applicable, co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within project UID/CEC/04234/2013 (CISTER Research Centre); also by FCT/MEC and the EU ARTEMIS JU within project ARTEMIS/0003/2012 - JU grant nr. 333053 (CONCERTO).

References

1. Aeronautical Radio, Inc.: ARINC Specification 653-1: Avionics Applicaiton Software Standard Interface (2003)
2. Almeida, L., Pedreiras, P.: Scheduling within temporal partitions: Response-time analysis and server design. In: Proc. of the 4th ACM International Conference on Embedded Software (2004)
3. Baldovin, A., Mezzetti, E., Vardanega, T.: A Time-composable Operating System. In: 12th WCET Workshop. OpenAccess Series in Informatics (OASIcs), vol. 23, pp. 69–80. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2012)

4. Davis, R., Burns, A.: Hierarchical fixed priority pre-emptive scheduling. In: Proc. of the 26th IEEE Real-Time System Symposium (2005)
5. Dijkstra, E.: On the role of scientific thought. In: Selected Writings on Computing: A personal Perspective, pp. 60–66. Texts and Monographs in Computer Science, Springer New York (1982)
6. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)
7. Masmano, M., Ripoll, I., Crespo, A., Metge, J.: Xtratum: a hypervisor for safety critical embedded systems. In: Proc. of the 11th Real-Time Linux Workshop (2009)
8. Moir, I., Seabridge, A., Jukes, M.: Civil avionics systems. Wiley-Blackwell (2013)
9. Panunzio, M., Vardanega, T.: Ada ravenscar code archetypes for component-based development. In: Reliable Software Technologies – Ada-Europe 2012, Lecture Notes in Computer Science, vol. 7308, pp. 1–17. Springer Berlin Heidelberg (2012)
10. Panunzio, M., Vardanega, T.: An architectural approach with separation of concerns to address extra-functional requirements in the development of embedded real-time software systems. Journal of Systems Architecture 60(9), 770 – 781 (2014)
11. Panunzio, M., Vardanega, T.: A component-based process with separation of concerns for the development of embedded real-time software systems. Journal of Systems and Software 96(0), 105 – 121 (2014)
12. Radio Technical Commission for Aeronautics: Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations (2005)
13. Salazar, E., Alonso, A., Garrido, J.: Mixed-criticality design of a satellite software system. In: Proc. of the 19th IFAC World Congress (2014)
14. Schmidt, D.: Guest editor's introduction: Model-driven engineering. Computer 39(2), 25–31 (2006)
15. Schneider, J.: Why you can't analyze RTOSs without considering applications and vice versa. In: Proc. of the 2nd WCET Workshop (2002)
16. The Object Management Group: MOF Model to Text Transformation Language, v1.0 (2008), `http://www.omg.org/spec/MOFM2T/1.0/`
17. The Object Management Group: UML Profile for MARTE: Modeling and Analysis of Real-time Embedded Systems (2011), `http://www.omg.org/spec/MARTE/1.1/`
18. Wartel, F. et al.: Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study. In: Proc. of the 8th IEEE International Symposium on Industrial Embedded Systems (SIES). pp. 241–248 (2013)
19. Wartel, F. et al.: Timing analysis of an avionics case study on complex hardware/-software platforms. In: Proc. of the 18th Design, Automation & Test in Europe Conference and Exhibition (DATE) (2015)