Model-Based Design, Analysis and Synthesis for TSP Multi-Core Space systems

Christophe Honvault*, Jérôme Hugues †, Claire Pagetti‡

* European Space Agency, Noordwijk, The Netherlands

† ISAE-SUPAERO, Université de Toulouse, Toulouse, France

‡ ONERA, Toulouse, France

***Abstract*—*Symmetric multiprocessing* (SMP) and *Time and Space Partitioning* (TSP) are two complementary paradigms for the design of multi-core-based aerospace systems. They impose new steps in the development process: capturing complex configuration attributes, analyzing their correctness – in particular their predictability – while guaranteeing performance. In this context, model-based techniques provide a well-suited framework to design, analyze and automatically synthesize those systems.**

In this paper, we report on a set of extenstions of TASTE to support SMP and TSP-based multi-core platforms. We first present the key architectural elements of these systems and then detail how these have been integrated as part of the code generation tool-chain. We then present experiments realized on two case studies and two hardware targets, both provided with the RTEMS operating system and XTRATUM hypervisor.

I. INTRODUCTION

A. Context

One major output of the European FP6 ASSERT project [PCD+10], [JDW12] is the TASTE (The ASSERT Set of Tools for Engineering) tool suite. This open-source tool chain is dedicated to the development of embedded real-time systems for the space domain. TASTE addresses the modelling, automatic code generation and deployment of distributed systems composed of heterogeneous software (models or source artifacts) and hardware components. From a collection of models capturing data, interfaces and deployment artifacts, TASTE provides automation of tedious and error-prone validation and integration tasks. TASTE is under continuous development and improvement by ESA and its partners, including ISAE-Supaero.

In parallel, multi-core processors have emerged as good candidates for the space domain. In particular qualified and hardened processors have been developed, such as the dual core processor GR712RC [COB16], and the quad core processor GR740 [COB17]. On top of these hardware, qualified executive layers, such as RTOS (real-time operating systems) or hypervisors, must be developed. Currently, no such qualified layer is yet available even though several studies are on going, among which we could mention RTEMS [OAR17], XTRATUM [MRC+09], pikeOS [SYS17].

B. Moving towards TASTE multi-core

During a one-year project funded by ESA, we evaluated potential extensions of TASTE to support multi-core in some pre-defined configurations. More precisely, we decided to focus on SMP (Symmetric MultiProcessing) and IMA (Integrated Modular Avionics) / TSP (Time and Space Partitioning) configurations. For that, we extended the TASTE environment with fundamental principles and that lead to the definition of design patterns. To validate the approach, we considered two use cases and implemented them first manually on the ZYNQ board [Xil01] and the LEON3 processors. Then, we extended TASTE to support these settings, and compared manual and TASTE-autocoded implementations.

Although the project focused on minimal TASTE extensions to support multi-core, the ideas can serve as a general approach for other frameworks as long as they share similar separation of concerns approach between high-level description of components, their combination and finally their deployment on top of an executive and hardware platform. We can mention in particular the Space Component Model [PV14] or ECOA [FCOB14].

In the following, we present TASTE mono-core, SMP and TSP principles (see section II). In section III, we detail the extensions of TASTE, both in terms of modelling with patterns and of code generation. The experimental part is described in sections IV and V. We end the paper with a related work section and a conclusion.

II. STARTING POINTS

The purpose of the project was to provide some multi-core extensions to TASTE. The targeted TRL was in the 2-3 range due to the duration (1 year) and the allocated effort. In the following, we review the context of the study.

A. TASTE

TASTE aims at automating the software development process of space-critical applications. It addresses the following system operational requirements: limited resources (memory, processor); real-time constraints (deadlines); applications of very different natures (control laws, resource management, protocols, fault detection); communication with hardware (sensors, actuators, FPGA); heterogeneous hardware (e.g. processors with different endianness); distribution over several physically independent platforms; may run autonomously for years; may not be physically accessible for maintenance (satellites).

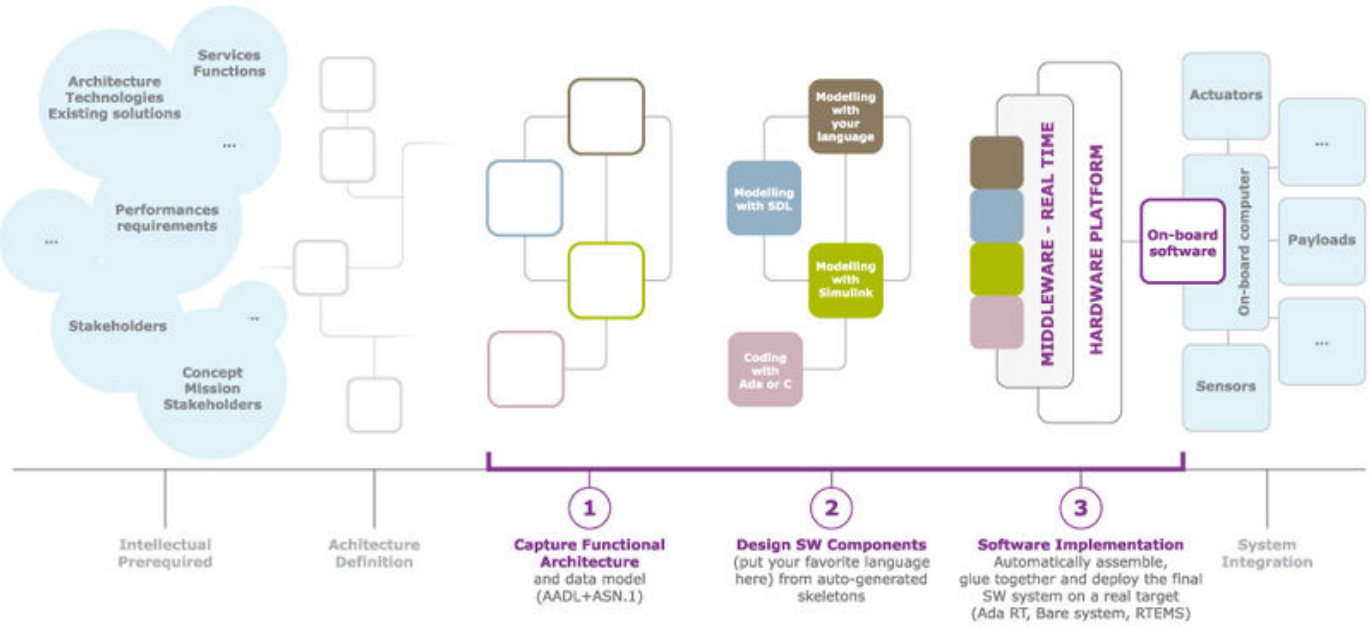

Figure 1: TASTE overview.

Hence, the quality of the generated code, along with the capacity to validate the system early in the design are of prime interest. The TASTE process is shown in Figure 1.

Step 1: functional architecture The philosophy is to let the user only focus on his functional code, letting him write it in the language of his choice, may it be a modelling language or a low-level implementation language. To achieve this, TASTE relies on the AADL [SAE12] and ASN.1 [asn02] text-based modelling languages that give sufficient power of expression to capture all the essential elements of a system that are required to generate the tasks, communication middleware and glue around the user functional code.

Step 2: internal function code Once a set of carefully selected system properties has been captured using these two languages, the core of the system's sub-components can be developed using C, Ada, SDL, SCADE, Simulink, or VHDL.

Step 3: automatic code generation TASTE tools are responsible for putting everything together, including drivers and communication means and ensuring that the execution at runtime is compliant with the specification of the system real-time constraints. Without any major overhead in the code, TASTE will produce binaries that can be directly executed on several supported targets: native Linux, Real-time Linux (Xenomai), RTEMS, and Ada bare-board targets.

TASTE evolved since 2007, it now supports several input languages for the functional part (SDL, SCADE, Simulink, C, Ada), multiple targets (Linux, FreeRTOS, Xenomai) and enables both massive code generation, but also scheduling analysis, simulation capabilities. These capabilities have been validated through ESA-funded studies, but also by partners.

B. System model

In the sequel, we consider that the platform is a multi-core composed with identical cores. The two targets used during the project were, as already mentioned, the ZYNQ board and the LEON dual-core processor.

Definition 1 (Multi-core): We consider a multi-core processor as a set of cores: $\mathcal{P} = \{P_i\}$, each core being uniquely identified over a contiguous interval from 0 to $n-1$.

The applications are also restricted per the definition of the Ravenscar computational model [BDV04] as follows.

Definition 2 (Application): An application is defined as a set of periodic and sporadic tasks $app = \{\tau_i = (A_i, C_i, T_i, D_i)\}$ where A_i is the activation pattern (e.g. periodic, sporadic), C_i is the WCET (Worst Case Execution Time), T_i is the period for periodic task or minimal inter-arrival time for sporadic task and D_i is the deadline. The deadline is equal to the period for periodic tasks and $D_i \leq T_i$ for sporadic tasks. A task τ_i can be unrolled as a set of jobs denoted $\tau_{i,j}$ – the j-th job of τ_i.

C. Definitions – reminder

Several programming paradigms exist for developing on a multi-core chip in order to deal with the parallelism. In the study, the partners investigated various strategies to leverage configurations with multiple processing units such as multi-core systems or multi-processor systems. We particularly analyzed requirements associated to AMP, SMP and TSP settings, most definitions of which can be found in the book by Hennessy and Patterson [HP90].

Asymmetric Multi-processor (AMP). In that case, the CPUs are not treated in the same way and individual processors can be dedicated to specific tasks at design time. Thus, individual functional processes are allocated to a separate core permanently and each core has its own operating system (multiple copies of the same operating system or a different one from core to core).

Symmetric multiprocessing (SMP) supposes some similar access to the shared main memory and all I/O devices from any core. A single operating system instance controls the processors and treats them equally. However, each processor may execute different programs, work on different data and has the capability to access shared common resources.

Time and Space Partitioning (TSP) / Integrated Modular Avionics (IMA) offers portable application software across an assembly of common hardware modules. Space domain refers to time and space partitioning while avionic domain refers to IMA. IMA concepts are partially captured in standards DO-297 [Rad] and ARINC653 [Aer97]. In the scope of this paper, we will restrict IMA/TSP to RTOS kernels with time and space isolation capabilities.

In the following, we restrict our paper to the TSP paradigm and the SMP case. Indeed, the space community has defined TSP-based building blocks, such as XTRATUM; as well as RTEMS SMP capabilities.

D. Multi-core and SMP/TSP-based RTOS for Space

1) RTEMS/SMP: Real-Time Executive for Multiprocessor Systems (RTEMS) [OAR17] is an open-source real-time operating system (RTOS) that supports a variety of open standard such as RT-POSIX and BSD sockets. It is used in space flight, medical devices, networking and many more embedded systems across a wide range of processor architectures including ARM, PowerPC, Intel, and SPARC.

RTEMS supports AMP through the execution of multiple instances of its kernel. As part of a recent study, Embedded Brains [CHS+14] provided a set of patch to make RTEMS SMP ready as part of the future RTEMS 4.12 release. The kernel has been significantly reorganized to support multiple cores, add necessary internal synchronization mechanisms to protect its internal data structures and provide SMP drivers. It relies on SMP extensions for RT-POSIX, along with the implementation of specific schedulers for SMP.

As part of this project, we focused on SMP configurations that take into account the affinity of a task, i.e. the core it is attached to. This means that the scheduling is partitioned.

Definition 3 (Configuration – SMP): An SMP configuration defines the affinity of each task, i.e. the core it is allowed to run on. More formally, it is a function $map_{app} : app \rightarrow \mathcal{P}, \tau_i \mapsto P_i$. The scheduling on a core then follows the FIFO per Priority (*SCHED_FIFO POSIX*) according to the Ravenscar rule.

2) XTRATUM *hypervisor:* XTRATUM is a TSP real-time hypervisor developed by Fentiss, a spin-off from the University of Valencia. In its current version, XTRATUM supports several targets, including ARM, PowerPC and the space dedicated processors LEON3 (GR712RC) and LEON4. XTRATUM is still being developed through various R&D projects (CNES, ESA, Thales Alenia Space and Airbus Defence and Space). XTRATUM is able to host various guest OS (Linux, RTEMS, LithOS) and has support for Ethernet, SpaceWire and 1553 devices as add-ons.

As a TSP kernel XTRATUM has builtin support for spatial and temporal isolation of the software partitions with scheduling capabilities inspired from ARINC 653. Static configuration is made via the XTRATUM Abstraction Layer (XAL). In this work, we consider bare-metal implementation of application.

An application is mapped as a set of *partitions* and a partition is defined by one or multiple *slots*, each with a start time and a length. Inside a slot, several tasks can be executed. Both the partitions slots and the schedule of tasks inside a slot are computed off-line, for instance with Xoncrete [BMR+10] the mapping and scheduling tool provided with XTRATUM. This off-line information is called *plan* in the XTRATUM terminology.

Definition 4 (Plan): A plan consists of:

- a *major frame* (MAF), the length of which is denoted *MAF_length*;
- a set of slots sl_i distributed over the cores and the MAF. A slot is defined as $sl_i = ([s_i, e_i], n_i)$ where s_i is the start time, e_i is the end time and n_i is the number of the core where the slot is allocated;
- a mapping of the jobs in the slots. Jobs are unrolled on the MAF and we know for all job $\tau_{i,j}$ in which slot sl_k it belongs to. We know moreover in which order are executed the jobs inside a slot.

A partition cannot be shared by two different applications.

XTRATUM and other TSP-based approaches are dedicated to periodic tasks sets. Thus sporadic tasks are either managed locally if the partition has a guest OS or handled as periodic tasks (e.g. periodically called with a condition of start).

III. SMP AND TSP PATTERNS

We detail in this section the patterns developed during the study.

A. Rationale

a) TASTE *modelling process:* We focused on SMP/TSP configurations and/or multi-core systems in a component-based, model-based approach. Usually, the modeling process is divided in multiple steps, or views.

In the context of TASTE, the *Interface View* defines the topology of the system. The Interface view lists the interface of the different functional blocks, and interconnects them. It is decorated with properties that indicate the execution pattern of the block: Periodic or Sporadic activation; Protected, and Unprotected concurrent access execution of subprograms. The *Deployment View* defines the supporting run-time platform, and the binding between the Interface view and these elements. Both views are modeled using AADLv2. Finally, the *Data View* contains the description of all the messages that are exchanged between functional blocks, using ASN.1.

From these views, a fourth model is generated: the Concurrency View that is plain AADLv2. This model is synthesized from the architectural parameters from the previous views: configuration of task and middleware building blocks, deployment of components on the various nodes and partitions.

b) Capturing SMP and TSP concerns: Configuring those systems has to preserve the initial separation of concerns found in TASTE, while extending the designer design space to new systems. Initially, TASTE only configured abstract applications, following definition 2.

The definition of extensions to TASTE followed a bottom-up approach. First, we identified elements to be configured at the target level. Then, we captured them as modeling *patterns* represented in the concurrency view. Finally, these patterns are abstracted away in the TASTE deployment view.

Definition 5 (Pattern): A pattern is a set of modeling artifacts and configuration elements combined, along with legality rules. Legality rules indicate correctness conditions of a pattern.

In the following, we provide a definition of the patterns, along with the rationale for their AADL/TASTE concurrency view representation.

B. Patterns for SMP

c) Definition: In this study, we focused on RTEMS/SMP configuration to extract the corresponding patterns. RTEMS/SMP builds upon the same configuration mechanisms as RTEMS mono-core: a set of configuration macros sets up the various resources; and an API configures them.

Let us note that RTEMS is a versatile RT-POSIX compliant RTOS. Yet, TASTE is restricting scheduling configuration parameters to those compatible with the Ravenscar profile, as they guarantee scheduling analyzability. Hence, we decided to opt for a multi-core extensions of Ravenscar, as detailed in [ZdlP13]. This preserves analyzability capabilities. The key configuration parameters are therefore:

- the cores of a processor following definition 1;
- the configuration following definition 3.

A *deployed* SMP application is therefore a tuple: $(app, \mathcal{P}, map_{app})$ (see figure 2). Let us note that by construction of this pattern, tasks are mapped to one and only one core. This guarantees determinism of the workload per core at run-time.

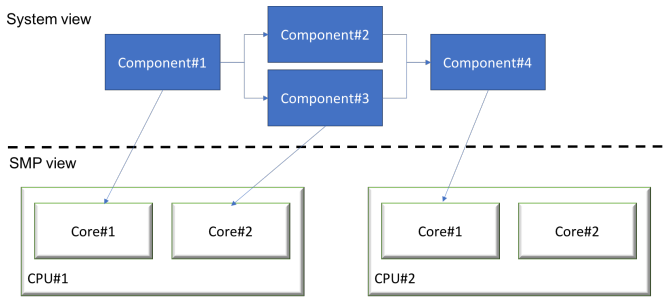

Figure 2: SMP pattern

d) AADL mapping: Turning this pattern into a valid AADL model requires extensions to AADLv2. We consider a multi-core processor as a regular processor, with multiple processor sub-components, each providing separate execution resources, with one property that uniquely identifies them.

Code 1 (AADL code - SMP definition):

```
processor implementation POSIX_CPU.Cores4
 subcomponents
   Cpu0 : processor a_core.impl { Core_Id => 0;};
   Cpu1 : processor a_core.impl { Core_Id => 1;};
   Cpu2 : processor a_core.impl { Core_Id => 2;};
   Cpu3 : processor a_core.impl { Core_Id => 3;};
end POSIX_CPU.Cores4;
```

Threads are bound to a particular core using using standard AADL binding (or allocation) mechanisms.

We note that this modeling pattern is incomplete: one cannot enforce syntactically that identifiers are unique and contiguous; or that all tasks are mapped to a specific core. These checks can be implemented separately, using model constraint languages.

C. Patterns for TSP

e) Definition: In this section, we revisit the definition of an application in the context of XTRATUM, from definitions 2 and 4. XTRATUM defines some legality rules to follow:

- An application is made of several plans, they are numbered as a consecutive set of integer values;
- plan $\#0$ is dedicated to configuration phase, other plans correspond to operational modes of the systems;
- slots inside a plan are numbered using consecutive integer values;
- all slots are executed at least once.

In addition to strict slot and plan configuration and identification, XTRATUM defines similar constraints on the memory configuration, so as to capture space partitioning: each slot is allocated a separated memory area.

f) AADL mapping: AADL representation for TSP targets relies on prescriptions from the ARINC653 annex [AC15], following ARINC653 concepts of partitions and memory regions (see figure 3). Adapting these patterns to XTRATUM requires little adaptation. We recall here the main steps:

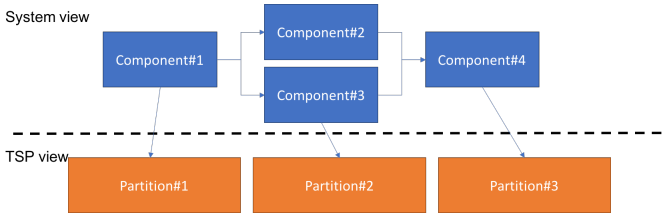

Figure 3: SMP pattern

The definition of memory partitions relies on default AADLv2 properties to define configuration parameters such as start address and length. Each partition is later attached to a memory partition using binding relationships.

Code 2 (AADL code - architecture and partition):

```
package ROSACE::XtratuM
public
  with ARINC653;

-- Memory components, representing a memory partition
  memory implementation  myram.stram
  properties
    Base_Address => 40000000;
    Byte_Count   => 4194304;
  end myram.stram;
-- [...]
```

The code 3 illustrates partition configuration, leverage ARINC653 annex of AADLv2: a partition is seen as a virtual processor, providing execution time.

Code 3 (AADL code - deployment):

```
-- Partitions
processor implementation leon3.xtratum_partitions
          extends leon3.xtratum
subcomponents
  P0 : virtual processor xtratum_partition.generic
  { ARINC653::Partition_Identifier => 0;
    ARINC653::Partition_Name       => "P0"; };
 [...]
properties
  ARINC653::Module_Schedule =>
    ( [Partition => reference (P0); Duration  => 2 ms;
       Periodic_Processing_Start => true;],
  [...]
```

Similarly to the SMP case, XTRATUM specific constraints on the identification of partitions, and mapping completeness are represented as constraints applied to the model using Resolute [GBC+14], a constraint language for AADL. Resolute allows one to define constraints a model has to satisfy in terms of logic predicates. As an illustration, the following snippet shows how to check that all virtual processors have a name and an identifier, or that processes are allocated to a memory segment.

Code 4 (TSP constraints – example):

```
check_arinc653_virtual_processors () <=
  ** "Virtual Processors are in processors" **
  forall (vp : virtual_processor) . true =>
     (exists (cpu : processor) . parent(vp) = cpu)
     and
     (has_property (vp, ARINC653::Partition_Identifier))
     and
     (has_property (vp, ARINC653::Partition_Name))

check_arinc653_process_memory (p : process) <=
   ** "Check  "p" is associated with a memory" **
   exists (segment : memory) . (is_bound_to (p, segment))
   and
   check_arinc653_memory_segment (segment)
```

Other constraints enforce consistency of TSP parameters and are presented in [HD17]. They check the following constraints:

- Each partition is associated with exactly one memory segment and one partition execution run-time.
- Each node specifies the partitions scheduling policy and executes each partition at least once during each scheduling period.
- Each task defines its scheduling characteristics (e.g. dispatch protocol, period, deadline).
- All queuing ports or buffers specify the maximum number of data instances they can store.

D. Composing TSP and SMP patterns

The previous patterns can be composed to model a time-space partitioned systems on top of a multi-core system. In this case, we use the following model elements: processors subcomponents as core in a multi-core CPU; virtual processors as logical partition in a TSP system.

Binding (allocation) relations define the association between all model elements. Hence, components are bound to partition, and then logical partition can be bound to a core inside a CPU. This pattern is a natural usage of the previous patterns, and illustrates composition of patterns (see figure 4).

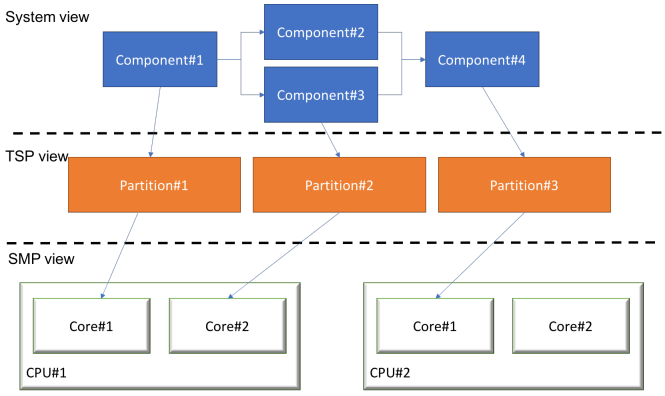

Figure 4: TSP & SMP patterns

All our contributions have been integrated at the level of the Concurrency View. The corresponding updates of the upper view is a future work activity, to be coordinated with ESA and other TASTE users, for instance in the scope of the H2020 ESROCOS project [AMW+17]. We will illustrate how we leveraged these in the next section.

IV. APPLICATION TO ROSACE

The ROSACE– Research Open-Source Avionics and Control Engineering – has been developed as a collaboration between ONERA, ISAE and Polytech Montréal and its initial specification has been published in [PSG+14]. Although of modest size, this controller is representative of real avionics applications by introducing typical characteristics such as a data-flow design or complex multi-periodic execution patterns.

The application is composed of 11 functions combining both functional parts as well as a mock up representing the aircraft. They run at different periods and exchange flight parameters and orders.

A. Implementation in XTRATUM

To run the tests, we chose to implement the environment, that is *Aircraft*, *engine* and *elevator*. We chose the base time unit to be in milliseconds. Since the period of some functions is $5ms$, there could be at most 5 partition slots during the $MAF = 5ms$. We compared two XTRATUM implementations. The first one was uni-processor for which we defined 3 partitions (P0, P1 and P2). In the second, we made a dual core schedule with 5 partitions, as shown in figure 6.

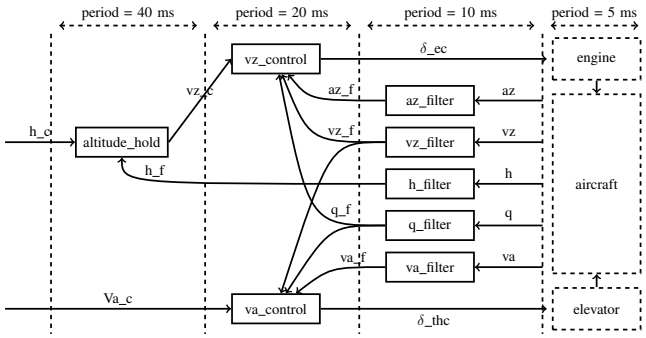

Figure 5: ROSACE architecture

*Code 5 (*AADL *mono-core and multi-core):*

```
thread Aircraft_Dynamics_T
— [..]
properties
  Dispatch_Protocol => Periodic;
  Period            => 5 ms; — 200 Hz
end Aircraft_Dynamics_T;

system implementation ROSACE_POSIX.Monocore
properties —
  Dispatch_Offset => 200 us applies to Software.H_filter;
  Dispatch_Offset => 300 us applies to Software.Az_filter;
— [..]
end ROSACE_POSIX.Monocore;

system implementation ROSACE_POSIX.Multicore extends
    ROSACE_POSIX.Monocore
subcomponents
  Hardware : refined to processor
      ROSACE::Hardware::POSIX_CPU.Cores4;

properties
  Actual_Processor_Binding => (reference (Hardware.Cpu1))
    applies to Software.Aircraft_Dynamics;

— [..]
```

The partitions communicate via *sampling ports* in these two configurations.

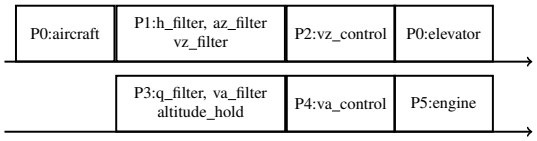

Figure 6: ROSACE partition schedule

This initial implementation of ROSACE has been done manually. As a complement, we modeled the system using TASTE-Concurrency View, following the patterns presented previously for the TSP case. This model captures the schedule of the various partitions. From this model, we could generate XTRATUM configuration tables using Ocarina, and check their equivalence. Hence, we assessed that the patterns capture all the necessary information.

B. Implementation in TASTE/ RTEMS

In the following, we detail the equivalent system modeled in TASTE, targeting RTEMS. We revisited the XTRATUM manual implementation to propose an equivalent model to be generated using the TASTE tool-chain.

We have associated a thread to each function and tested several hard-coded schedules. Following ROSACE specifications, each thread is periodic, with a period and dispatch offset that follows an off-line a priori scheduling configuration.

The code 5 shows both a (subset of) mono-core scheduling and the multi-core one done as part of the TASTE-Concurrency View. An interesting feature here is that the multi-core is done as an extension of the mono-core, where we adjust some scheduling parameters like the thread affinity or the dispatch offset[1]

[1]The model is available from: http://www.openaadl.org/aadlib.html

C. Validation

We extended the Ocarina component of TASTE in order to support these patterns, but also to generate XTRATUM, RTEMS and POSIX SMP configuration files to test the completeness of our patterns towards the final target.

In addition to support these patterns, we defined verification policies to ensure the validity of the configuration and deployment. For instance, they ensure that each function is bound to a CPU, partition or core; it is scheduled or that the definition of memory areas or schedule are sound.

As an additional validation step, we ran the code made manually and the one generated for the RTEMS/SMP and XTRATUM targets. The logs were compliant with the ROSACE checker, that checks the generated traces.

V. APPLICATION TO GCU CASE STUDY

The second case study is extracted from the French national project Spacify [ABB+10]. More precisely, we rely on the CNES case study detailed in [Spa10] that models the Payload Data Management System (GCU – Gestionnaire de Charge Utile in French) of a satellite. The system's mission is to apply commands from the ground to switch to a given mode and to confirm to the ground that requests have been correctly applied.

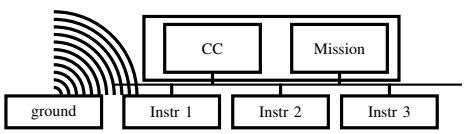

Figure 7: GCU architecture

Figure 7 shows the overall architecture. The GCU system is composed of *CC* (control / command module) and *Mission*. The GCU is connected via an embedded bus to three on board instruments (*I1, I2* and *I3*) and via telemetry to the ground platform (*ground*).

A. End-to-end behaviors

Compared to ROSACE, this use case is purely event triggered (thus composed of sporadic tasks) since the ground triggers the GCU by requests to activate or de-activate on-board instruments and to make some measurements. As a consequence, validating an implementation does not consist, as in ROSACE, to fulfill the tasks periods but to satisfy end-to-end responses to distribute and reply to *ground* requests.

CC is in charge of routing and formatting all the messages in the GCU.

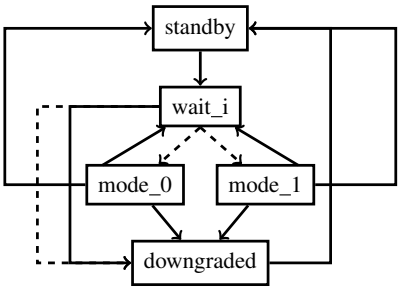

Figure 8: GCU mission architecture

Mission is in charge of activating or de-activating the instruments according to the ground requests and is implemented as an automaton as shown in figure 8. There are two types of arrows: plain when representing a request from the ground and dashed when a local decision is taken.

Initially the automaton is in *standby*. When the ground asks to move to *mode_i*, it transits through *wait_i* to start the according *Instrument*. If the start-up fails, the automaton moves to *downgraded*, otherwise to the asked *mode_i*. At any time, the ground can request to move to the other *mode*, to *downgraded* or to reset to *standby*.

We identified three scenarios of end-to-end behaviours that any implementation must comply with. The *scenario 1*, shown in figure 9, is composed of three steps.

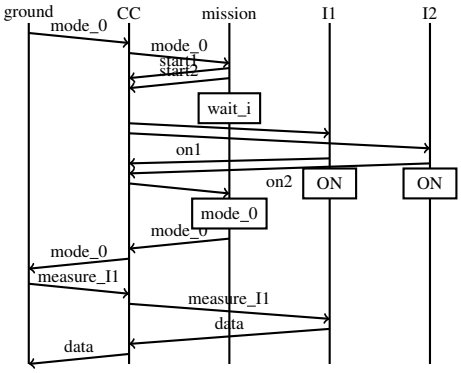

Figure 9: Scenario 1

First, the ground requests the platform to reach the configuration of *mode_0* where *I1* and *I2* must be activated. The second step is the switching of the platform from its current mode to the *mode_0*. In the scenario, the platform is supposed in *standby*, meaning all instruments are *OFF*. The component *CC* transfers the request to *mission* which tries to start the instruments. The orders to the instrument go through *CC* and in the scenario, the instruments do not encounter any problem and turn on. They inform *mission* of their status and the latter can now reach *mode_0*. The *ground* receives the current configuration of the platform and can now ask for data measured by *I1* or *I2*.

The *scenario 2* consists in moving from *mode_0* to *mode_1*. Thus, first the instruments *I1* and *I2* are turned off. Instrument *I3* is activated and the *ground* makes some measurements.

The *scenario 3* considers the request of switching from *mode_1* to *mode_0* but ends in *Downgraded*. Indeed, *I2* never starts and a timer in *waiti* detects the failure.

B. Implementation in XTRATUM

Again, we implemented the system with a base clock at 1ms and included the environment behaviour, that is the instruments *I1*, *I2* and *I3*, as well as the *ground*. We have implemented all components of the architecture in XTRATUM and compared several periods / MAF choices. Such a choice has an impact on the response time of the system.

Since no temporal constraints were specified, we made some arbitrary choices on the schedule, periods and MAF. The communication between partitions is ensured via sampling ports. Contrary to ROSACE, a message must not be consumed twice. We compared several single core and dual-core scheduling, such as *v1* and *v2* of figure 10.

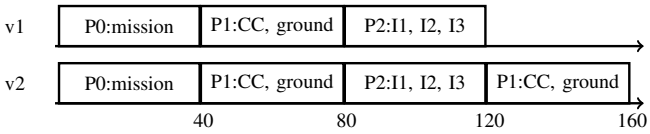

Figure 10: GCU partition schedule

These two scheduling illustrate clearly the impact of the schedule order on the response time. If we compute the response time of scenario 1 to reach *mode_0* from *standby*, it will be 400ms for *v1* and 280ms for *v2* as illustrated in figure 11.

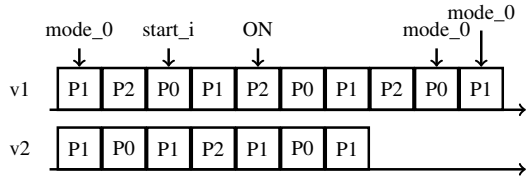

Figure 11: Response time of scenario 1 in v1 and v2

If *v2* seems better, it only applies for this response time. If we compute the response time of scenario 1 to measure a data, that is from a request by the *ground* to the reception of a data, it will be 200ms for *v1* and 240ms for *v2*.

C. Implementation in TASTE/ RTEMS

We followed the same approach as for ROSACE, and built an equivalent model for TASTE/ RTEMS that lead to correct code generation using the TASTE tool-chain.

We have defined each component as a POSIX thread and assessed several scheduling. Since RTEMS does not have time-partitioning, we could implement the system as one monolithic application, using the same philosophy as for ROSACE: defining both a mono-core and a multi-core schedule, and leveraging event port communications to run on top of RTEMS. This approach makes it possible to have a complete event-driven implementation. This reduces the global end-to-end latency, at the expense of the loss of TSP safety/security features supported by XTRATUM.

D. Experiments

We run the three scenarios defined in section V-A, with various scheduling and on the two targets. The results were compliant with the expected behaviour with variation in the end-to-end delays as explained before.

Since this case study is event-driven, we note that extra-care should be done to reduce latency in the system. This would call for extra-optimizations steps that were outside the scope of the study.

VI. RELATED WORK

In this paper, we explored the extensions of a domain-specific critical modeling notation to support multi-core applications. Our approach tries to minimize the impact of these extensions on the overall engineering process. Similar options have been taken in other domains.

In [UO15], the authors introduce the notion of "virtual ECU" as a way to modularize access to multi-core CPUs for the automotive domain. The proposed approach relies on an AMP-like paradigm: each virtual ECU embeds a lightweight instance of the AutoSAR OS. Our approach relies on a different paradigm, driven by the technological capabilities of the target RTOS instead of the hardware capabilities.

In [LSSH15], we have proposed patterns to describe multi-core systems in SysML, with the objective to perform system optimization. The patterns cover various kinds of multi-core hardware platforms, but did not address the configuration of the target OS.

In [NPPV14], the authors propose to describe multi-core systems using MARTE, with the objective to perform code generation, taking into account the MCAPI programming interface for multi-core. As opposed to our approach, the patch taken here follows a top-down approach, starting from MARTE concepts (such as schedulable resources) and later performing model transformation to match MCAPI concepts. In our approach, we followed a bottom-up approach, eliciting updates to be performed on TASTE concepts to support both TSP and SMP RTOS.

During the DREAMS project [DRE13] (2013-2017), a tool-chain [BDM+17] has been developed to generate configuration files. The methodology starts from the modelling in AUTOFOCUS 3 that has been extended to take into account the DREAMS platform specifics. The configuration files are dedicated to KVM and XTRATUM hypervisors that are the basics of the DREAMS middleware.

In our approach, featuring SMP and TSP extends the definition of the hardware platform or OS, and adjusts the binding (or allocation mechanism) to these new elements. This approach makes it possible to reconfigure a mono-core system as a multi-core one as shown in section IV-B; or as a TSP one as demonstrated as part of the ROSACE case study.

VII. CONCLUSION

Multi-core CPUs for embedded systems, in particular space critical systems, are now in most roadmaps for deployment. Starting from hardware elements, and preliminary support in RTOS, the key issues became the correct engineering of systems leveraging these new capabilities.

In this paper, we have presented our approach to bring to designers the configuration space of both SMP and TSP RTOS, and how to combine them through patterns. We then illustrated how to refine these patterns for the RTEMS and XTRATUM OS, and the ESA TASTE modeling tool-chain. The driving objective is to ensure that moving from a regular mono-core to a multi-core and/or SMP deployment can be done with minimal model adaptations.

We illustrate how we could capture the configuration space of two representative software for avionics and space domains, and generate back the software architecture, demonstrating no loss of power of expression. This works illustrated requirements to automate or at least support the optimization process of the deployment to reduce end-to-end latency.

Future work will consolidate the editors necessary to capture the new configuration space at higher-level of abstraction. This will be tested as part of the H2020 ESROCOS project, along with other TRP initiative driven by ESA.

REFERENCES

[ABB+10] Paul Arberet, Jean-Paul Bodeveix, Frédéric Boniol, Jérémy Buisson, Gilles Cannenterre, David Chemouil, Alexandre Cortier, Fabien Dagnat, François Dupont, Mamoun Filali, Emmanuel Fleury, Gérald Garcia, Frédéric Herbreteau, Eric Morand, Julien Ouy, Grégoire Sutre, Ana-Elena Rugina, Martin Streker, and Jean-Pierre Talpin. SPaCIFY: a Formal Model-Driven Engineering for Spacecraft On-Board Software. In *Proceedings of the 5th Conference on Embedded Real Time Software and Systems (ERTS'10)*, 2010.

[AC15] SAE AS2-C. SAE Architecture Analysis and Design Language (AADL) Annex Volume 1, Revision A. Standard AS5506/1A, SAE International, 2015.

[Aer97] Aeronautical Radio Inc. *Avionics Application Software Standard Interface*, 1997.

[AMW+17] M. Muñoz Arancón, G. Montano, M. Wirkus, K. Hoeflinger, D. Silveira, N. Tsiogkas, J. Hugues, H. Bruyninckx, I. Dragomir, and A. Muhammad. Esrocos: A robotic operating system for space and terrestrial applications. In *ASTRA 2017*, June 2017.

[asn02] REC. X680-X.683, ISO/IEC: Abstract Syntax Notation (ASN.1). Technical report, ITU-T, 2002.

[BDM+17] Simon Barner, Alexander Diewald, Jörn Migge, Ali Syed, Gerhard Fohler, Madeleine Faugére, and Daniel Gracia Pérez. Dreams toolchain: Model-driven engineering of mixed-criticality systems. In *Proceedings of 20th International Conference on Model Driven Engineering Languages and Systems (Models'17)*, 2017.

[BDV04] Alan Burns, Brian Dobbing, and Tullio Vardanega. Guide for the use of the ada ravenscar profile in high integrity systems. *Ada Lett.*, XXIV(2):1–74, June 2004.

[BMR+10] Vicent Brocal, Miguel Masmano, Ismael Ripoll, Alfons Crespo, Patricia Balbastre, and Jean-Jacques Metge. Xoncrete. In *Proceedings of the 5th Conference on Embedded Real Time Software and Systems (ERTS'10)*, 2010. http://www.fentiss.com/documents/xoncrete_overview.pdf.

[CHS+14] Daniel Cederman, Daniel Hellström, Joel Sherrill, Gedare Bloom, Mathieu Patte, and Marco Zulianello. RTEMS SMP for LEON3/LEON4 Multi-Processor Devices. In *Data Systems In Aerospace*, Warsaw, Poland, June 2014.

[COB16] COBHAM. GR712RC Dual-Core LEON3-FT SPARC V8 Processor, 2016. http://www.gaisler.com/doc/gr712rc-datasheet.pdf.

[COB17] COBHAM. GR740 Quad Core LEON4 SPARC V8 Processor, 2017. http://www.gaisler.com/doc/gr740/GR740-UM-DS.pdf.

[DRE13] DREAMS consortium. DREAMS: Distributed REal-time Architecture for Mixed Criticality Systems. http://dreams-project.eu, 2013.

[FCOB14] J. Fenn, T. Cornilleau, Y. Oakshott, and A. Britto. A pragmatic approach to capturing safety and security relevant information for reusable european component oriented architecture software components. In *9th IET International Conference on System Safety and Cyber Security (2014)*, pages 1–6, Oct 2014.

[GBC+14] Andrew Gacek, John Backes, Darren Cofer, Konrad Slind, and Mike Whalen. Resolute: an assurance case language for architecture models. In *Proceedings of the 2014 ACM SIGAda annual conference on High integrity language technology*, pages 19–28. ACM, 2014.

[HD17] Jérôme Hugues and Julien Delange. Model-based design and automated validation of arinc653 architectures using the aadl. In Shin Nakajima, Jean-Pierre Talpin, Masumi Toyoshima, and Huafeng Yu, editors, *Cyber-Physical System Design from an Architecture Analysis Viewpoint : Communications of NII Shonan Meetings*, pages pp. 33–52. Springer, 2017.

[HP90] John L. Hennessy and David A. Patterson. *Computer Architecture, Fifth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., 1990.

[JDW12] Christophe Honvault Julien Delange and James Windsor. Model-based engineering approach for system architecture exploration. In *Proceedings of the 6th Conference on Embedded Real Time Software and Systems (ERTS'12)*, 2012.

[LSSH15] Patrick Leserf, Pierre de Saqui-Sannes, and Jérôme Hugues. Multi Domain optimization with SysML modeling. In *20th IEEE International Conference on Emerging Technologies and Factory Automation*, Luxembourg, September 2015.

[MRC+09] Miguel Masmano, Ismael Ripoll, Alfons Crespo, Jean-Jacques. Metge, and Paul Arberet. Xtratum: An open source hypervisor for TSP embedded systems in aerospace. In *DASIA 2009. DAta Systems In Aerospace.*, May. Istanbul 2009.

[NPPV14] A. Nicolas, H. Posadas, P. Peñil, and E. Villar. Automatic deployment of component-based embedded systems from uml/-marte models using mcapi. In *Design of Circuits and Integrated Systems*, pages 1–6, Nov 2014.

[OAR17] OAR Corp. RTEMS – Real-Time Executive for Multiprocessor Systems, 2017. https://www.rtems.org.

[PCD+10] Maxime Perrotin, Eric Conquet, Pierre Dissaux, Thanassis Tsiodras, and Jérôme Hugues. The TASTE toolset: turning human designed heterogeneous systems into computer built homogeneous software. In *Proceedings of the 5th Conference on Embedded Real Time Software and Systems (ERTS'10)*, 2010.

[PSG+14] Claire Pagetti, David Saussié, Romain Gratia, Eric Noulard, and Pierre Siron. The ROSACE Case Study: From Simulink Specification to Multi/Many-Core Execution. In *20th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'14)*, pages 309–318, 2014.

[PV14] Marco Panunzio and Tullio Vardanega. A component-based process with separation of concerns for the development of embedded real-time software systems. *Journal of Systems and Software*, 96:105 – 121, 2014.

[Rad] Radio Technical Commission for Aeronautics (RTCA) and EURopean Organisation for Civil Aviation Equipment (EUROCAE). DO-297: Software, electronic, integrated modular avionics (ima) development guidance and certification considerations.

[SAE12] SAE. *Architecture Analysis & Design Language v2.1 (AS5506B)*. SAE, sep 2012.

[Spa10] Spacify consortium. Etude de cas CNES : Modélisation Synoptic de la partie Commande / Contrôle du GCU (Gestionnaire de Charge Utile). Technical report, ANR, 2010.

[SYS17] SYSGO. PikeOS 4.2 – RTOS with Hypervisor-Functionality, 2017. https://www.sysgo.com/products/pikeos-hypervisor/.

[UO15] M. Urbina and R. Obermaisser. Multi-core architecture for autosar based on virtual electronic control units. In *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–5, Sept 2015.

[Xil01] Xilinx. Zynq-7000 All Programmable SoC ZC702 Evaluation Kit, 201. https://www.xilinx.com/support/documentation/boards_and_kits/zc702_zvik/xtp310-zc702-quickstart.pdf.

[ZdlP13] Juan Zamorano and Juan A. de la Puente. On real-time partitioned multicore systems. *Ada Lett.*, 33(2):33–39, November 2013.