TASTE: a toolchain for multicore TSP applications

Laura Gouveia[1], Maxime Perrotin[2], Thanassis Tsiodras[3], Jérôme Hugues[4], Daniel Silveira[5]

[1] Laura Gouveia, GMV, lasequeiragouveia@gmv.com
[2] Maxime Perrotin, ESA, maxime.perrotin@esa.int
[3] Thanassis Tsiodras, ESA, thanassis.tsiodras@esa.int
[4] Jérôme Hugues, CMU/SEI, jhugues@andrew.cmu.edu
[5] Daniel Silveira, GMV, daniel.silveira@gmv.com

Keywords: Model-Based, Model Transformation, TASTE, AIR, AADL.

1 Introduction

TASTE, "The ASSERT Set of Tools for Engineering" [1], is a development environment dedicated to embedded, real-time systems. It can be used to design small to medium-size systems, relying on formal languages and based on the concept of building "correct by construction" software. It has been recently improved to include support for Time and Space Partitioning (TSP) architectures, specifically GMV's AIR hypervisor [2], and improve code generation performance and tool expandability. In this, we addressed the challenge of generating an Execution Platform with support for multiple partitions on a multicore CPU. TASTE's new TSP functionalities are being implemented in a complex use case, the EagleEye OBSW, deployed on a LEON4-N2X board [4] using AIR with TSP and RTEMS RTOS in multicore [5].

2 Deploying a multicore TSP application using TASTE

Deploying a TSP application using TASTE is not different that a regular TASTE application. In the following, we detail the extensions we performed on the various steps of the TASTE process. We recall the main steps:

- Interface View: The Interface View (IV) defines the logical functions and their interactions within the system. On the Interface View, functions are defined and their interfaces are specified. TASTE is then capable of generating the application code skeletons, clearly identifying where user defines the behaviour of the function. The user can specify the function behaviour either in a programming language (Ada, C, C++ and Micropython are supported), or using a graphical modelling language (SDL, Simulink, etc.), for which code cam be generated and integrated automatically. The Interface View remains unchanged for both TSP and non-TSP applications.

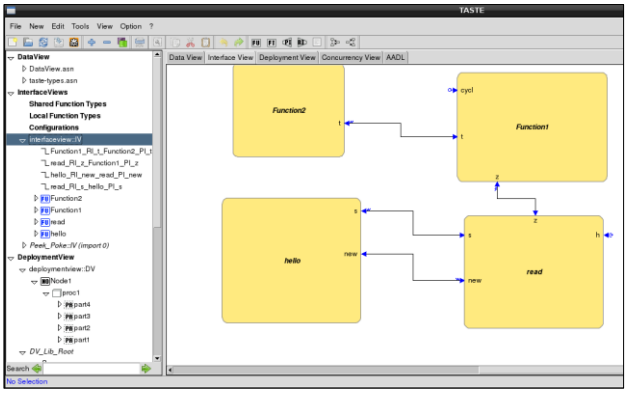

Figure 1 - TASTE Interface View

- Deployment View: The Deployment View (DV) shows how the logical functions of the system are deployed on the target hardware. The Deployment View reuses predefined hardware component descriptors that are available within an AADL library (HW Library). This library contains configuration parameters for the operating system (processor) or the communication libraries (endpoints). These elements are used by the Ocarina code generator and PolyORB-HI middleware to configure the system on the target platform.

A set of additional entities and attributes has been added to the Deployment View to support TSP architectures. Time partitioning is defined by additional scheduling attributes within the Processor, whereas space partitioning requires the definition of memory segments associated with each Partition. Additional information such as the criticality level of each Partition can be also specified.

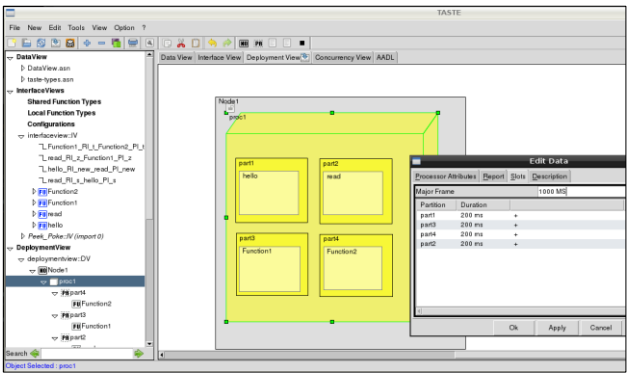

Figure 2 - TASTE Deployment View with partition timing slots definition

- Concurrency View: The Concurrency View is the result of an automatic model transformation whose inputs are the Interface and Deployment Views and the output is a new AADL model including a multi-threading architecture complying with the Ravenscar Computation Model (RCM). The concurrency view is used to perform code generation, but also used scheduling analysis providing two scheduling analysis functions by using Cheddar [6] and MAST [7]. In the Concurrency View, properties can be adjusted to finely tune SMP usage, such as task allocation to core and priority.

3 Code generation and build system

3.1 PolyORB-HI

PolyORB-HI is the main execution platform used in TASTE. It provides the code that interacts with the underlying operating system: RTEMS, GNAT, Linux, FreeRTOS, etc. PolyORB-HI was upgraded to support the latest version of RTEMS that is compatible with multicore platforms and the AIR hypervisor. In particular, PolyORB-HI can now interface its own communication mechanisms (queues, semaphores) with the inter-partition ports provided by AIR.

3.2 Kazoo

Kazoo is the build system of TASTE. It is in charge of computing the set of runtime resources that are needed to deploy the system on target according to the requirements from the Interface and Deployment Views. Kazoo generates the Concurrency View, together with code that ensures the system orchestration together with PolyORB-HI. In the scope of this work, Kazoo was extended to enable the deployment of threads on TSP partitions. This was made possible by the flexible design of Kazoo, which allows creating new code generators via a powerful templating engine.

4 Results and way forward

The main result of the study is an augmented MBSE toolchain that allows to specify and design multi-partition communicating systems. It benefits from a mature MBSE process that abstracts away a lot of complexity and facilitates the prototyping and deployment of TSP systems. The work is not over yet: support of I/O partitions will shortly allow to have isolated hardware-software interactions ; scheduling analysis of TSP systems based on the models ; finer-grain specification of the processor core usage in combination with multi-partitions ; integration with system-level models (via OSRA), etc. MBSE allows for moving step by step from a manual, error-prone development lifecycle to a much more solid and consistent process supported by tools.

References

1. TASTE (The ASSERT Set of Tools for Engineering) Website: http://taste.tools
2. AIR Website: http://www.gmv.com/en/Products/air/
3. ESA Contract No. 4000121551/17/NL/FE for ITT AO/1-8834/17/NL/FE – Multicore implementation of the On-Board Software Reference Architecture with IMA capability:
4. Andersson, J., Hjorth, M., Habinc, S., Gaisler J.: Development of a functional prototype of the quad core NGMP space processor. In Proceedings of Aerospace Conference DASIA (2019).
5. RTEMS real time operating system (RTOS), 2020. https://www.rtems.org/
6. F. Singhoff, J. Legrand, L. Nana, L. Marcé. “Cheddar: a Flexible Real-Time Scheduling Framework”, ACM SIGAda Ada Letters, 24(4):1-8, ACM Press. 2004
7. M. Gonzalez Harbour; J.J. Gutierrez Garcia; J.C. Palencia Gutierrez; J.M. Drake Moyano. MAST: Modeling and analysis suite for real time applications, Proceedings 13th Euromicro Conference on Real-Time Systems, IEEE, 13-15 June 2001.