Distributed Real-time Architecture for Mixed Criticality Systems

Integration and Support Report D4.4.2

Project Acronym	DREAMS	**Grant Agreement Number**		FP7-ICT-2013.3.4-610640	
Document Version	1.0	**Date**	28.7.2017	**Deliverable No.**	4.4.2
Contact Person	Jörn Migge	**Organisation**		RTAW	
Phone	+33 354958661	**E-Mail**		Jorn.migge@realtimeatwork.com	

Contributors

Name	Partner
Jörn Migge	RTAW
Tiziana Mastroti	RTAW
Alexander Diewald	FORTISS
Ali Syed	TUKL
Claire Pagetti	ONERA

Table of Contents

Table of Figures

Executive Summary

The DREAMS tool chain consists of a modelling tool, variability, design space exploration and scheduling design tools, verification tools for timing and safety and last but not least, configuration file generation tools (see D4.4.1[1]). The role of this deliverable is to show how the tool chain has been applied to the demonstrators of the project or publically available use case.

1 Introduction

The goal of T4.4 is not only the creation of a tool chain based on the tools that implement algorithms defined in T4.1, T4.2 and T4.3, but also to make it concretely applicable. For this reason T4.4 has first described the functionalities offered by each tool with their means for interconnections and has identified tool chain use cases. To furthermore ease the adoption by the demonstrators, support has been provided in applying the tool chain, which is documented by this deliverable.

1.1 Relationship to other DREAMS Deliverables

Besides D4.4.1[1], which is the main input regarding the tool chain, all other WP4 deliverables that describe tools and the implemented algorithms are also relevant. For the descriptions of the demonstrators and the perimeter of the application of the tool chain the inputs are D7.1.1 and D8.3.1.

1.2 Positioning of the Deliverable in the Project

This deliverable is the second and last deliverable of the working task T4.4 “Tool integration and Demonstrator Support”.

1.3 Structure of the Deliverable

Three tool chain use cases have been defined in D4.4.1[1]. Their main step are reminded in Section 2. In Section 3 we describe how the tool chain has been applied to the wind power demonstrator (WP7) according to the Use Case 3 “Variability and Design-Space Exploration”. Use Case 2 “Scheduling Configuration with Resource Management” has been applied to a public avionics use case, see Section 4. In Section 5 we describe how the tool chain has been used in the healthcare demonstrator to configure the on-chip and off-chip.

2 Tool Chain Use Cases

In this section we briefly remind the three tool chain use cases defined in D4.4.1[1]. The tool chain application sections are structured according to the (groups of) steps listed in the provided synthetic tables.

The first use case covers the basic configuration of Schedules. Table 1 shows all steps from the definition of the Logical Architecture until the generation of the platform configuration files.

N°	**Description of Step**	**Tool(s)**
	Applications and Resources	
1	Modelling of the Logical Architecture	AF3
2	Modelling the Timing Requirements	AF3
3	Modelling the Platform Architecture	AF3
	Deployment	
4	Modelling the System Software model	AF3
5.A	Defining the Deployment (Manual)	AF3
5.B	Defining the Deployment (DSE)	AF3
	Configuration of Schedulers	
6	Timing Decomposition (optional)	RTaW-Timing
7	Creation of an empty System Schedule	AF3
8	Adding partition/task schedules	Xoncrete
9	Adding on-chip transmission phases for time triggered virtual links	RTaW-Timing
10	Adding off-chip communication schedules	TTE-Plan
11	Timing Analysis	RTaW-Timing
	Platform building block configuration file generation	
12	Generation of XtratuM configuration files	AF3-plug-in
13	Generation of on-chip network communication configuration files	AF3-plug-in
14	Generation of TTEthernet configuration files	TTE-Plan

Table 1 - Use Case 1: Basic Scheduling Configuration

The second use case is an extension of the first one, which includes the configuration of resource managers, in particular for core failures. Table 2 shows all steps.

N°	Description of Step	Tool(s)
Applications and Resources		
1	Modelling of the Logical Architecture	AF3
2	Modelling of Timing Requirements	AF3
3	Modelling of the Platform Architecture	AF3
Deployment		
4	Modelling of the System Software	AF3
5	Defining the Deployment **for the nominal mode**	AF3
Configuration of Schedulers		
7A	Creation of a System Schedule with partition scheduling slots, with Xoncrete.	Xoncrete
7B	Manual creation of a System Schedule with partition scheduling slots.	AF3
8	Adding Scheduling reconfigurations for **failure modes**	GRec
9	Adding transition modes	MCOSF
10	Adding off-chip communication schedules	TTE-Plan
11	Adding on-chip transmission phases for time triggered virtual links	RTaW-Timing
12	Timing Analysis	RTaW-Timing
Platform Building Block Configuration File Generation		
13	Generation of the Resource Management configuration files	AF3-plug-in
14	Generation of on-chip network communication configuration files	AF3-plug-in
15	Generation of TTEthernet configuration files	TTE-Plan

Table 2 - Use Case 2: Scheduling Configuration with Resource Management

The third use case covers the modeling and exploitation of product lines, see Table 3.

N°	Description of Step	Tool
	Construction of the Product-Line	
1	Collect all existing products, as DREAMS system models	AF3
2	Build the 150 % model	AF3
3	Model variation points	BVR
4	Define of the feature realisations	BVR
	Variability and Design Space Exploration	
5	Product-line sampling	BVR
6	Product Realisation	BVR
7	Goal definition	AF3
8	Exploration	AF3

Table 3 - Use Case 3: Variability and Design-Space Exploration

3 Application to the Windpower Demonstrator (WP7)

The windpower demonstrator provided by work package WP7 serves as an evaluation platform for the methods and technologies developed in the DREAMS project applied to the industrial domain. The demonstration application is the control unit of an offshore wind turbine that consists of safety-critical and non-critical subsystems and is connected to a remote SCADA unit via EtherCAT.

3.1 Introduction

The predecessor deliverable D4.4.1[1] defines three use cases to describe the application of the toolchain: use case one and three (see also Section 2) are applied in this demonstrator since reconfiguration is not used for this fail-safe application. From a workflow perspective, the steps included in use case three are executed before the steps of use case one, thus use case three is described first.

Use case three describes the derivation of concrete product models from a product line model. It consists of models that contain all possible features that may be present in the resulting products and variability models that describe all possible features and their interactions. For the windpower demonstrator, these models include a logical architecture, a library of logical components describing different component designs, a safety model, a feature model, and system software and hardware platform models. The workflow described in [2] serves as a base for this use case. A more detailed description of this workflow can be found in the deliverables D4.3.1[3] summarizing the state-of-the art in MCS product line engineering, and D4.3.2[3] and D4.3.3 [4] that describe the methods for each step of the workflow used to produce the product models. Applied to the windpower demonstrator, the resulting product models consist of a modified logical architecture, a system software and hardware platform architecture, a modified safety model, and a generated deployment of the component architecture to the platform architecture.

These product models are used as an input to the steps described by use case 1. In particular, only the steps 6 to 14 (see Section 2) are exercised since due to the application of the workflow described above, a full system deployment is already available. Section 3.4 covers the configuration of the Schedulers, starting with the decomposition of end-to-end latency constraints into sub-constraints for the task and on-chip communication, followed by the actual generation of time-triggered partition and task schedules, as well as a schedule of the virtual links transferred via the NoC of the DREAMS harmonized platform. Finally, the resulting system schedule is checked for the satisfaction of end-to-end latency. Section 3.5 covers the generation of configuration files for scheduling related building block of the target, which includes the NoC, the hypervisors, the gateways etc.

3.2 Application Architecture and Temporal Specification

In the following, the WP7 application will be briefly described using one concrete set of product models that is derived from the set of product line models using the steps outlined above. The variability resolution process found 8 potential product model sets using the configuration described in [2] out of which 6 were identified to be able to satisfy the safety constraints by the DSE and the safety analysis. We will focus on the product model set number 6 that contains all features from the product model and thus is suitable to explain the application at a concrete instance.

3.2.1 Logical and Platform Architecture Models

Product 6 of the Windpower demonstrator is characterized by the redundancy of the safety protection function. Therefore, the corresponding Components "SafetyProtection", "Diagnostic" and "IOServer" have been automatically duplicated during variability resolution, as can be seen in Figure 1. The physical components of the Platform Architecture used in the Windpower demonstrator, are depicted inFigure 2. It consists of 4 Processor Tiles (red boxes), connected by a NoC (blue box). The right hand side represents the DHP that installed in the GALILEO box (see [5]).

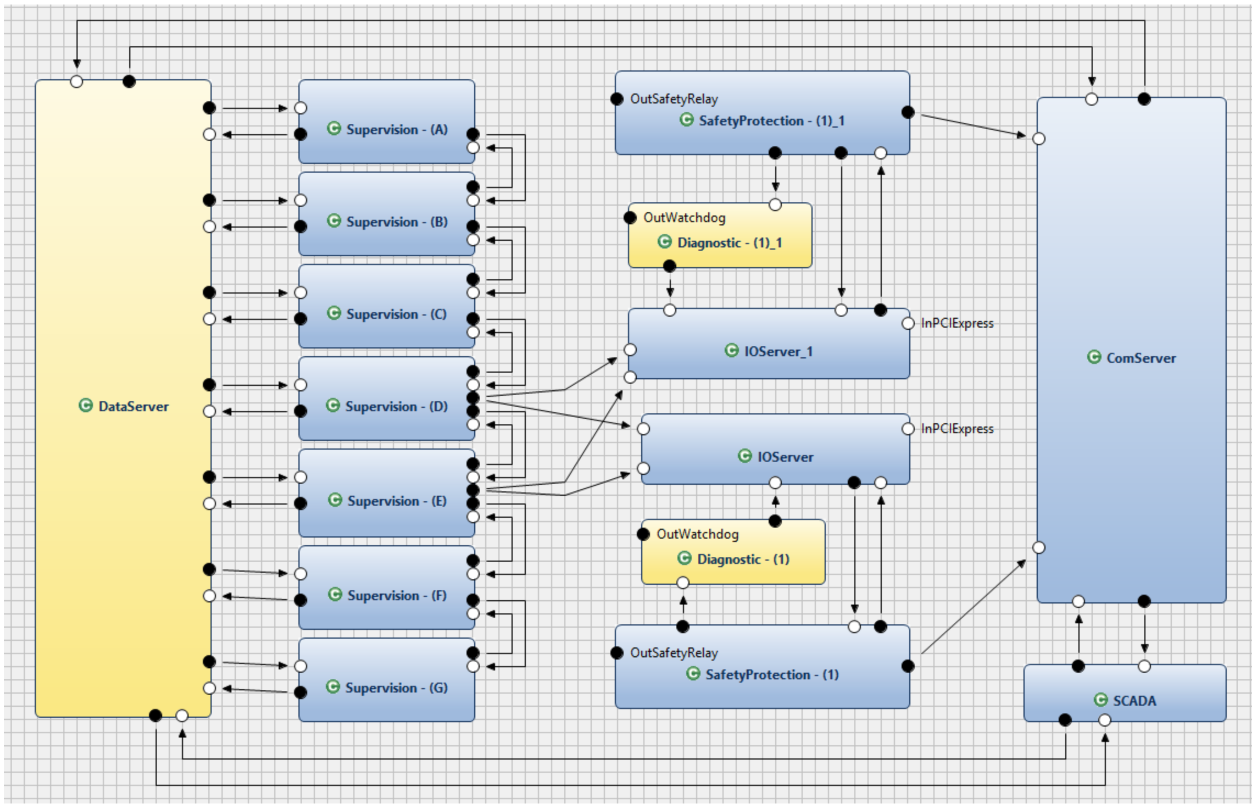


Figure 1 – WP7: Logical Architecture of Product6 of the Windpower Demonstrator (AF3).

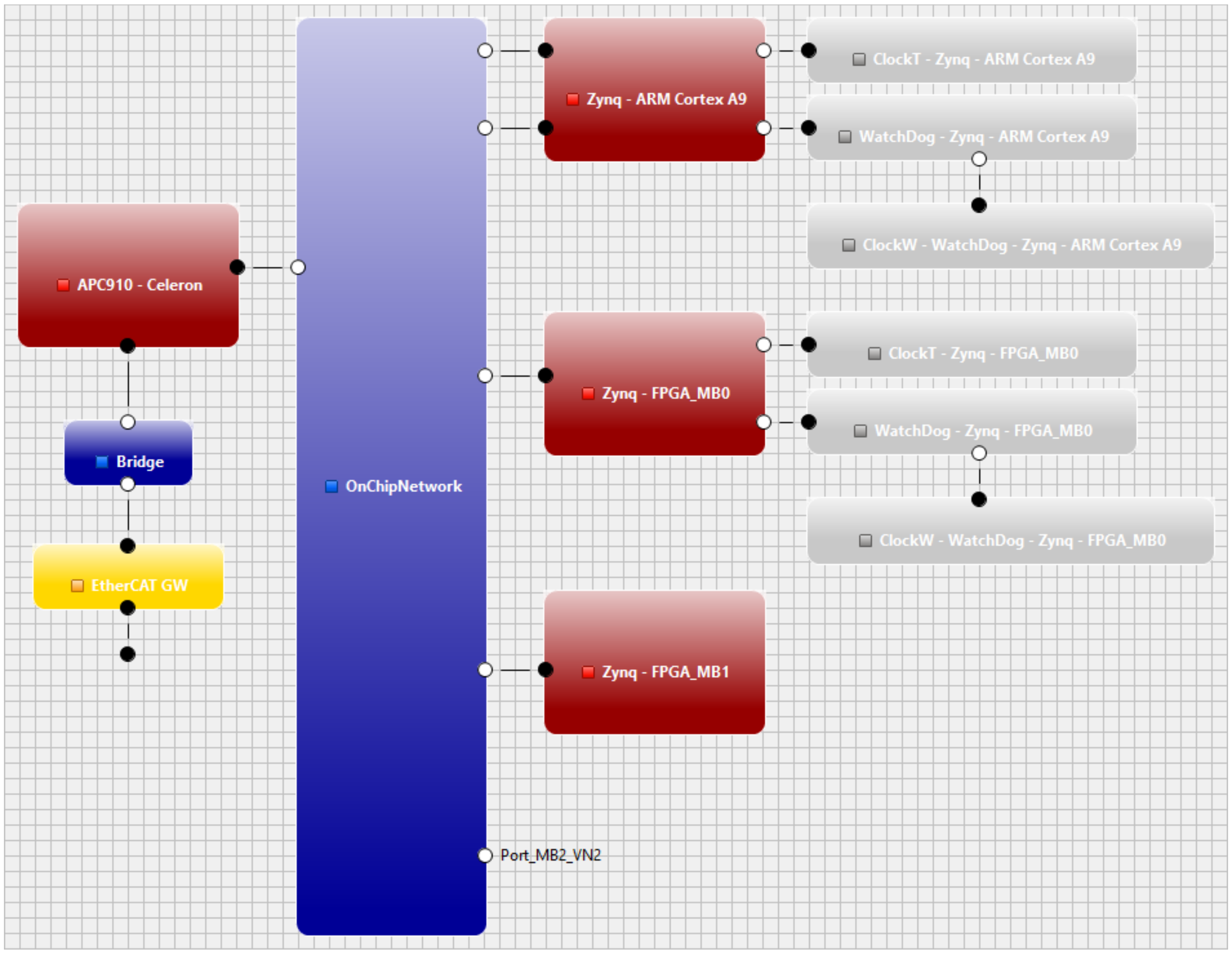
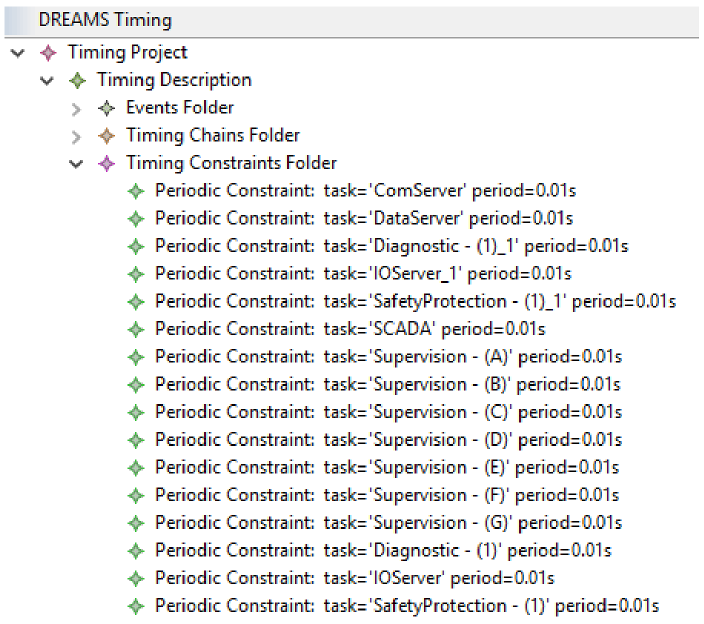

ClockT - Zynq - ARM Cortex A9
Zynq - ARM Cortex A9
WatchDog - Zynq - ARM Cortex A9
ClockW - WatchDog - Zynq - ARM Cortex A9
APC910 - Celeron
Bridge
EtherCAT GW
OnChipNetwork
ClockT - Zynq - FPGA_MB0
Zynq - FPGA_MB0
WatchDog - Zynq - FPGA_MB0
ClockW - WatchDog - Zynq - FPGA_MB0
Zynq - FPGA_MB1
Port_MB2_VN2

Figure 2 - WP7: Platform architecture modelling the hardware platform of the windpower demonstrator (AF3).

3.2.2 Timing Requirements

The definition of timing requirements corresponds to step 2 of the Tool Chain Use Cases 1 and 2, see [1].

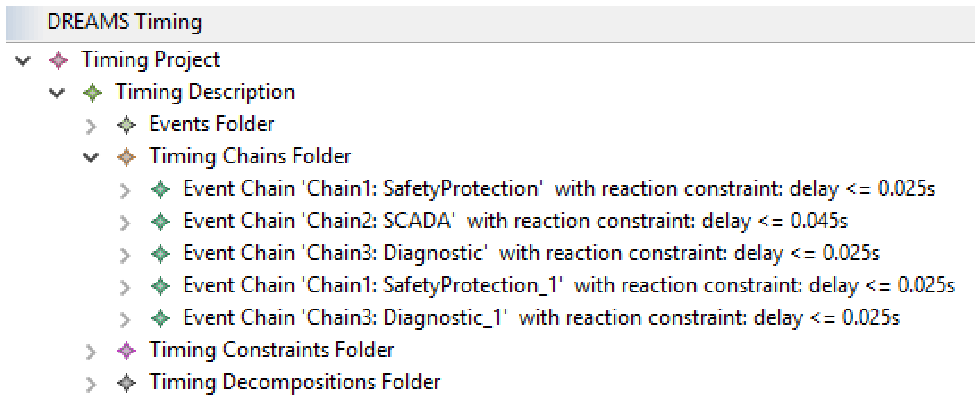
All tasks of the Windpower demonstrator (see Figure 1) must be executed with a period of 10ms. These timing requirements are expressed as Periodic Constraints and specified through the Timing Model editor integrated into AF3, see Figure 3.



DREAMS Timing
- Timing Project
  - Timing Description
    - Events Folder
    - Timing Chains Folder
    - Timing Constraints Folder
      - Periodic Constraint: task='ComServer' period=0.01s
      - Periodic Constraint: task='DataServer' period=0.01s
      - Periodic Constraint: task='Diagnostic - (1)_1' period=0.01s
      - Periodic Constraint: task='IOServer_1' period=0.01s
      - Periodic Constraint: task='SafetyProtection - (1)_1' period=0.01s
      - Periodic Constraint: task='SCADA' period=0.01s
      - Periodic Constraint: task='Supervision - (A)' period=0.01s
      - Periodic Constraint: task='Supervision - (B)' period=0.01s
      - Periodic Constraint: task='Supervision - (C)' period=0.01s
      - Periodic Constraint: task='Supervision - (D)' period=0.01s
      - Periodic Constraint: task='Supervision - (E)' period=0.01s
      - Periodic Constraint: task='Supervision - (F)' period=0.01s
      - Periodic Constraint: task='Supervision - (G)' period=0.01s
      - Periodic Constraint: task='Diagnostic - (1)' period=0.01s
      - Periodic Constraint: task='IOServer' period=0.01s
      - Periodic Constraint: task='SafetyProtection - (1)' period=0.01s

Figure 3 – WP7: Periodicity constraint on task level components.

Three Timing Chains with latency constraints have been identified for the Windpower demonstrator (see Section 4.2.2 on how to declare them). Because of the redundancy of the safety protection in "product 6", some tasks and the corresponding Timing Chains have been automatically duplicated during variability resolution, as can be seen in Figure 4.



DREAMS Timing
- Timing Project
  - Timing Description
    - Events Folder
    - Timing Chains Folder
      - Event Chain 'Chain1: SafetyProtection' with reaction constraint: delay <= 0.025s
      - Event Chain 'Chain2: SCADA' with reaction constraint: delay <= 0.045s
      - Event Chain 'Chain3: Diagnostic' with reaction constraint: delay <= 0.025s
      - Event Chain 'Chain1: SafetyProtection_1' with reaction constraint: delay <= 0.025s
      - Event Chain 'Chain3: Diagnostic_1' with reaction constraint: delay <= 0.025s
    - Timing Constraints Folder
    - Timing Decompositions Folder

Figure 4 - WP7: Timing Chains with Reaction Constraints.

The Timing Chain "Safety Protection" spans two tasks, namely "IOServer" and "SafetyProtection": it starts with the reading of the input at port "InPCIExpress" by the task "IOServer" and ends with the writing to the output "OutSafetyRelay", see in Figure 5.

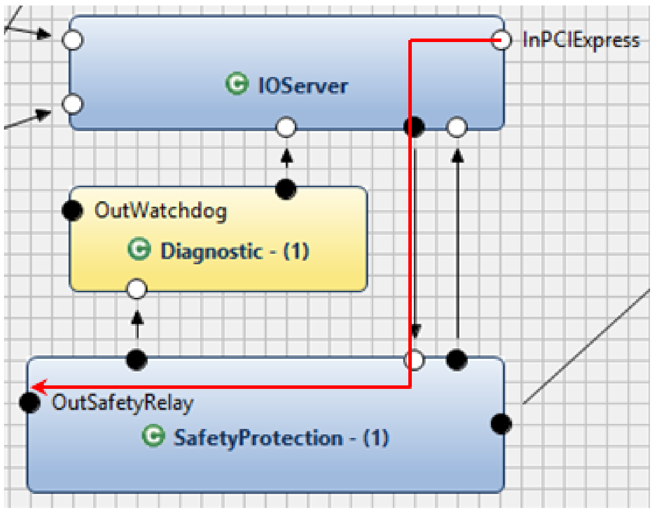
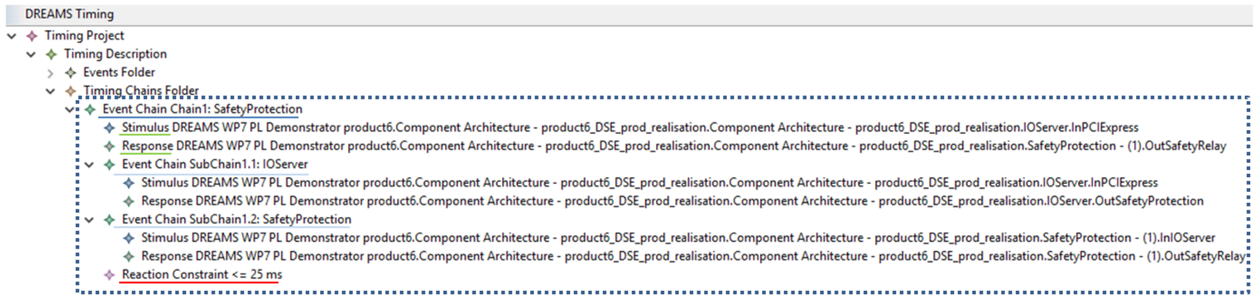


Figure 5 - WP7: Path of Timing Chain "Safety Protection"

Figure 6 shows the two sub-chains corresponding to the tasks “Safety Protection” and “IOServer”, with their corresponding stimulus and response events.



DREAMS Timing
- Timing Project
  - Timing Description
    - Events Folder
    - Timing Chains Folder
      - Event Chain Chain1: SafetyProtection
        - Stimulus DREAMS WP7 PL Demonstrator product6.Component Architecture - product6_DSE_prod_realisation.Component Architecture - product6_DSE_prod_realisation.IOServer.InPCIExpress
        - Response DREAMS WP7 PL Demonstrator product6.Component Architecture - product6_DSE_prod_realisation.Component Architecture - product6_DSE_prod_realisation.SafetyProtection - (1).OutSafetyRelay
        - Event Chain SubChain1.1: IOServer
          - Stimulus DREAMS WP7 PL Demonstrator product6.Component Architecture - product6_DSE_prod_realisation.Component Architecture - product6_DSE_prod_realisation.IOServer.InPCIExpress
          - Response DREAMS WP7 PL Demonstrator product6.Component Architecture - product6_DSE_prod_realisation.Component Architecture - product6_DSE_prod_realisation.IOServer.OutSafetyProtection
        - Event Chain SubChain1.2: SafetyProtection
          - Stimulus DREAMS WP7 PL Demonstrator product6.Component Architecture - product6_DSE_prod_realisation.Component Architecture - product6_DSE_prod_realisation.SafetyProtection - (1).InIOServer
          - Response DREAMS WP7 PL Demonstrator product6.Component Architecture - product6_DSE_prod_realisation.Component Architecture - product6_DSE_prod_realisation.SafetyProtection - (1).OutSafetyRelay
        - Reaction Constraint <= 25 ms

Figure 6 - WP7: Sub-chains of the Timing Chain "Safety Protection"

3.3 Deployment

3.3.1 System Software

As shown in Figure 7, hypervisors are defined for all Tiles, such that none of the tiles is used as a bare-metal processor.

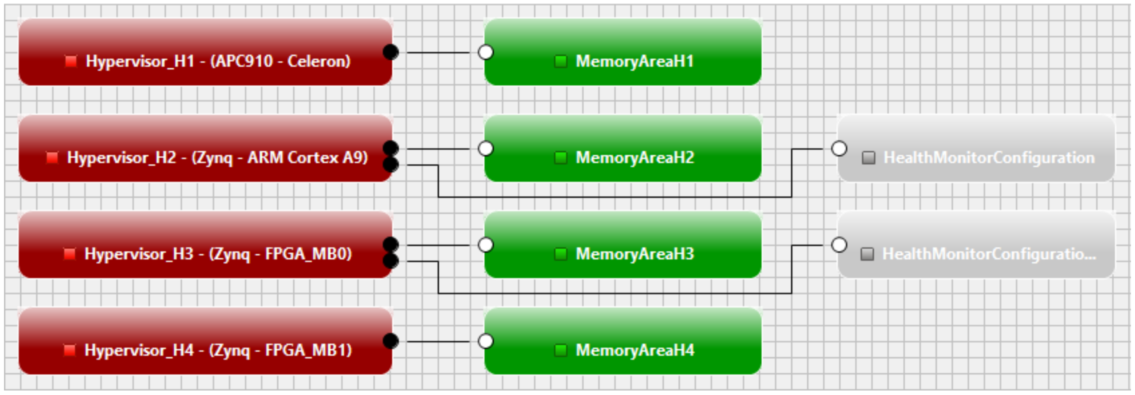
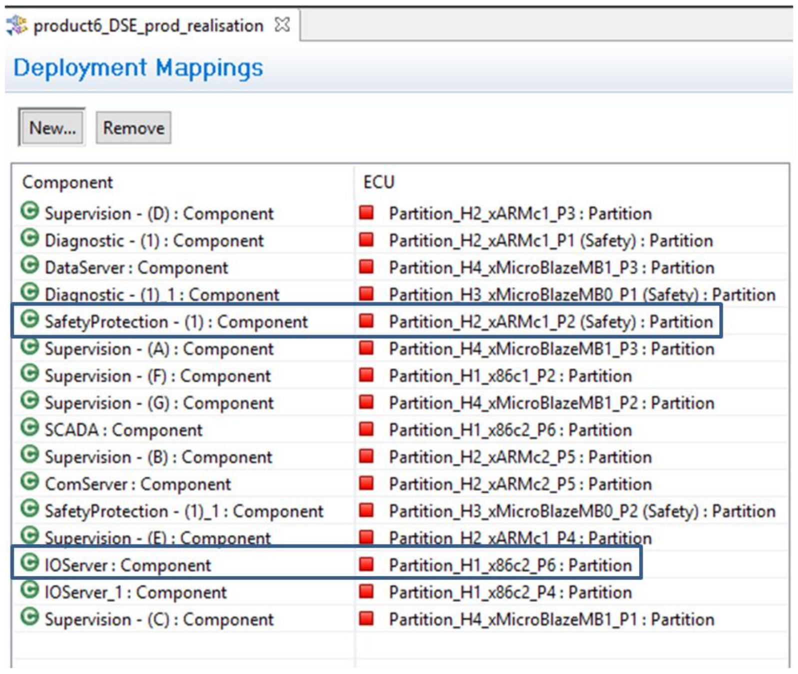

Hypervisor_H1 - (APC910 - Celeron)
MemoryAreaH1
Hypervisor_H2 - (Zynq - ARM Cortex A9)
MemoryAreaH2
HealthMonitorConfiguration
Hypervisor_H3 - (Zynq - FPGA_MB0)
MemoryAreaH3
HealthMonitorConfiguratio...
Hypervisor_H4 - (Zynq - FPGA_MB1)
MemoryAreaH4

Figure 7 - WP7: System software (AF3).

3.3.2 Deployment

Figure 8 shows the Deployment “Product6_DSE_prod_realisation” of the component level tasks to the Partitions of the hypervisors, as defined by the DSE for Product6. As can be seen, the two tasks of the timing chain “SafetyProtection” are allocated to two different Tiles and thus the message sent from “IOServer” to “SafetyProtecton” has to transit over the on-chip network.



Component	ECU
Supervision - (D) : Component	Partition_H2_xARMc1_P3 : Partition
Diagnostic - (1) : Component	Partition_H2_xARMc1_P1 (Safety) : Partition
DataServer : Component	Partition_H4_xMicroBlazeMB1_P3 : Partition
Diagnostic - (1)_1 : Component	Partition_H3_xMicroBlazeMB0_P1 (Safety) : Partition
SafetyProtection - (1) : Component	Partition_H2_xARMc1_P2 (Safety) : Partition
Supervision - (A) : Component	Partition_H4_xMicroBlazeMB1_P3 : Partition
Supervision - (F) : Component	Partition_H1_x86c1_P2 : Partition
Supervision - (G) : Component	Partition_H4_xMicroBlazeMB1_P2 : Partition
SCADA : Component	Partition_H1_x86c2_P6 : Partition
Supervision - (B) : Component	Partition_H2_xARMc2_P5 : Partition
ComServer : Component	Partition_H2_xARMc2_P5 : Partition
SafetyProtection - (1)_1 : Component	Partition_H3_xMicroBlazeMB0_P2 (Safety) : Partition
Supervision - (E) : Component	Partition_H2_xARMc1_P4 : Partition
IOServer : Component	Partition_H1_x86c2_P6 : Partition
IOServer_1 : Component	Partition_H1_x86c2_P4 : Partition
Supervision - (C) : Component	Partition_H4_xMicroBlazeMB1_P1 : Partition

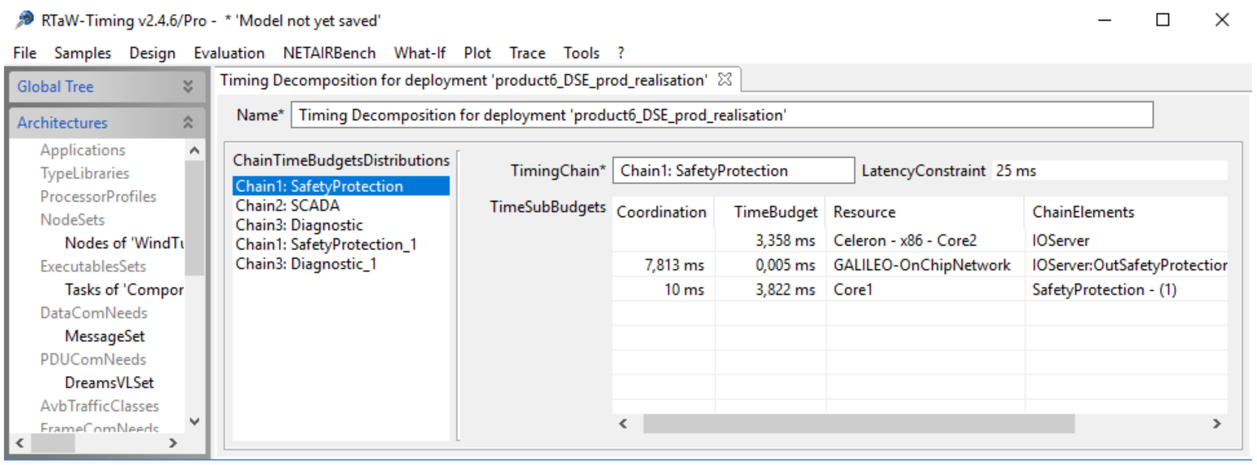
Figure 8 - Wp7: Deployment for Product6 of the Windpower demonstrator.

3.4 Configuration of Schedulers

At this point, the deployment of the applications to Tiles and Partitions is defined and thus end-to-end latency constraints may be decomposed into sub-constraints (Section 3.4.1) which are the inputs for the configurations of the schedules of the task scheduling (Section 3.4.2) and on-chip communication scheduling domains (Section 3.4.3). Before generating the configuration file of the platform building blocks, end-to-end delays resulting from the domain schedules are verified against the latency constraints (Section 3.4.4).

3.4.1 Timing Decomposition

The decomposition of end-to-end latency constraints into sub-constraints for the different scheduling domains (task scheduling an on-chip communication) corresponds to step 6 of the Tool Chain Use Case 1, see [1]. In order to execute the decomposition algorithm implemented in RTaW-Timing, the “Timing Decomposition” entry of the context menu of the Deployment “Product6_DSE_prod_realisation” must be selected (see Section 4.3.1.5 of [1]). As a result, the system description is automatically exported to RTaW-Timing, which computes the decomposition and shows the result in its GUI, see Figure 9. Furthermore, the results are automatically imported back into AF3 and visible in the Timing Model Editor (see Figure 10).



Name* Timing Decomposition for deployment 'product6_DSE_prod_realisation'

ChainTimeBudgetsDistributions
- Chain1: SafetyProtection
- Chain2: SCADA
- Chain3: Diagnostic
- Chain1: SafetyProtection_1
- Chain3: Diagnostic_1

TimingChain* Chain1: SafetyProtection LatencyConstraint 25 ms

TimeSubBudgets

Coordination	TimeBudget	Resource	ChainElements
	3,358 ms	Celeron - x86 - Core2	IOServer
7,813 ms	0,005 ms	GALILEO-OnChipNetwork	IOServer:OutSafetyProtectior
10 ms	3,822 ms	Core1	SafetyProtection - (1)

Figure 9 - WP7: Details of a timing decomposition (RTaW-Timing).

In Section 3.3.2 we have seen that the two tasks of the Timing Chain “SafetyProtection” are deployed to two different Tiles. Therefore, the decomposition of the end-to-end latency constraints consists of 3 sub-constraints or “budgets”, one for the processing on the “Celeron” Tile, a second for the communication of the NoC and a third for the processing on the “ARM” Tile. Figure 9 shows the details of the decomposition, where the so called “coordination delays” are listed separately. A coordination delay is the time it may take, for example, between the latest possible time the output of a task is available for the transmission over the network, until the packet that transports that output is queued for transmission. The amount of this delay depends on how well the execution of the task and the queuing of the packet are coordinated. In the most favourable case, the frame is instantiated “just after” the latest possible execution end of the task and then queued with the fresh outputs. But coordination is only possible if periods are harmonic, i.e. one is a divider of the other or if they are simply equal. But defining harmonic periods might be impossible, because of different granularities of time, even if the clocks are synchronized as in the DREAMS architecture. The granularity of time is 1µs for the XtratuM hypervisor, but 2^{-n} s for the NoC NI. In this use case, tasks have periods of 10 ms, but 10 ms are not a negative power of 2. To allow that every task outputs is sent over the network and never overwritten, a shorter compatible period must be chosen for the packet, which is 2^{-7} s = 7,813 ms. This implies non-harmonic periods and makes coordination impossible. As a result, the coordination delay may reach the period of the frame that transports the output of the task or equal to the period of the task that reads the data transported by a frame. There a corresponding budget for the coordination is foreseen.

In this use case it means for example that a message produced by “IOServer” on the “Celeron” Tile may have to wait up to 7,813 ms before being queued for sending over the NoC and when the message arrives at the “ARM” Tile, it may have to wait up to 10ms before being read by the next execution of “SafetProtection”. These are the values visible in the “Coordination” column in Figure 9.

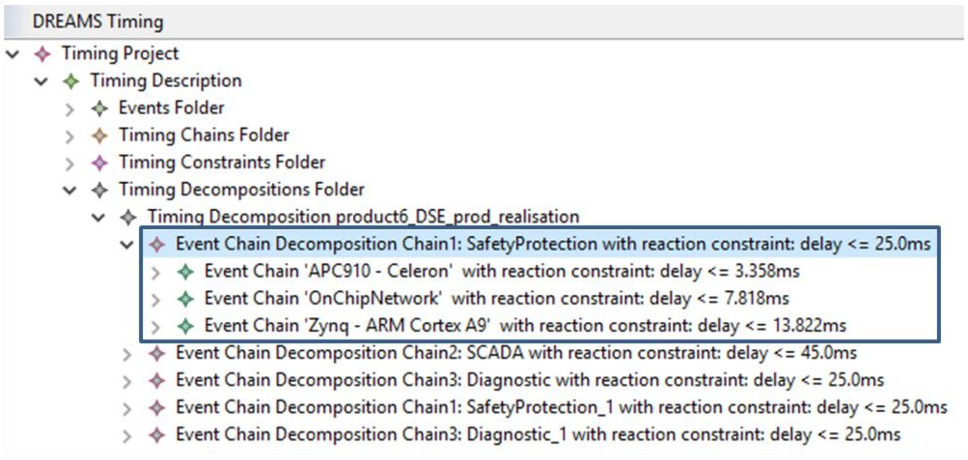

DREAMS Timing
Timing Project
Timing Description
Events Folder
Timing Chains Folder
Timing Constraints Folder
Timing Decompositions Folder
Timing Decomposition product6_DSE_prod_realisation
Event Chain Decomposition Chain1: SafetyProtection with reaction constraint: delay <= 25.0ms
Event Chain 'APC910 - Celeron' with reaction constraint: delay <= 3.358ms
Event Chain 'OnChipNetwork' with reaction constraint: delay <= 7.818ms
Event Chain 'Zynq - ARM Cortex A9' with reaction constraint: delay <= 13.822ms
Event Chain Decomposition Chain2: SCADA with reaction constraint: delay <= 45.0ms
Event Chain Decomposition Chain3: Diagnostic with reaction constraint: delay <= 25.0ms
Event Chain Decomposition Chain1: SafetyProtection_1 with reaction constraint: delay <= 25.0ms
Event Chain Decomposition Chain3: Diagnostic_1 with reaction constraint: delay <= 25.0ms

Figure 10 - WP7: Timing Decomposition of the chain "Safety Protection"

3.4.2 Task and Partition Scheduling

The generation of schedules for partitions and tasks corresponds to step 8 of the Tool Chain Use Case 1, see [1]. In order to perform this step with Xoncrete, the “Xoncrete Eprj file Export” entry of the context menu of the Deployment must be selected, as show in Figure 11 (changed since the explanations given in [1]).

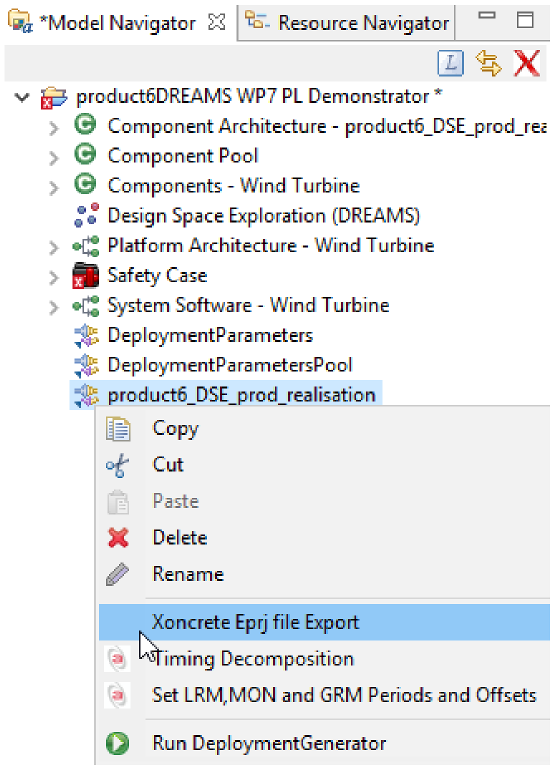
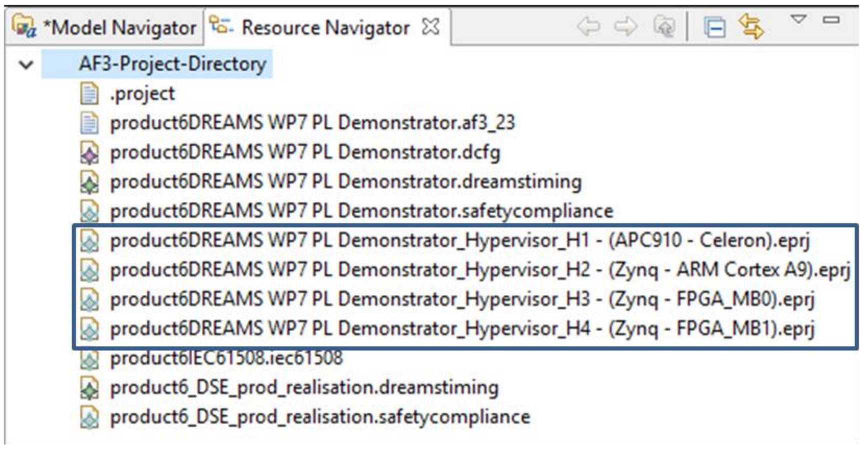

*Model Navigator
Resource Navigator
product6DREAMS WP7 PL Demonstrator *
Component Architecture - product6_DSE_prod_rea
Component Pool
Components - Wind Turbine
Design Space Exploration (DREAMS)
Platform Architecture - Wind Turbine
Safety Case
System Software - Wind Turbine
DeploymentParameters
DeploymentParametersPool
product6_DSE_prod_realisation
Copy
Cut
Paste
Delete
Rename
Xoncrete Eprj file Export
Timing Decomposition
Set LRM,MON and GRM Periods and Offsets
Run DeploymentGenerator

Figure 11- WP7: Invocation of the Xoncrete file exporter.

The result is the creation of 4 Xoncrete project files, one for each Tile/Hypervisor. They are located besides the AF3 model file and can be visualized in the “Resource Navigator”, see Figure 12.



AF3-Project-Directory
.project
product6DREAMS WP7 PL Demonstrator.af3_23
product6DREAMS WP7 PL Demonstrator.dcfg
product6DREAMS WP7 PL Demonstrator.dreamstiming
product6DREAMS WP7 PL Demonstrator.safetycompliance
product6DREAMS WP7 PL Demonstrator_Hypervisor_H1 - (APC910 - Celeron).eprj
product6DREAMS WP7 PL Demonstrator_Hypervisor_H2 - (Zynq - ARM Cortex A9).eprj
product6DREAMS WP7 PL Demonstrator_Hypervisor_H3 - (Zynq - FPGA_MB0).eprj
product6DREAMS WP7 PL Demonstrator_Hypervisor_H4 - (Zynq - FPGA_MB1).eprj
product6IEC61508.iec61508
product6_DSE_prod_realisation.dreamstiming
product6_DSE_prod_realisation.safetycompliance

Figure 12 - WP7: Xoncrete input files (AF3).

Notice that the exporter has calculated the least common multiple (LCM) of all task periods (=10ms) and has exported this LCM as common MAF for all Tile/Hypervisors.

Next, Xoncrete needs to be applied separately to each file, since Xoncrete can generates the scheduling of tasks for only one Tile/Hypervisor. For each file, the following steps need to be performed in Xoncrete:

1. The file must be imported with the help of the “File->Load” menu entry.
2. The “Analysis->Temporal Analysis” menu entry must be selected.
3. The “Adj. Periods to given MAF” button must be clicked.
4. The “2.Schedule Generation” button must be clicked.
5. The “Run Analyzer” button must be clicked.
6. When the schedule has been generated, the “File->Export to XMC file” menu entry must be selected and the result file exported to the folder where the input files are located.

When all schedules are created, a SystemSchedule needs to be created in AF3 and the context menu “XtratuM Xmc file Scheduling Import” must be used to import the Xoncrete result files, as shown in Figure 13 (changed since the explanations given in [1]).

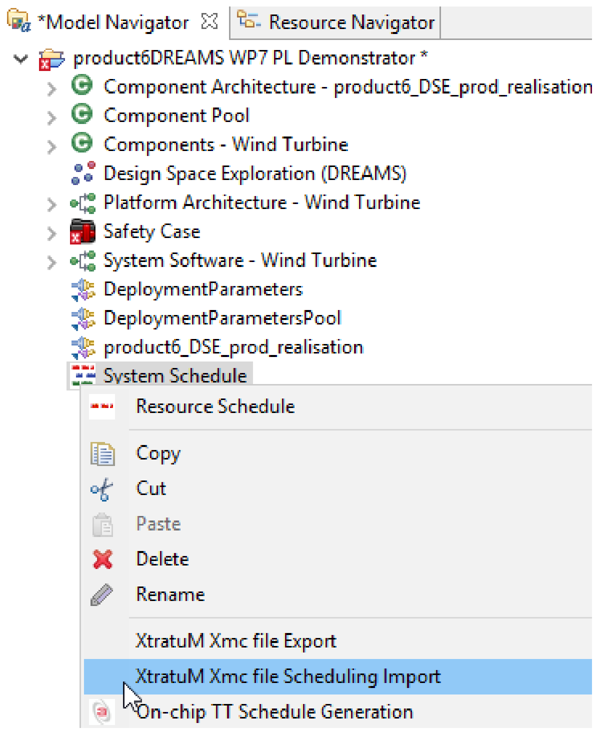
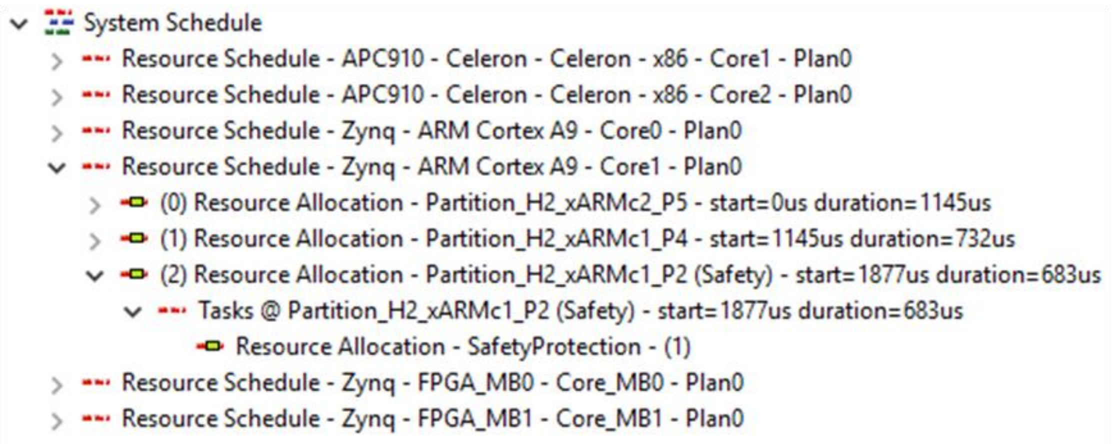

*Model Navigator
Resource Navigator
product6DREAMS WP7 PL Demonstrator *
Component Architecture - product6_DSE_prod_realisation
Component Pool
Components - Wind Turbine
Design Space Exploration (DREAMS)
Platform Architecture - Wind Turbine
Safety Case
System Software - Wind Turbine
DeploymentParameters
DeploymentParametersPool
product6_DSE_prod_realisation
System Schedule
Resource Schedule
Copy
Cut
Paste
Delete
Rename
XtratuM Xmc file Export
XtratuM Xmc file Scheduling Import
On-chip TT Schedule Generation

Figure 13 - WP7: Import of Xoncrete output.

The result of the import is the creation of the corresponding partition and task scheduling model elements in AF3, as shown in Figure 14.



System Schedule
- Resource Schedule - APC910 - Celeron - Celeron - x86 - Core1 - Plan0
- Resource Schedule - APC910 - Celeron - Celeron - x86 - Core2 - Plan0
- Resource Schedule - Zynq - ARM Cortex A9 - Core0 - Plan0
- Resource Schedule - Zynq - ARM Cortex A9 - Core1 - Plan0
  - (0) Resource Allocation - Partition_H2_xARMc2_P5 - start=0us duration=1145us
  - (1) Resource Allocation - Partition_H2_xARMc1_P4 - start=1145us duration=732us
  - (2) Resource Allocation - Partition_H2_xARMc1_P2 (Safety) - start=1877us duration=683us
    - Tasks @ Partition_H2_xARMc1_P2 (Safety) - start=1877us duration=683us
      - Resource Allocation - SafetyProtection - (1)
- Resource Schedule - Zynq - FPGA_MB0 - Core_MB0 - Plan0
- Resource Schedule - Zynq - FPGA_MB1 - Core_MB1 - Plan0

Figure 14 - WP7: imported task and partition schedules.

Notice that the exporter has created ETEFs for the defined Timing Chains, with offset and deadline constraints that correspond to their timing decomposition.

Let us take again the Timing Chain “Safety Protection” considered in Sections 3.2.2 and 3.4.1. The task “SafetyProtection” is the last segment of the chain and executed on the “ARM” Tile. The offset constraint for the corresponding ETEF is computed as follows:

1. Sum of the “budgets” of the preceding segments “processing on the Celeron Tile” and “transmission over the off-chip network”, (see Figure 10): 3,358 ms + 7,818 ms = 11,176 ms.

2. The offset needs to be normalized with respect to the period of the task, i.e. the modulo with the task period needs to be computed: 11,176 ms mod 10 ms = **1,176 ms**.

Since the budget for the processing on the “ARM” Tile is 13,822 ms (see Figure 10), the deadline constraint for the ETEF is 1,176 ms + 13,822 ms = **14,998 ms**.

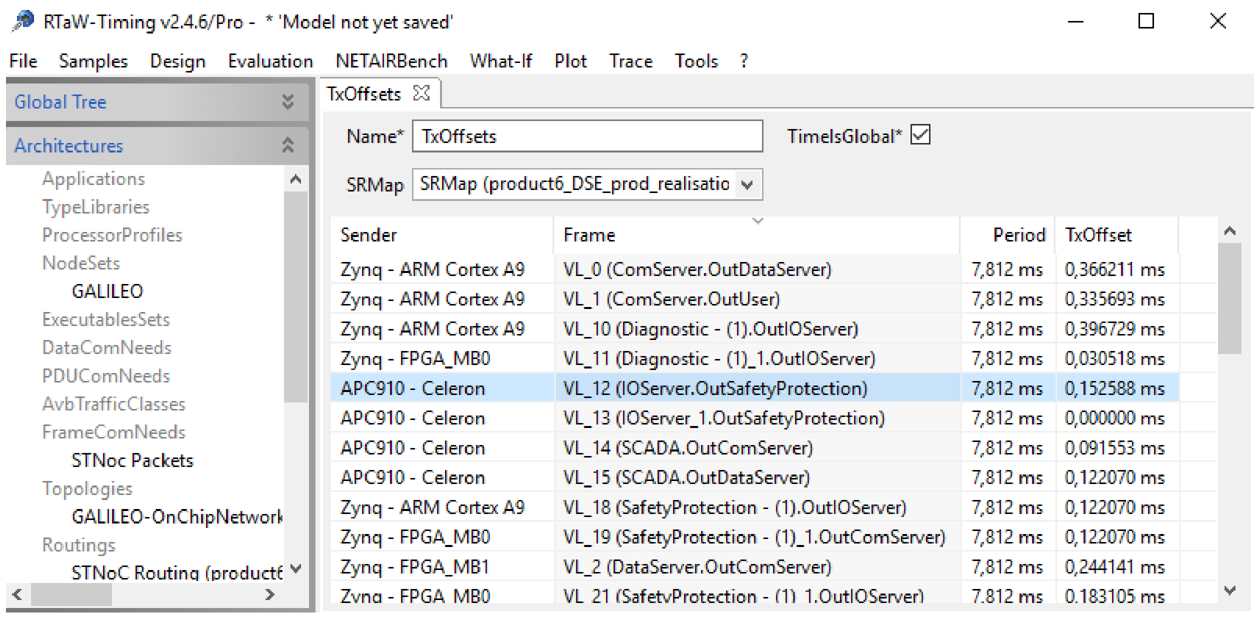
As can be seen in Figure 14, the task “SafetyProtection” is executed in the slot starting at 1,877ms for a duration of 683 µs, which is between 1,176 ms and 14,998 ms requested by the offset and deadline constraint.

3.4.3 On-chip Scheduling of TT Virtual Links

The generation of a time triggered scheduling configuration for TT Virtual Links over the on-chip network corresponds to step 9 of the Tool Chain Use Case 1, see [1]. In order to execute the algorithm implemented in RTaW-Timing, the “On-chip TT Schedule Generation” entry of the context menu of the SystemSchedule must be selected, as explained in Section 4.3.3.5 of [1]. As a result, the description of the on-chip communication requirements is exported to RTaW-Timing, which computes for every TT Virtual Link:

- a period, compatible with the time granularity of the on-chip network (2^{-n} s)
- a transmission phase that avoid any collision with other TT Virtual Links

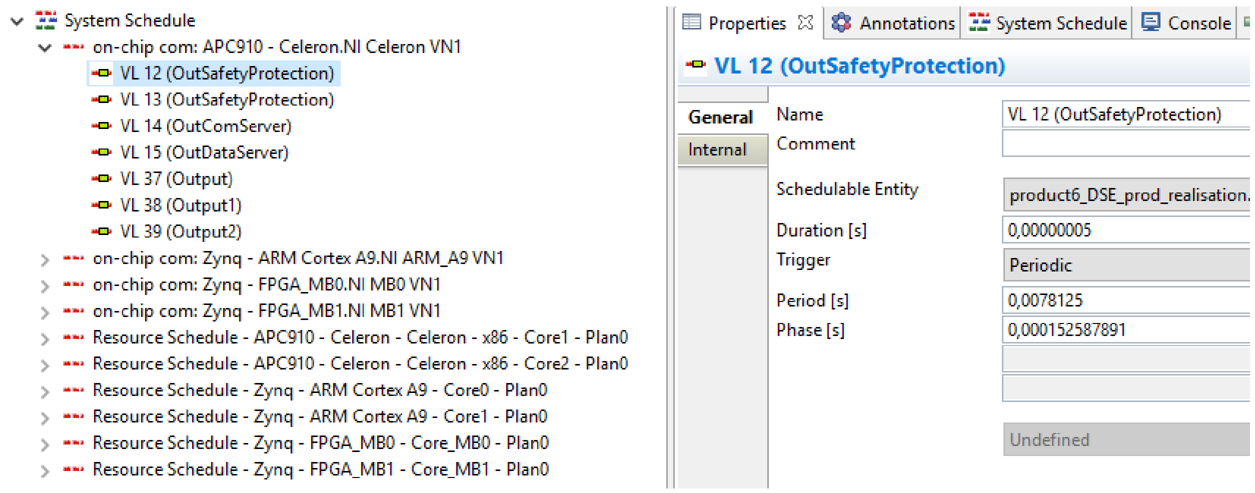
The computed parameters are shown in the GUI of RTaW-Timing (Figure 15) and automatically imported into AF3, see Figure 16.



Name* TxOffsets
TimeIsGlobal* ☑
SRMap SRMap (product6_DSE_prod_realisatio

Sender	Frame	Period	TxOffset
Zynq - ARM Cortex A9	VL_0 (ComServer.OutDataServer)	7,812 ms	0,366211 ms
Zynq - ARM Cortex A9	VL_1 (ComServer.OutUser)	7,812 ms	0,335693 ms
Zynq - ARM Cortex A9	VL_10 (Diagnostic - (1).OutIOServer)	7,812 ms	0,396729 ms
Zynq - FPGA_MB0	VL_11 (Diagnostic - (1)_1.OutIOServer)	7,812 ms	0,030518 ms
APC910 - Celeron	VL_12 (IOServer.OutSafetyProtection)	7,812 ms	0,152588 ms
APC910 - Celeron	VL_13 (IOServer_1.OutSafetyProtection)	7,812 ms	0,000000 ms
APC910 - Celeron	VL_14 (SCADA.OutComServer)	7,812 ms	0,091553 ms
APC910 - Celeron	VL_15 (SCADA.OutDataServer)	7,812 ms	0,122070 ms
Zynq - ARM Cortex A9	VL_18 (SafetyProtection - (1).OutIOServer)	7,812 ms	0,122070 ms
Zynq - FPGA_MB0	VL_19 (SafetyProtection - (1)_1.OutComServer)	7,812 ms	0,122070 ms
Zynq - FPGA_MB1	VL_2 (DataServer.OutComServer)	7,812 ms	0,244141 ms
Zynq - FPGA_MB0	VL_21 (SafetyProtection - (1)_1.OutIOServer)	7,812 ms	0,183105 ms

Figure 15 - WP7: On-chip TT Scheduling computed by RTaW-Timing


System Schedule
- on-chip com: APC910 - Celeron.NI Celeron VN1
  - VL 12 (OutSafetyProtection)
  - VL 13 (OutSafetyProtection)
  - VL 14 (OutComServer)
  - VL 15 (OutDataServer)
  - VL 37 (Output)
  - VL 38 (Output1)
  - VL 39 (Output2)
- on-chip com: Zynq - ARM Cortex A9.NI ARM_A9 VN1
- on-chip com: Zynq - FPGA_MB0.NI MB0 VN1
- on-chip com: Zynq - FPGA_MB1.NI MB1 VN1
- Resource Schedule - APC910 - Celeron - Celeron - x86 - Core1 - Plan0
- Resource Schedule - APC910 - Celeron - Celeron - x86 - Core2 - Plan0
- Resource Schedule - Zynq - ARM Cortex A9 - Core0 - Plan0
- Resource Schedule - Zynq - ARM Cortex A9 - Core1 - Plan0
- Resource Schedule - Zynq - FPGA_MB0 - Core_MB0 - Plan0
- Resource Schedule - Zynq - FPGA_MB1 - Core_MB1 - Plan0

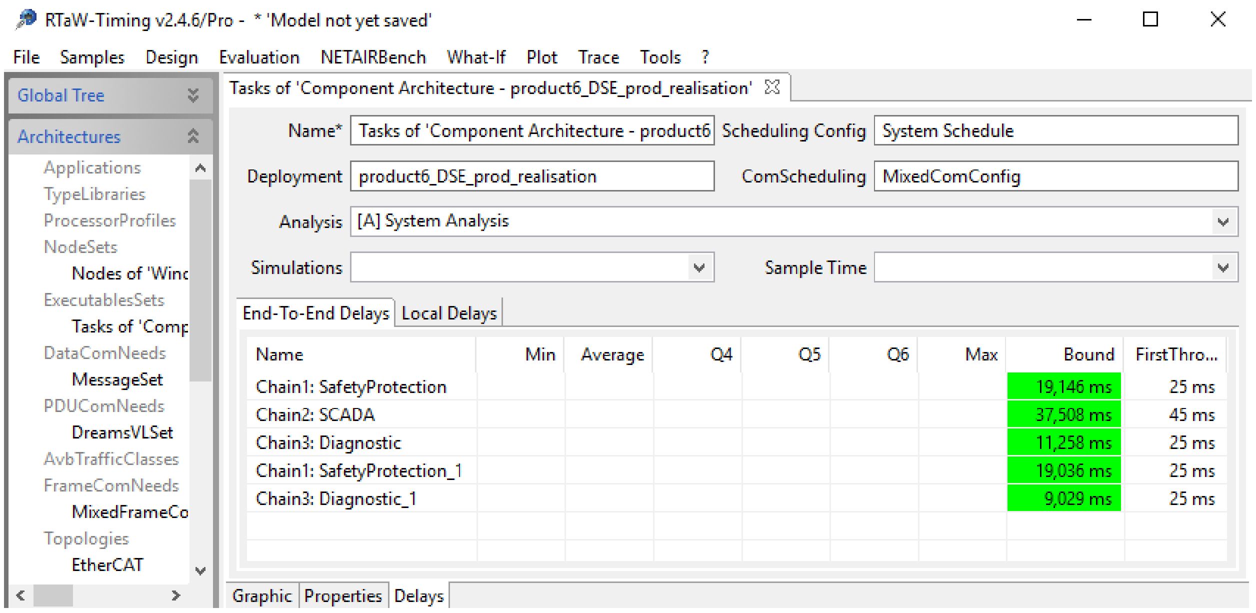
VL 12 (OutSafetyProtection)

Name	VL 12 (OutSafetyProtection)
Comment	
Schedulable Entity	product6_DSE_prod_realisation.
Duration [s]	0,00000005
Trigger	Periodic
Period [s]	0,0078125
Phase [s]	0,000152587891
	Undefined

Figure 16 - WP7: On-chip TT Scheduling parameters added to the SystemSchedule (AF3).

3.4.4 Timing Analysis

Performing worst case analysis of end-to-end delays, in order to verify that latency constraints on the timing chains are satisfied, corresponds to step 11 of the Tool Chain Use Case 1, see [1]. In order to execute the algorithm implemented in RTaW-Timing, the "Timing Evaluation" entry of the context menu of the SystemSchedule must be selected, as explained in Section 4.3.2.5 of [1]. As a result, the complete description of designed system is exported to RTaW-Timing, which executes the worst-case analysis and displays the computed bounds in its GUI, as shown in Figure 17.


RTaW-Timing v2.4.6/Pro - * 'Model not yet saved'

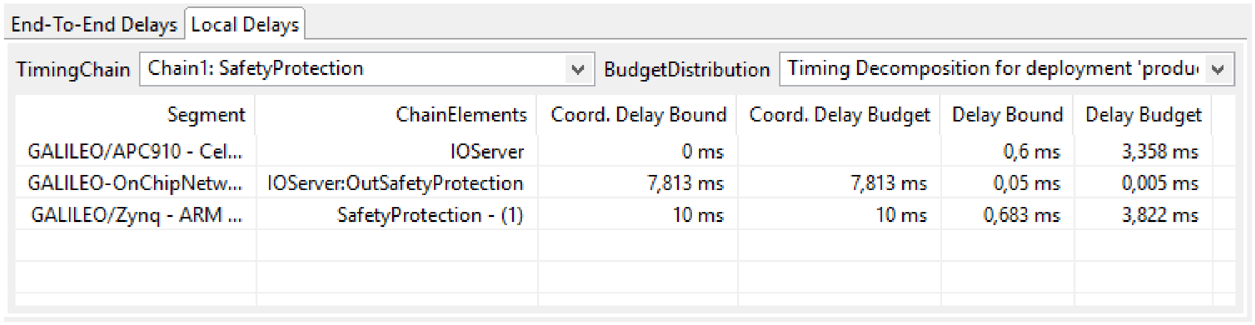
Tasks of 'Component Architecture - product6_DSE_prod_realisation'

Name* Tasks of 'Component Architecture - product6 | Scheduling Config System Schedule
Deployment product6_DSE_prod_realisation | ComScheduling MixedComConfig
Analysis [A] System Analysis
Simulations | Sample Time

Name	Min	Average	Q4	Q5	Q6	Max	Bound	FirstThro...
Chain1: SafetyProtection							19,146 ms	25 ms
Chain2: SCADA							37,508 ms	45 ms
Chain3: Diagnostic							11,258 ms	25 ms
Chain1: SafetyProtection_1							19,036 ms	25 ms
Chain3: Diagnostic_1							9,029 ms	25 ms

Figure 17 - WP7: Bounds on worst-case delays of timing chains (RTaW-Timing).

All bounds are displayed in green, because their values are smaller than the corresponding delay constraint.



TimingChain: Chain1: SafetyProtection
BudgetDistribution: Timing Decomposition for deployment 'produ

Segment	ChainElements	Coord. Delay Bound	Coord. Delay Budget	Delay Bound	Delay Budget
GALILEO/APC910 - Cel...	IOServer	0 ms		0,6 ms	3,358 ms
GALILEO-OnChipNetw...	IOServer:OutSafetyProtection	7,813 ms	7,813 ms	0,05 ms	0,005 ms
GALILEO/Zynq - ARM ...	SafetyProtection - (1)	10 ms	10 ms	0,683 ms	3,822 ms

Figure 18 - WP7: Bounds on worst-case delays of timing chain sub-segments (RTaW-Timing).

Figure 18 shows the bounds on the sub-segments of the timing chain “Safety Protection”, already described in more details in previous sections. It can be noticed that for the on-chip communication, the worst-case bound is equal to the budget. Since no coordination is possible between tasks and on-chip communication, the coordination delay is equal to the message period in the worst case. The budget has been chosen accordingly by the timing decomposition algorithm (Section 3.4.1). Furthermore, since the on-chip schedule is based on phases that avoid any collision, the worst-case traversal time is actually the time needed in case of an empty network. The time budget has been chosen accordingly and thus the bound is equal to the budget. Notice that the comprehensive time (coordination + traversal) is independent of the number of VLs. Therefore it was possible to allocate tight budgets (see also [6], Section 4).

On the other hand, the delays induced by the task segments depend on the processor load. Therefore all available slack with respect to the end-to-end latency constraint has been distributed over the task segments and since there is indeed slack (end-to-end delay bound << end-to-end constraint), the bounds on the worst case delays of the task segments are much lower than the budgets.

3.5 Platform Building Block Configuration File Generation

At this point, task and on-chip communication schedules have been defined and verified and thus the building block configuration files may be generated.

3.5.1 XtratuM configuration files

The generation of the configuration files for the XtratuM hypervisors corresponds to step 12 of the Tool Chain Use Case 1, see [1].

The generation is implemented as AF3 plugin and can be executed by selecting the “XtratuM Xmc file Export” entry from the context menu of the SystemSchedule as shown in Figure 19 (changed since the explanations given in [1]).

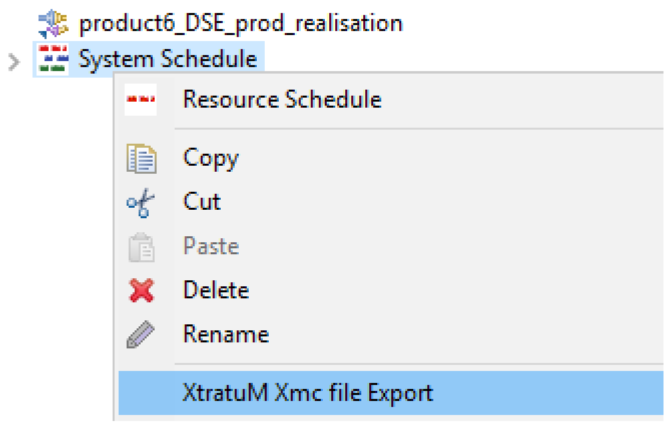
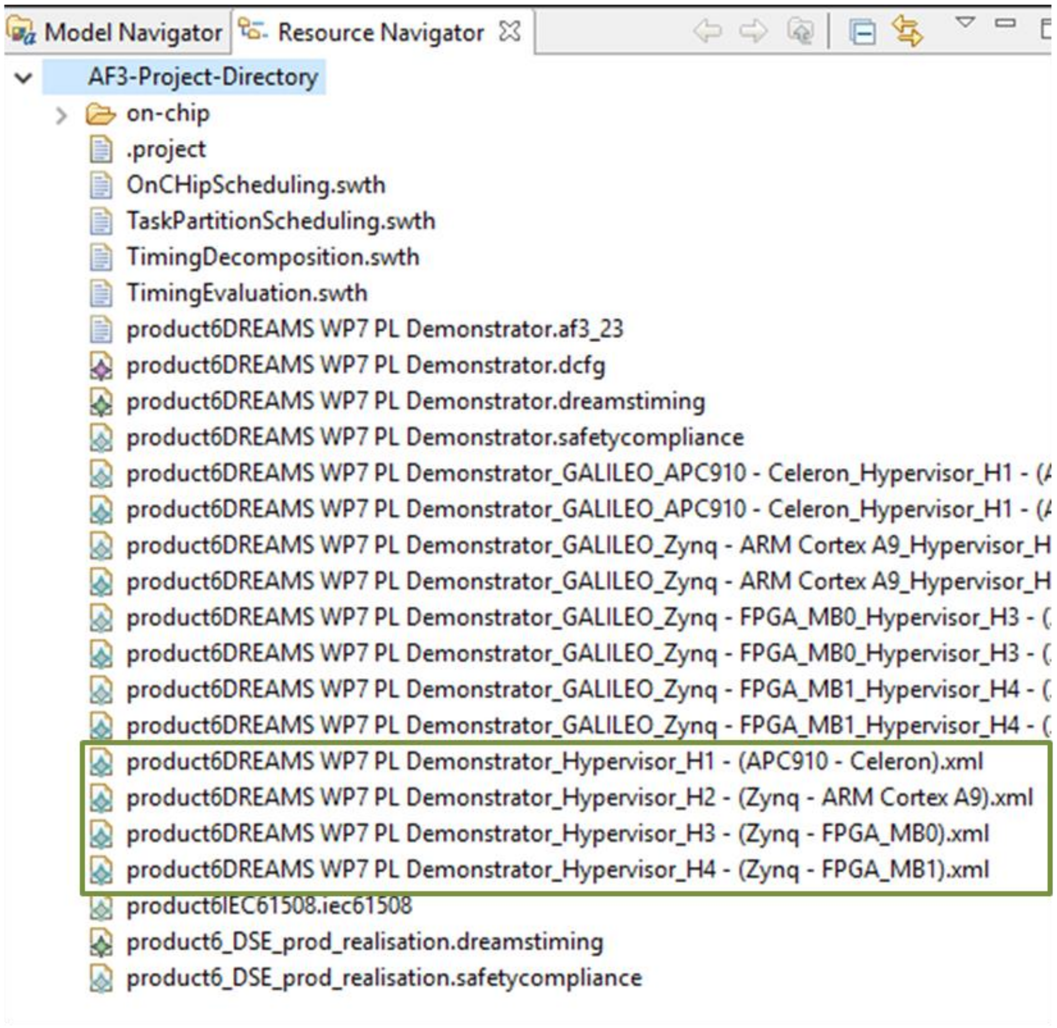


Figure 19 - WP7: XtratuM configuration file export.

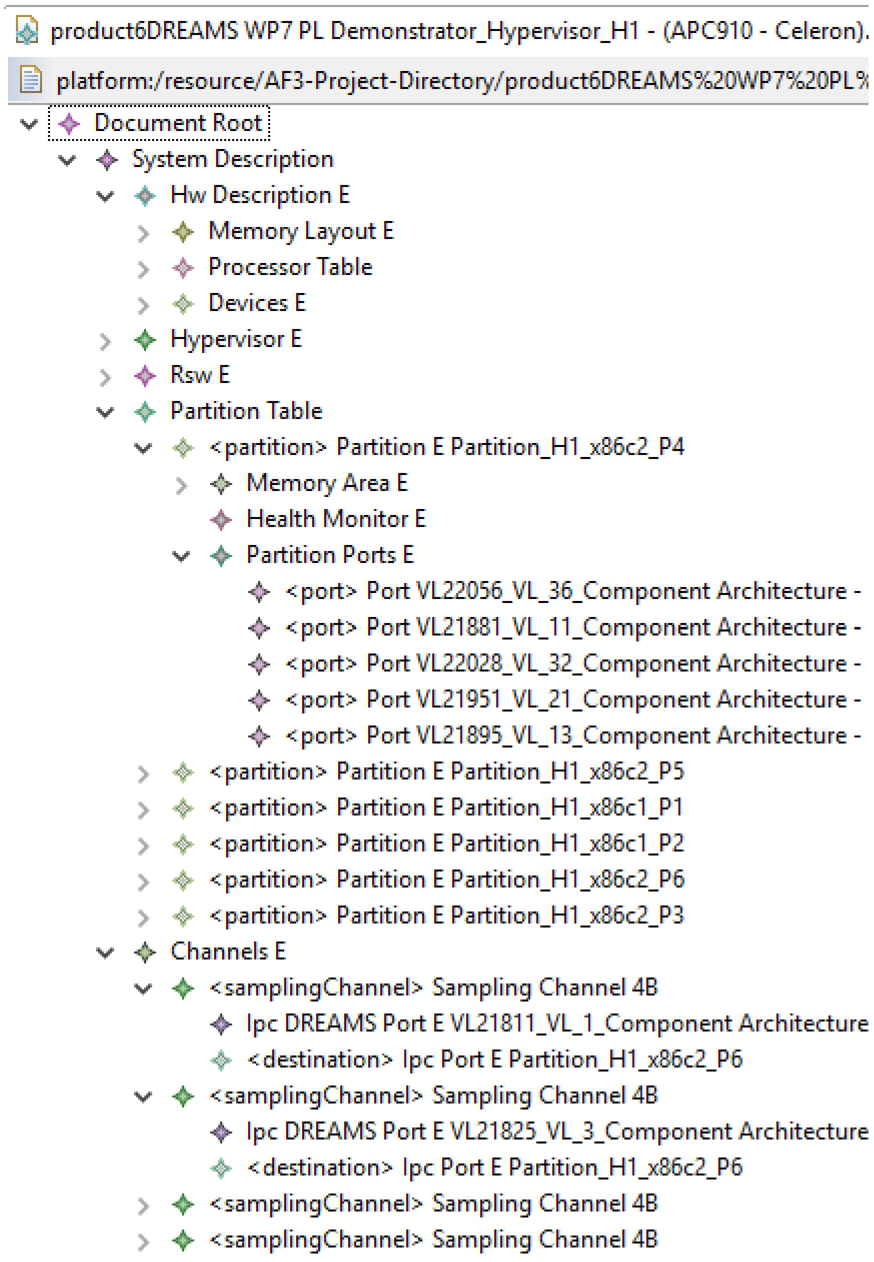
The produced configuration files, one for each Tile, are located in the project folder and can be accessed in the "Resource Navigator" as shown in Figure 20.



AF3-Project-Directory
on-chip
.project
OnCHipScheduling.swth
TaskPartitionScheduling.swth
TimingDecomposition.swth
TimingEvaluation.swth
product6DREAMS WP7 PL Demonstrator.af3_23
product6DREAMS WP7 PL Demonstrator.dcfg
product6DREAMS WP7 PL Demonstrator.dreamstiming
product6DREAMS WP7 PL Demonstrator.safetycompliance
product6DREAMS WP7 PL Demonstrator_GALILEO_APC910 - Celeron_Hypervisor_H1 - (
product6DREAMS WP7 PL Demonstrator_GALILEO_APC910 - Celeron_Hypervisor_H1 - (
product6DREAMS WP7 PL Demonstrator_GALILEO_Zynq - ARM Cortex A9_Hypervisor_H
product6DREAMS WP7 PL Demonstrator_GALILEO_Zynq - ARM Cortex A9_Hypervisor_H
product6DREAMS WP7 PL Demonstrator_GALILEO_Zynq - FPGA_MB0_Hypervisor_H3 - (
product6DREAMS WP7 PL Demonstrator_GALILEO_Zynq - FPGA_MB0_Hypervisor_H3 - (
product6DREAMS WP7 PL Demonstrator_GALILEO_Zynq - FPGA_MB1_Hypervisor_H4 - (
product6DREAMS WP7 PL Demonstrator_GALILEO_Zynq - FPGA_MB1_Hypervisor_H4 - (
product6DREAMS WP7 PL Demonstrator_Hypervisor_H1 - (APC910 - Celeron).xml
product6DREAMS WP7 PL Demonstrator_Hypervisor_H2 - (Zynq - ARM Cortex A9).xml
product6DREAMS WP7 PL Demonstrator_Hypervisor_H3 - (Zynq - FPGA_MB0).xml
product6DREAMS WP7 PL Demonstrator_Hypervisor_H4 - (Zynq - FPGA_MB1).xml
product6IEC61508.iec61508
product6_DSE_prod_realisation.dreamstiming
product6_DSE_prod_realisation.safetycompliance

Figure 20 - WP7: Generated Xtratum configuration files.

Figure 21 shows the configuration file for the "APC910" Tile in the EMF tree viewer. Notice that the output of Xoncrete that contains the task and partition schedules (Section 3.4.2) is already produced in the XML format of the XtratuM hypervisor, but with respect to these files, the finally generated configuration files also contain communication related data, namely the partition ports and the communication channels between the partition ports and also the ports for the on-chip communication.



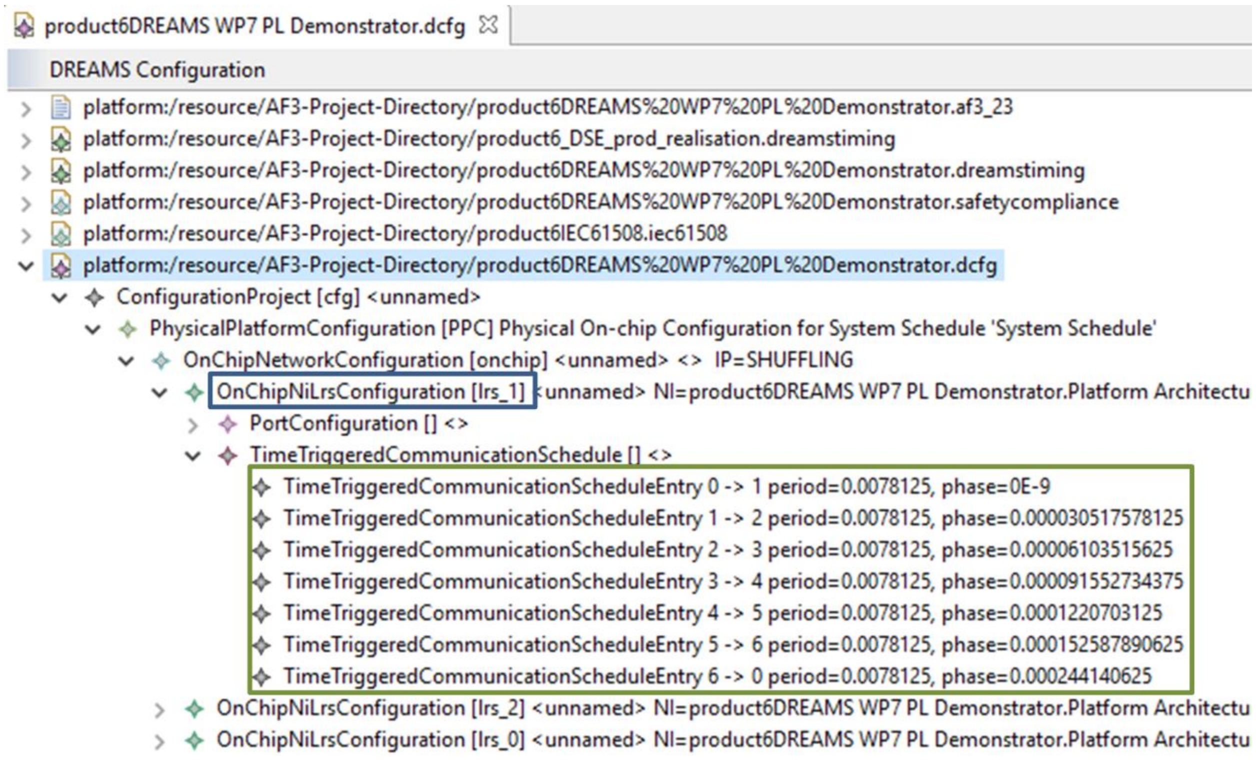
```
product6DREAMS WP7 PL Demonstrator_Hypervisor_H1 - (APC910 - Celeron).
platform:/resource/AF3-Project-Directory/product6DREAMS%20WP7%20PL%
Document Root
  System Description
    Hw Description E
      Memory Layout E
      Processor Table
      Devices E
    Hypervisor E
    Rsw E
    Partition Table
      <partition> Partition E Partition_H1_x86c2_P4
        Memory Area E
        Health Monitor E
        Partition Ports E
          <port> Port VL22056_VL_36_Component Architecture -
          <port> Port VL21881_VL_11_Component Architecture -
          <port> Port VL22028_VL_32_Component Architecture -
          <port> Port VL21951_VL_21_Component Architecture -
          <port> Port VL21895_VL_13_Component Architecture -
      <partition> Partition E Partition_H1_x86c2_P5
      <partition> Partition E Partition_H1_x86c1_P1
      <partition> Partition E Partition_H1_x86c1_P2
      <partition> Partition E Partition_H1_x86c2_P6
      <partition> Partition E Partition_H1_x86c2_P3
    Channels E
      <samplingChannel> Sampling Channel 4B
        Ipc DREAMS Port E VL21811_VL_1_Component Architecture
        <destination> Ipc Port E Partition_H1_x86c2_P6
      <samplingChannel> Sampling Channel 4B
        Ipc DREAMS Port E VL21825_VL_3_Component Architecture
        <destination> Ipc Port E Partition_H1_x86c2_P6
      <samplingChannel> Sampling Channel 4B
      <samplingChannel> Sampling Channel 4B
```

Figure 21 - WP7: Tree view of a XtratuM configuration file.

3.5.2 On-chip network communication configuration files

The generation of the configuration files for the on-chip communication, corresponds to step 13 of the Tool Chain Use Case 1, see [1].

The generation is implemented as a two-step process, as explained in Section 3.3.1.5 of [7]. First the "Generate 'Physical On-Chip Communication' Configuration Model" entry from the context menu of the SystemSchedule must be selected in order to generate the configuration model, which is visible in the "Resoure Navigator" (.dcfg file). Figure 22 shows transmission phases for TT VL created in step 9 (Section 3.4.3) for the LRS of one of the NIs.



```
product6DREAMS WP7 PL Demonstrator.dcfg
DREAMS Configuration
> platform:/resource/AF3-Project-Directory/product6DREAMS%20WP7%20PL%20Demonstrator.af3_23
> platform:/resource/AF3-Project-Directory/product6_DSE_prod_realisation.dreamstiming
> platform:/resource/AF3-Project-Directory/product6DREAMS%20WP7%20PL%20Demonstrator.dreamstiming
> platform:/resource/AF3-Project-Directory/product6DREAMS%20WP7%20PL%20Demonstrator.safetycompliance
> platform:/resource/AF3-Project-Directory/product6IEC61508.iec61508
v platform:/resource/AF3-Project-Directory/product6DREAMS%20WP7%20PL%20Demonstrator.dcfg
  v ConfigurationProject [cfg] <unnamed>
    v PhysicalPlatformConfiguration [PPC] Physical On-chip Configuration for System Schedule 'System Schedule'
      v OnChipNetworkConfiguration [onchip] <unnamed> <>  IP=SHUFFLING
        v OnChipNiLrsConfiguration [lrs_1] <unnamed> NI=product6DREAMS WP7 PL Demonstrator.Platform Architectu
          > PortConfiguration [] <>
          v TimeTriggeredCommunicationSchedule [] <>
              TimeTriggeredCommunicationScheduleEntry 0 -> 1 period=0.0078125, phase=0E-9
              TimeTriggeredCommunicationScheduleEntry 1 -> 2 period=0.0078125, phase=0.000030517578125
              TimeTriggeredCommunicationScheduleEntry 2 -> 3 period=0.0078125, phase=0.00006103515625
              TimeTriggeredCommunicationScheduleEntry 3 -> 4 period=0.0078125, phase=0.000091552734375
              TimeTriggeredCommunicationScheduleEntry 4 -> 5 period=0.0078125, phase=0.0001220703125
              TimeTriggeredCommunicationScheduleEntry 5 -> 6 period=0.0078125, phase=0.000152587890625
              TimeTriggeredCommunicationScheduleEntry 6 -> 0 period=0.0078125, phase=0.000244140625
        > OnChipNiLrsConfiguration [lrs_2] <unnamed> NI=product6DREAMS WP7 PL Demonstrator.Platform Architectu
        > OnChipNiLrsConfiguration [lrs_0] <unnamed> NI=product6DREAMS WP7 PL Demonstrator.Platform Architectu
```

Figure 22 - WP7: Configuration model for the on-chip communication.

The second step consists in transforming the configuration model into text files. For this purpose, the "Run configuration generation framework" entry from the context menu of the model need to be selected as shown in Figure 23 (not explained in [7]).

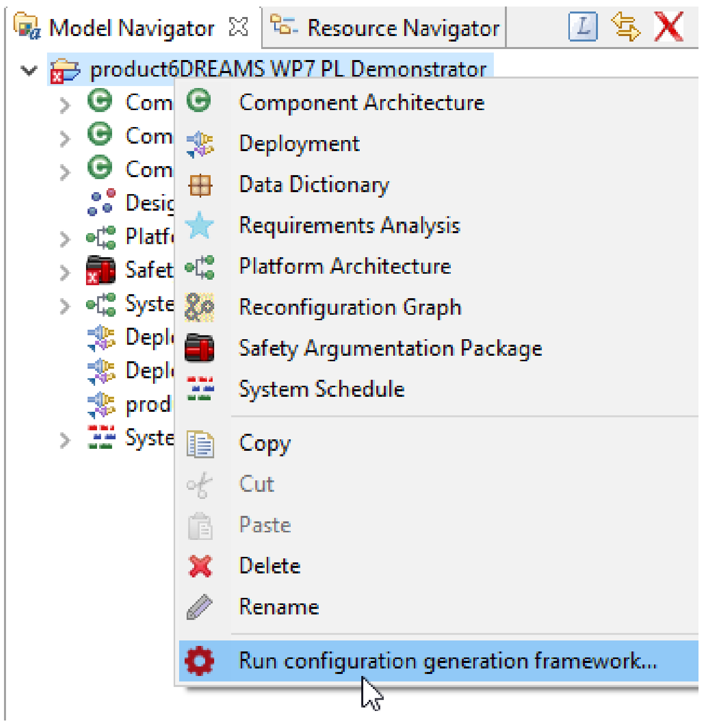
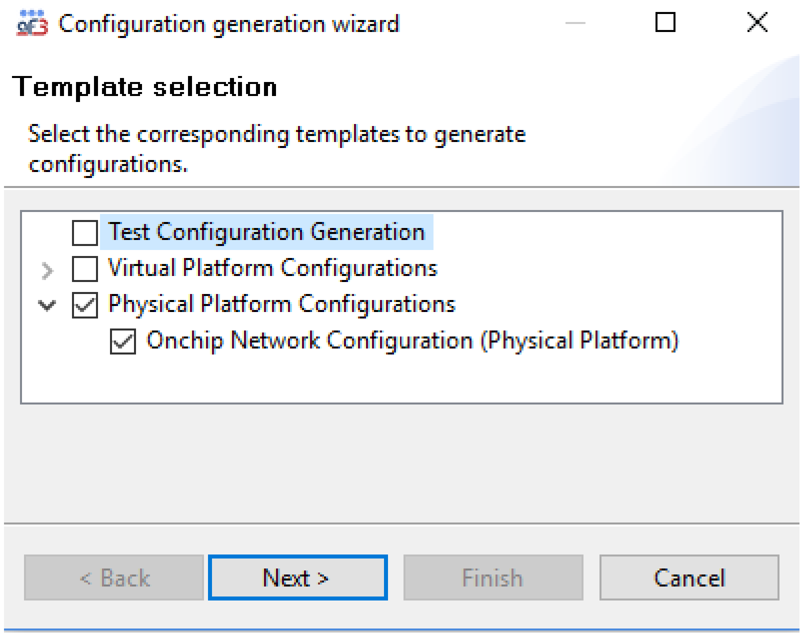


Figure 23 - WP7: Running the configuration generation framework.

In dialog must be selected the “Onchip Network Configuration (Physical Platform)” entry, see Figure 24.


Configuration generation wizard

Template selection

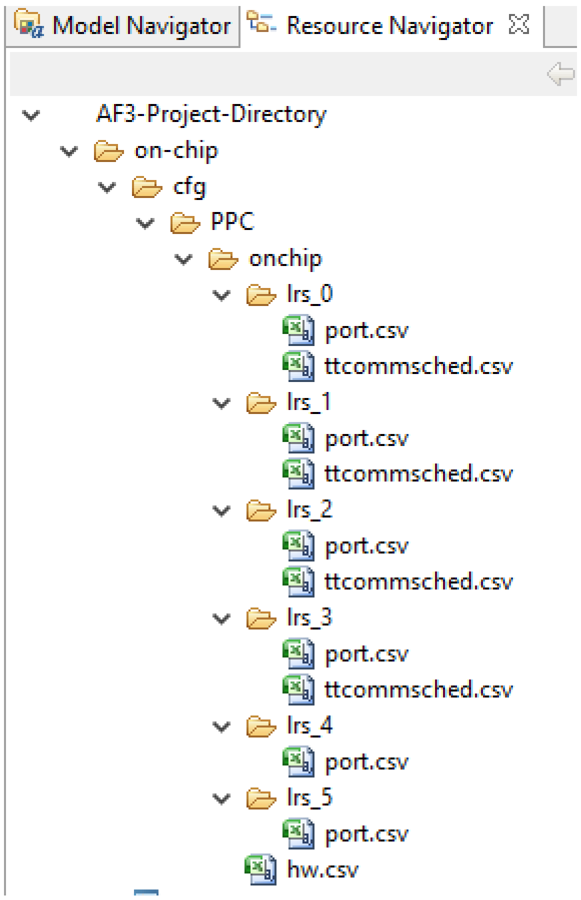
Select the corresponding templates to generate configurations.

- [ ] Test Configuration Generation
- [ ] Virtual Platform Configurations
- [x] Physical Platform Configurations
  - [x] Onchip Network Configuration (Physical Platform)

Figure 24 - WP7: Configuration generation framework Wizard.

The “Next” button moves to the selection of an output folder. We suggest creating a sub-folder in the project folder. Clicking “Finish” starts the generation, which produces a structure of nested text files with global and NI specific configuration files (see Figure 25), suitable for DRCSV2BIN (Section 3.3.1.2 in

[7]), which allows to translate the textual configuration files into binary files for the actual configuration of the platform building block.



```
AF3-Project-Directory
  on-chip
    cfg
      PPC
        onchip
          lrs_0
            port.csv
            ttcommsched.csv
          lrs_1
            port.csv
            ttcommsched.csv
          lrs_2
            port.csv
            ttcommsched.csv
          lrs_3
            port.csv
            ttcommsched.csv
          lrs_4
            port.csv
          lrs_5
            port.csv
          hw.csv
```

Figure 25 - WP7: Generated textual configuration files for the on-chip communication.

4 Application to an avionics use case

4.1 Introduction

In this section we apply the tool chain use case 2 “Scheduling Configuration with Resource Management” defined D4.4.1 [1] to an avionics use case, which is an extension of the ROSACE case study [8]. Here we have set up a system with three applications on two multi-core platforms to illustrate the inter-node communication between applications when reconfiguration is considered.

With respect to the wind power use case described in Section 3, the definition of applications and resources (Section 4.2) considers only one product and the deployment is defined manually. But in this use case we create several deployments and resource schedules, managed by the local (LRM) and global (GRM) resource manager (see D2.2.2 [9]) so as to guarantee a certain degree of continuity of services for (critical) application in case of core failures (Section 4.4.2). The generation of the corresponding configuration files of the platform services is also illustrated (Section 4.5).

4.2 Applications and Resources

The construction of the DREAMS logical and platform architecture models in the AutoFOCUS3 (AF3) DREAMS edition tools is achieved via the context menu of an AF3 model, which is described in more detail in [1], [10], and [11]. Here, we will focus on a description of the ROSACE application models that include the application’s logical architecture, a timing requirement specification, and a target platform model.

4.2.1 Logical Architecture

The logical architecture of the ROSACE application consists of two parts: The application components and the resource management components that provide the runtime resource and monitoring DREAMS services (see Figure 26). The ROSACE application consists of a non-critical MPEG video decoder, an order generator and a critical control sub-application (see Figure 27). The decoder is a single black box component and represents a large resource consumer. The control application description has been taken from [8] and is modelled by means of the DREAMS meta-model whereby the components are considered black-boxes with annotated properties. The order generator component represents an additional input to the engine control that communicates desired application operation modes to the control application, e.g. change flight course.

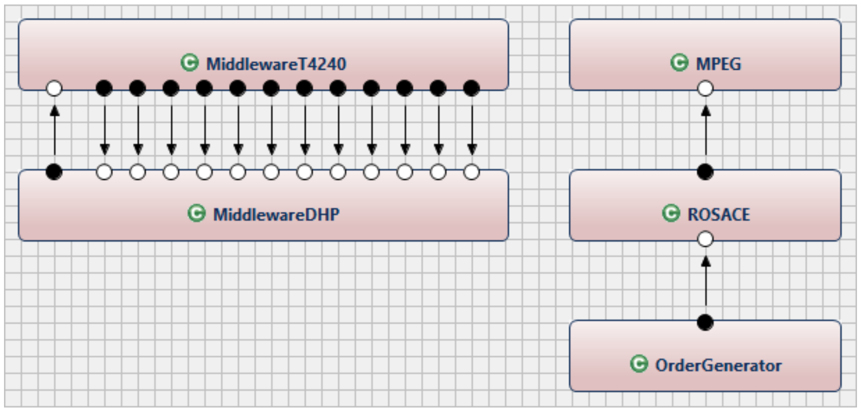


Figure 26 - Top-level view on the logical components: ROSACE logical Architecture.

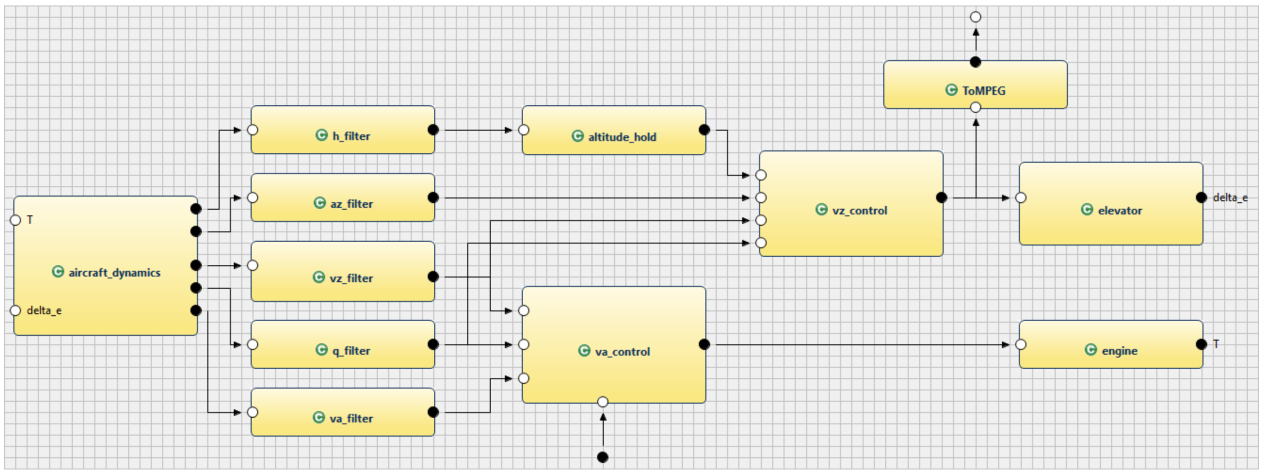


Figure 27 - ROSACE control application

The DREAMS platform services are modelled as components of the logical architectures such that they can be represented in the system deployment and schedules. Due to the fact that we have two nodes available in the platform architecture, the model contains two middleware components: a middleware component for the DHP that hosts the MPEG decoder, and a middleware component hosting the ROSACE control application (see Figure 26). The global resource manager (GRM) component is located in the DHP's middleware (see Figure 28) to which the local resource managers (LRMs) of the DHP and the LRMs of the T4240 (see Figure 29) are connected.

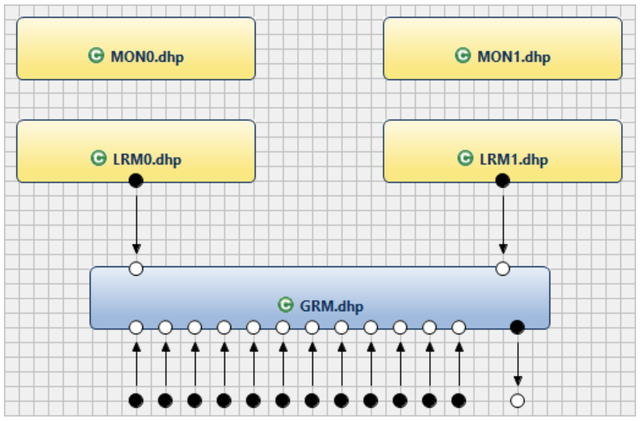


Figure 28 - Middleware components of the DHP.

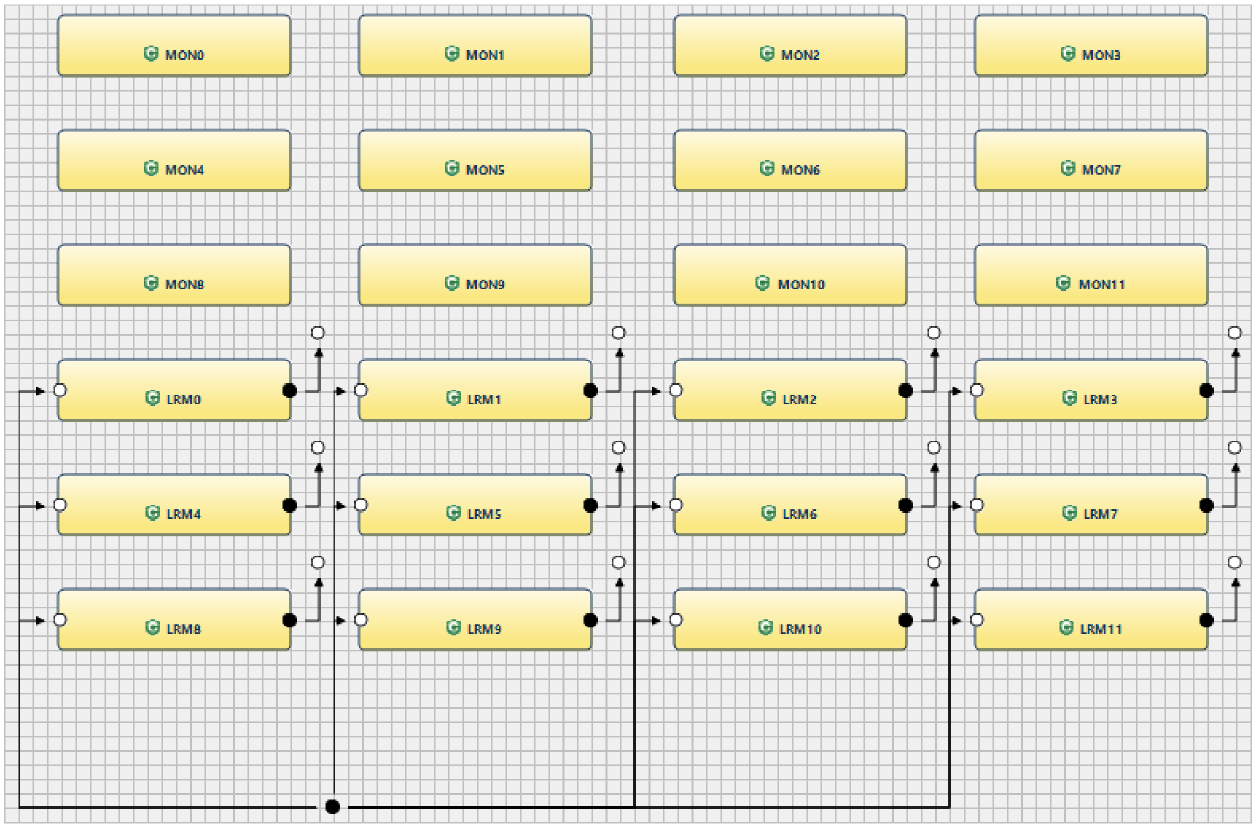
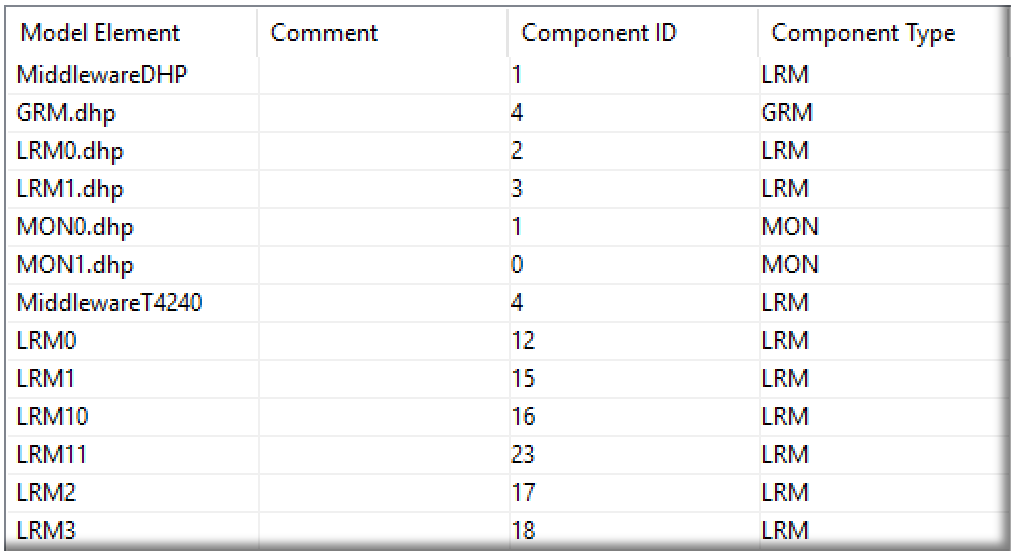


Figure 29 - Middleware components of the T4240.

Each component in the model is decorated with an annotation that specifies the application type: It can be either an application, the global resource manager, a local resource manager, or a monitor. Figure 30 lists the component types for the MPEG component (first two rows in Figure 30; Type: Application) and the resource management component types of the two middleware components. The GRM located in the middleware component of the DHP is marked as such (see row 4 in Figure 30).



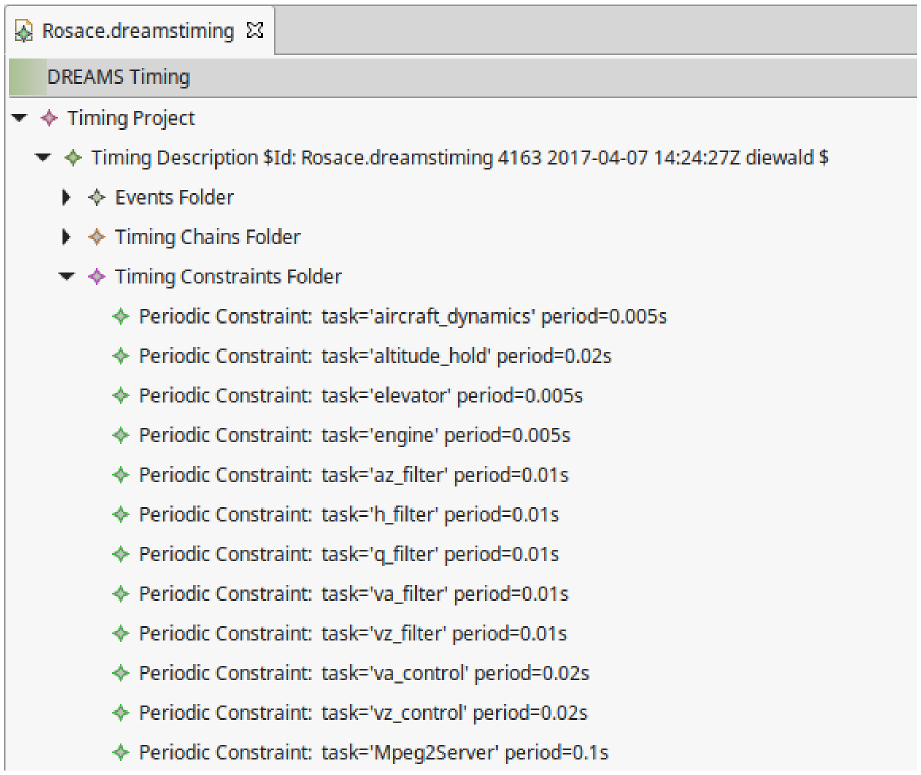
Model Element	Comment	Component ID	Component Type
MiddlewareDHP		1	LRM
GRM.dhp		4	GRM
LRM0.dhp		2	LRM
LRM1.dhp		3	LRM
MON0.dhp		1	MON
MON1.dhp		0	MON
MiddlewareT4240		4	LRM
LRM0		12	LRM
LRM1		15	LRM
LRM10		16	LRM
LRM11		23	LRM
LRM2		17	LRM
LRM3		18	LRM

Figure 30 - Logical component types.

4.2.2 Timing Requirements

The definition of timing requirements corresponds to step 2 of the Tool Chain Use Cases 1 and 2, see [1].

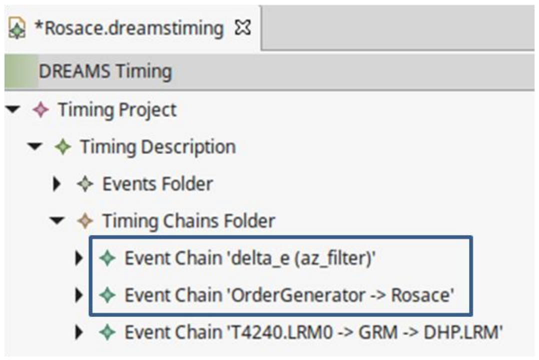
All application tasks of the ROSACE demonstrator shall be executed periodically (5ms, 10ms, 20ms, 100ms, 1s). These timing requirements are expressed as Periodic Constraints and specified through the Timing Model editor integrated into AF3, see Figure 31



```
Rosace.dreamstiming
DREAMS Timing
▾ Timing Project
  ▾ Timing Description $Id: Rosace.dreamstiming 4163 2017-04-07 14:24:27Z diewald $
    ▸ Events Folder
    ▸ Timing Chains Folder
    ▾ Timing Constraints Folder
        Periodic Constraint:  task='aircraft_dynamics' period=0.005s
        Periodic Constraint:  task='altitude_hold' period=0.02s
        Periodic Constraint:  task='elevator' period=0.005s
        Periodic Constraint:  task='engine' period=0.005s
        Periodic Constraint:  task='az_filter' period=0.01s
        Periodic Constraint:  task='h_filter' period=0.01s
        Periodic Constraint:  task='q_filter' period=0.01s
        Periodic Constraint:  task='va_filter' period=0.01s
        Periodic Constraint:  task='vz_filter' period=0.01s
        Periodic Constraint:  task='va_control' period=0.02s
        Periodic Constraint:  task='vz_control' period=0.02s
        Periodic Constraint:  task='Mpeg2Server' period=0.1s
```

Figure 31 - ROSACE: Periodicity constraints on application task.

Two Timing Chains related to applications have been considered, see Figure 32. The first one only spans tasks of the ROSACE application as illustrated in Figure 33, whereas the second covers the executions of a task of the application “OrderGenerator” and a task of the application “ROSACE” with the communication of some data from the former to the later.



```
*Rosace.dreamstiming
DREAMS Timing
▾ Timing Project
  ▾ Timing Description
    ▸ Events Folder
    ▾ Timing Chains Folder
      ▸ Event Chain 'delta_e (az_filter)'
      ▸ Event Chain 'OrderGenerator -> Rosace'
      ▸ Event Chain 'T4240.LRM0 -> GRM -> DHP.LRM'
```

Figure 32 - ROSACE: Timing Chains.

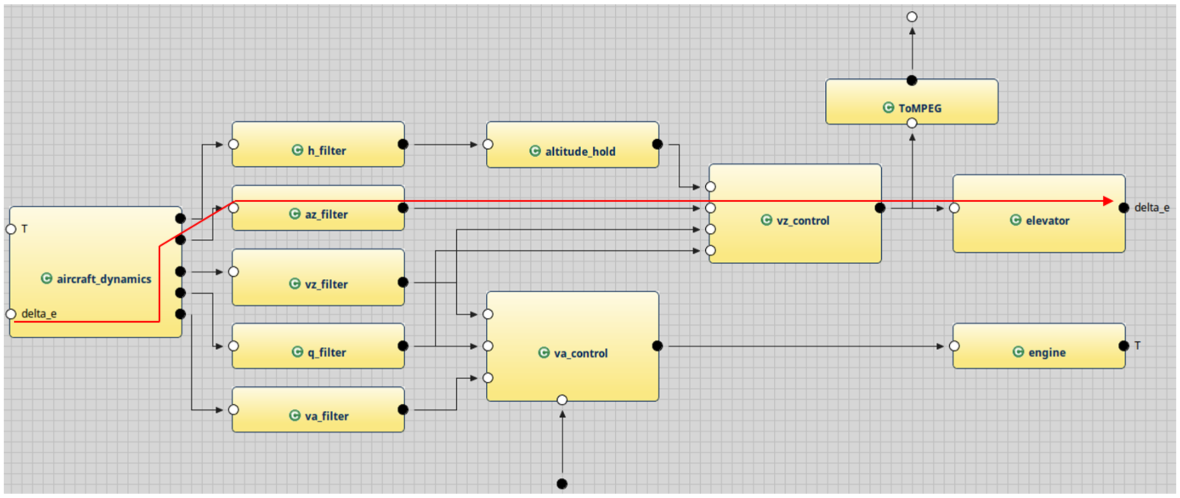


Figure 33 - ROSACE: Timing chains spanning several task of one application

Timing Chains may be declared with the help of the entry “Declare a Timing Chain” from the context menu of the component architecture, see Figure 34 (not shown in [1]).

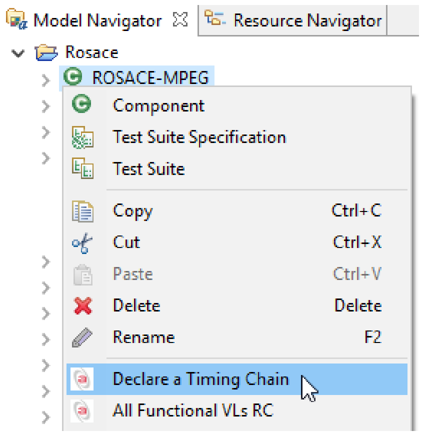
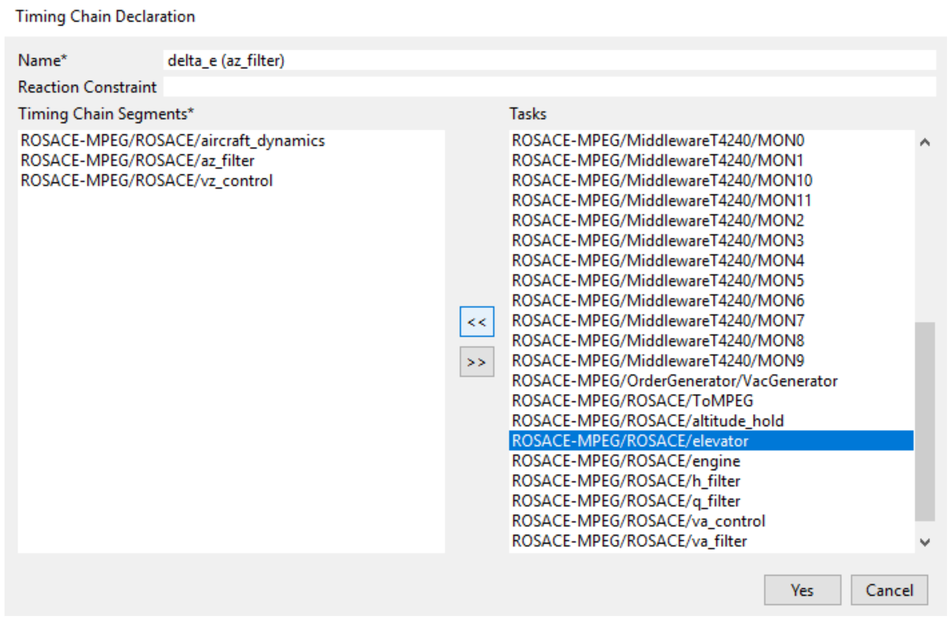


Figure 34 – Context menu for declaring a Timing Chain.

In the creation dialog (shown in Figure 35), a name must be provided and the list of involved tasks, in the order in which information flows from the start to the end of chain. An optional reaction constraint can be provided in seconds.


Timing Chain Declaration

Name* delta_e (az_filter)

Reaction Constraint

Timing Chain Segments*
ROSACE-MPEG/ROSACE/aircraft_dynamics
ROSACE-MPEG/ROSACE/az_filter
ROSACE-MPEG/ROSACE/vz_control

Tasks
ROSACE-MPEG/MiddlewareT4240/MON0
ROSACE-MPEG/MiddlewareT4240/MON1
ROSACE-MPEG/MiddlewareT4240/MON10
ROSACE-MPEG/MiddlewareT4240/MON11
ROSACE-MPEG/MiddlewareT4240/MON2
ROSACE-MPEG/MiddlewareT4240/MON3
ROSACE-MPEG/MiddlewareT4240/MON4
ROSACE-MPEG/MiddlewareT4240/MON5
ROSACE-MPEG/MiddlewareT4240/MON6
ROSACE-MPEG/MiddlewareT4240/MON7
ROSACE-MPEG/MiddlewareT4240/MON8
ROSACE-MPEG/MiddlewareT4240/MON9
ROSACE-MPEG/OrderGenerator/VacGenerator
ROSACE-MPEG/ROSACE/ToMPEG
ROSACE-MPEG/ROSACE/altitude_hold
ROSACE-MPEG/ROSACE/elevator
ROSACE-MPEG/ROSACE/engine
ROSACE-MPEG/ROSACE/h_filter
ROSACE-MPEG/ROSACE/q_filter
ROSACE-MPEG/ROSACE/va_control
ROSACE-MPEG/ROSACE/va_filter

Figure 35 - Declaration of a timing chain.

Resource management tasks (LRM, MON and GRM) shall all be executed with the same periodicity. The corresponding periodicity constraints can be generated through a context menu of the nominal Deployment, as shown in Figure 36. The produced constraints are added to the Timing Model.

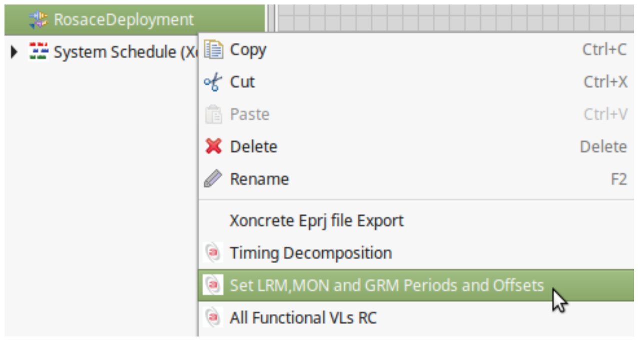


Figure 36 - Context menu entry for specifying period of resource management tasks.

For the ROSACE Use Case a resource management period of 1s has been chosen, which is equal to the smallest common multiple of the functional task periods (MAF).

Since furthermore, LRM tasks must be executed at the end of the MAF, the timing constraint generator also adds an offset constraint that induces an execution start at t = MAF - WCET when the task schedules are generated (Section 4.4.1).

When a core fails, the master LRM of the concerned Tile detects the event, applies a local reconfiguration (if possible) and informs the GRM (possibly running on another Tile) about the change. As a response, the GRM might order local reconfigurations on other Tiles, by sending an appropriate message to them. In this context it is interesting to consider the timing chain that spans the execution of an LRM, followed by the sending of a message to the GRM, the execution of the GRM, followed by the sending of an order message to some LRM and finally the execution of that LRM, which may trigger a scheduling plan change. We have considered such a chain: from LRM0 of the T4240 node until the LRM0 of the DHP node. It is the last Timing Chain “T4240.LRM0 -> GRM -> DHP.LRM0” at the bottom of Figure 32.

4.2.3 Platform Architecture

The DREAMS platform meta-model consists of a hardware platform meta-model and a system software meta-model, i.e., the middleware [10]. The hardware model of the ROSACE example application is a single cluster that consists of a model of a DHP and a T4240 (see Figure 37) that are connected by a TTEthernet network model. For the DHP, only the ARM A9 Tile is modelled since the optional Microblaze processors are not used in the example. The T4240 node consists primarily of the T4240 Tile that has 12 Cores.

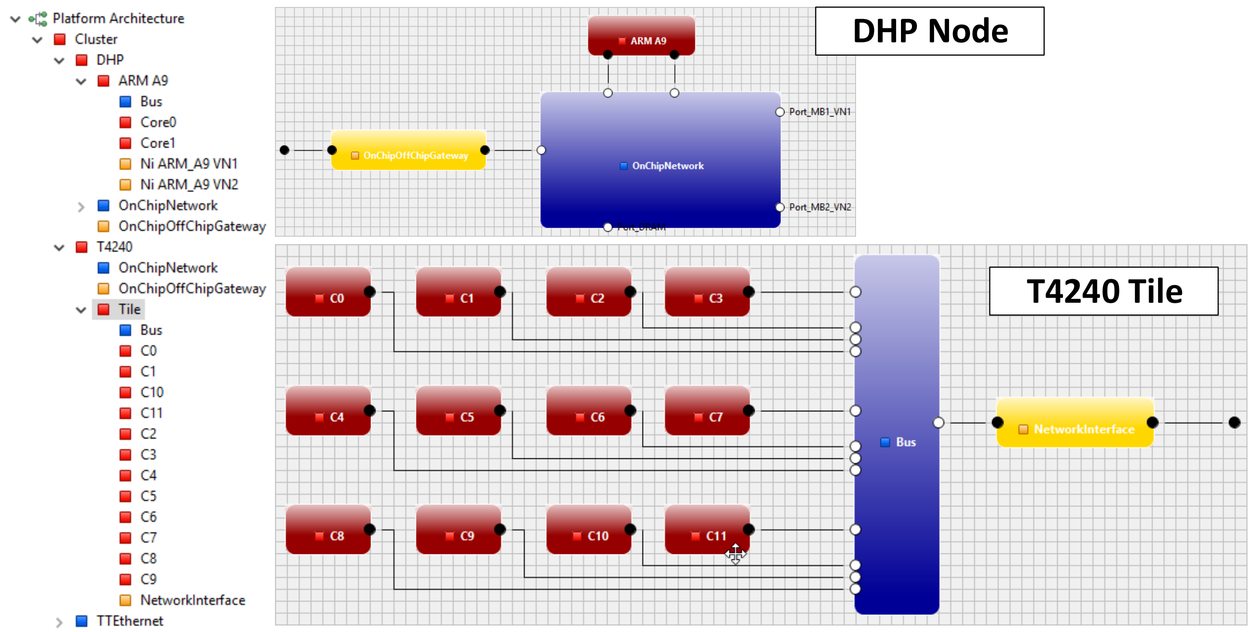


Figure 37: ROSACE HW platform model: DHP node and T4240 Tile.

4.2.4 System Software

The system software model of the ROSACE example consists of two Hypervisors: one assigned to the ARM A9 Tile of the DHP (see Figure 38), and to the T4240 Tile (see Figure 39). Each Hypervisor hosts a partition for each of the resource manager components, and partitions for the MPEG and the ROSACE application components. These partitions define the set of instantiable partitions that manifest in the configuration files for the Hypervisor. Here, a LRM and a monitor is defined for each of the cores present in the system. As discussed in [10], Hypervisors and Tiles (and Partitions to Cores) are assigned to each other by so-called Resource Link annotations.

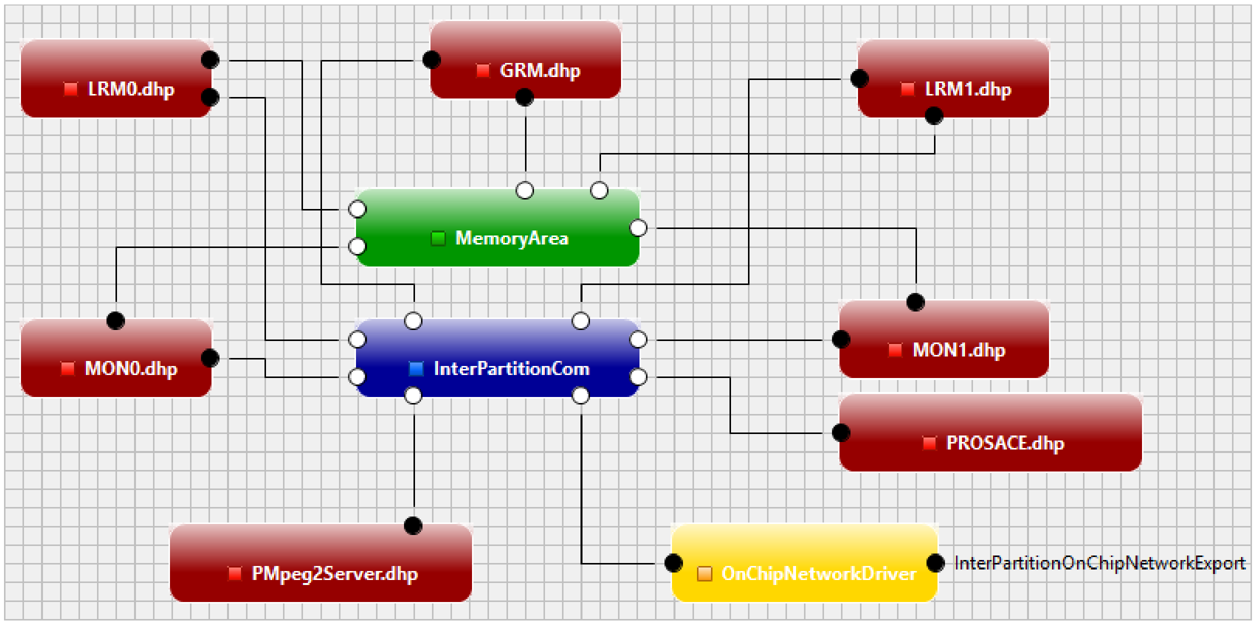


Figure 38: Partitions and virtualized memory and communication elements of the ARM A9's Hypervisor.

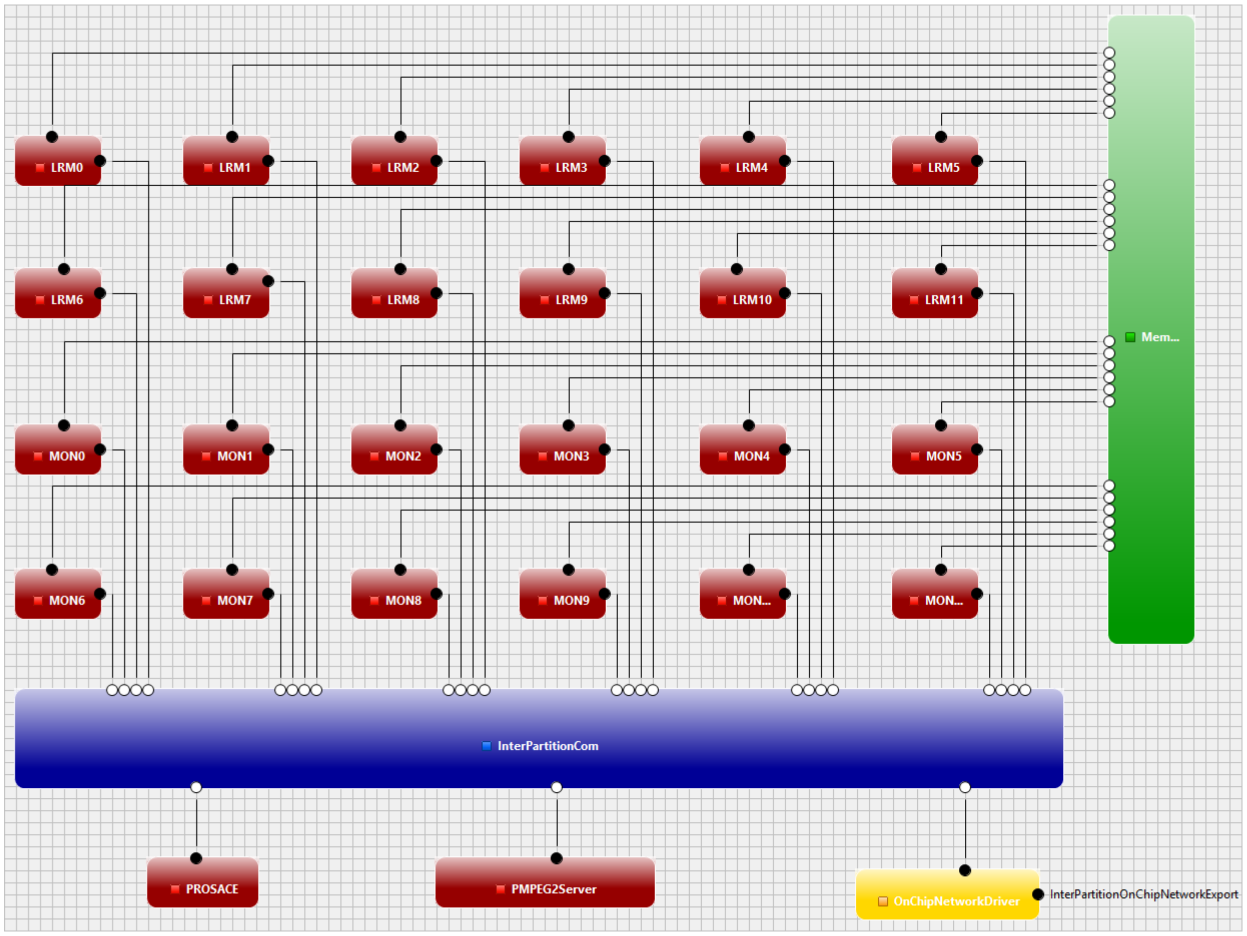
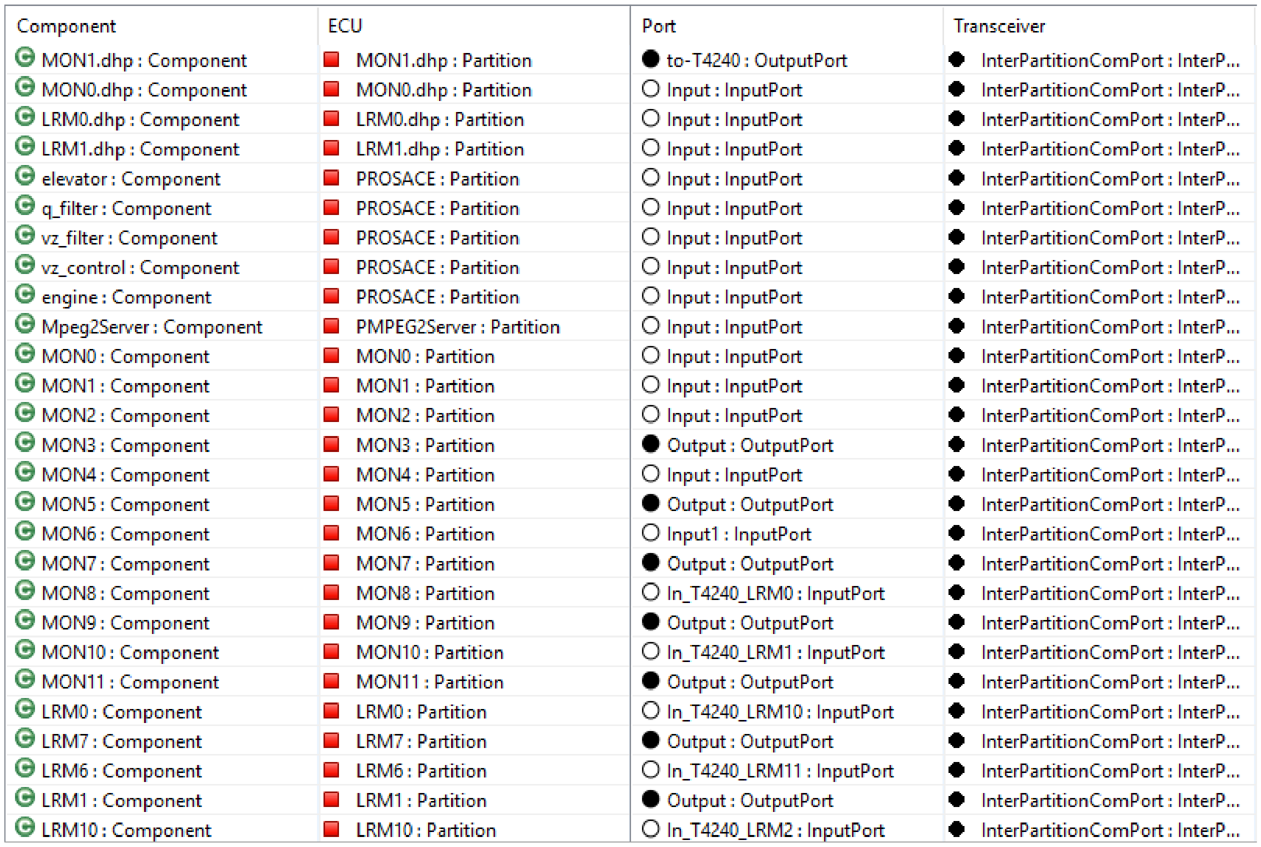


Figure 39: Partitions and virtualized memory and communication elements of the T4240's Hypervisor.

4.3 Deployment

The DREAMS deployment meta-model defines component-to-execution unit allocations, an application's Virtual Links, and the mapping of logical in- and output ports to the in- and outputs of the target platform (see [10],[11]). In contrast to the Deployments that are generated by the DSE for the wind power demonstrator application (see Section 3.3.2), the initial Deployment for the ROSACE example is defined mainly manually: the component-to-execution unit mapping is done using drag and drop in the deployment editor, while the Virtual Links are generated.

As noted in the previous section, each resource management component is assigned to a separate partition (see Figure 40). The actual applications of the ROSACE example, the MPEG component and the components, constituting the ROSACE control application, are assigned to partitions hosted on the Hypervisor of the T4240 Tile (PROSACE and PMPEG2Server; see the missing ".dhp" suffix in the table shown in Figure 40). Hence, the counter part of these partitions present in the Hypervisor model assigned to the DHP Tile are fall-back partitions that may be used by the reconfiguration methods. The failure mode calculation uses these partitions to derive alternative deployment schedules that may use these partitions if it is required by some failure scenario.



Component	ECU	Port	Transceiver
MON1.dhp : Component	MON1.dhp : Partition	to-T4240 : OutputPort	InterPartitionComPort : InterP...
MON0.dhp : Component	MON0.dhp : Partition	Input : InputPort	InterPartitionComPort : InterP...
LRM0.dhp : Component	LRM0.dhp : Partition	Input : InputPort	InterPartitionComPort : InterP...
LRM1.dhp : Component	LRM1.dhp : Partition	Input : InputPort	InterPartitionComPort : InterP...
elevator : Component	PROSACE : Partition	Input : InputPort	InterPartitionComPort : InterP...
q_filter : Component	PROSACE : Partition	Input : InputPort	InterPartitionComPort : InterP...
vz_filter : Component	PROSACE : Partition	Input : InputPort	InterPartitionComPort : InterP...
vz_control : Component	PROSACE : Partition	Input : InputPort	InterPartitionComPort : InterP...
engine : Component	PROSACE : Partition	Input : InputPort	InterPartitionComPort : InterP...
Mpeg2Server : Component	PMPEG2Server : Partition	Input : InputPort	InterPartitionComPort : InterP...
MON0 : Component	MON0 : Partition	Input : InputPort	InterPartitionComPort : InterP...
MON1 : Component	MON1 : Partition	Input : InputPort	InterPartitionComPort : InterP...
MON2 : Component	MON2 : Partition	Input : InputPort	InterPartitionComPort : InterP...
MON3 : Component	MON3 : Partition	Output : OutputPort	InterPartitionComPort : InterP...
MON4 : Component	MON4 : Partition	Input : InputPort	InterPartitionComPort : InterP...
MON5 : Component	MON5 : Partition	Output : OutputPort	InterPartitionComPort : InterP...
MON6 : Component	MON6 : Partition	Input1 : InputPort	InterPartitionComPort : InterP...
MON7 : Component	MON7 : Partition	Output : OutputPort	InterPartitionComPort : InterP...
MON8 : Component	MON8 : Partition	In_T4240_LRM0 : InputPort	InterPartitionComPort : InterP...
MON9 : Component	MON9 : Partition	Output : OutputPort	InterPartitionComPort : InterP...
MON10 : Component	MON10 : Partition	In_T4240_LRM1 : InputPort	InterPartitionComPort : InterP...
MON11 : Component	MON11 : Partition	Output : OutputPort	InterPartitionComPort : InterP...
LRM0 : Component	LRM0 : Partition	In_T4240_LRM10 : InputPort	InterPartitionComPort : InterP...
LRM7 : Component	LRM7 : Partition	Output : OutputPort	InterPartitionComPort : InterP...
LRM6 : Component	LRM6 : Partition	In_T4240_LRM11 : InputPort	InterPartitionComPort : InterP...
LRM1 : Component	LRM1 : Partition	Output : OutputPort	InterPartitionComPort : InterP...
LRM10 : Component	LRM10 : Partition	In_T4240_LRM2 : InputPort	InterPartitionComPort : InterP...

Figure 40: Excerpt of the Component-Execution Unit & Port allocations.

Virtual Links are part of the Deployment meta-model [11]. Their generation is triggered from the context menu of the Virtual Link editor that is provided as a tab in the Deployment editor (see Figure 41). The Virtual Link generation is automated since it requires solving a number of complex tasks, including the multi-hop routing through the modelled platform.

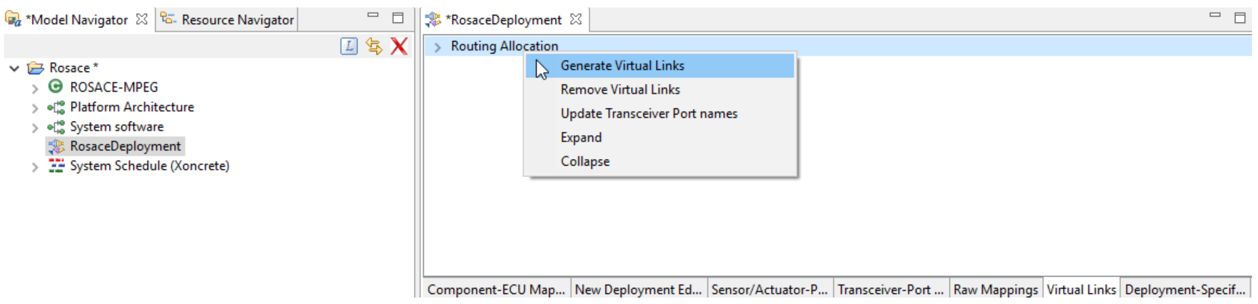
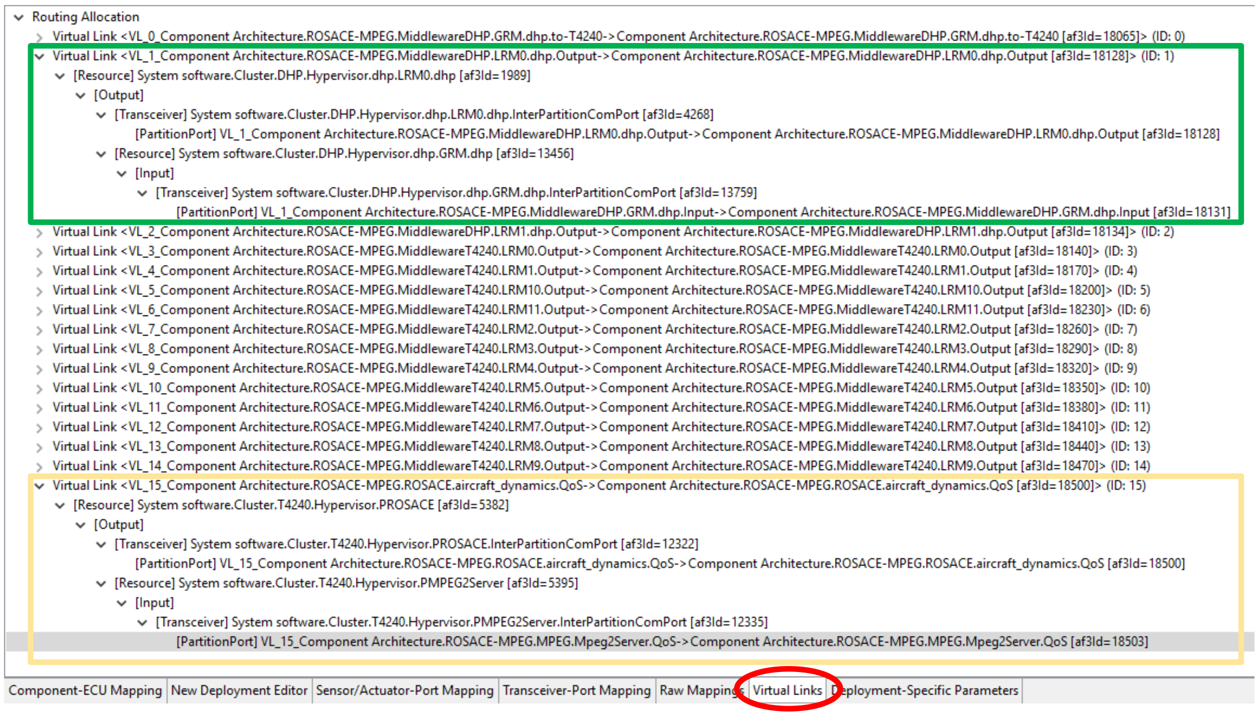


Figure 41: Virtual Link editor.

For the ROSACE example, the majority of Virtual Links (here: 15) are used for the communication between the resource management components that are located on different partitions, e.g., the Virtual Link in the green box in Figure 42). Since Virtual Links are not required if communicating logical components reside on the same partition and the components constituting the ROSACE component are located within one partition, only one Virtual Link is generated for the actual application (yellow box in Figure 42). It provides a Quality of Service feedback for the MPEG component.



```
Routing Allocation
  Virtual Link <VL_0_Component Architecture.ROSACE-MPEG.MiddlewareDHP.GRM.dhp.to-T4240->Component Architecture.ROSACE-MPEG.MiddlewareDHP.GRM.dhp.to-T4240 [af3Id=18065]> (ID: 0)
  Virtual Link <VL_1_Component Architecture.ROSACE-MPEG.MiddlewareDHP.LRM0.dhp.Output->Component Architecture.ROSACE-MPEG.MiddlewareDHP.LRM0.dhp.Output [af3Id=18128]> (ID: 1)
    [Resource] System software.Cluster.DHP.Hypervisor.dhp.LRM0.dhp [af3Id=1989]
      [Output]
        [Transceiver] System software.Cluster.DHP.Hypervisor.dhp.LRM0.dhp.InterPartitionComPort [af3Id=4268]
          [PartitionPort] VL_1_Component Architecture.ROSACE-MPEG.MiddlewareDHP.LRM0.dhp.Output->Component Architecture.ROSACE-MPEG.MiddlewareDHP.LRM0.dhp.Output [af3Id=18128]
        [Resource] System software.Cluster.DHP.Hypervisor.dhp.GRM.dhp [af3Id=13456]
          [Input]
            [Transceiver] System software.Cluster.DHP.Hypervisor.dhp.GRM.dhp.InterPartitionComPort [af3Id=13759]
              [PartitionPort] VL_1_Component Architecture.ROSACE-MPEG.MiddlewareDHP.GRM.dhp.Input->Component Architecture.ROSACE-MPEG.MiddlewareDHP.GRM.dhp.Input [af3Id=18131]
  Virtual Link <VL_2_Component Architecture.ROSACE-MPEG.MiddlewareDHP.LRM1.dhp.Output->Component Architecture.ROSACE-MPEG.MiddlewareDHP.LRM1.dhp.Output [af3Id=18134]> (ID: 2)
  Virtual Link <VL_3_Component Architecture.ROSACE-MPEG.MiddlewareT4240.LRM0.Output->Component Architecture.ROSACE-MPEG.MiddlewareT4240.LRM0.Output [af3Id=18140]> (ID: 3)
  Virtual Link <VL_4_Component Architecture.ROSACE-MPEG.MiddlewareT4240.LRM1.Output->Component Architecture.ROSACE-MPEG.MiddlewareT4240.LRM1.Output [af3Id=18170]> (ID: 4)
  Virtual Link <VL_5_Component Architecture.ROSACE-MPEG.MiddlewareT4240.LRM10.Output->Component Architecture.ROSACE-MPEG.MiddlewareT4240.LRM10.Output [af3Id=18200]> (ID: 5)
  Virtual Link <VL_6_Component Architecture.ROSACE-MPEG.MiddlewareT4240.LRM11.Output->Component Architecture.ROSACE-MPEG.MiddlewareT4240.LRM11.Output [af3Id=18230]> (ID: 6)
  Virtual Link <VL_7_Component Architecture.ROSACE-MPEG.MiddlewareT4240.LRM2.Output->Component Architecture.ROSACE-MPEG.MiddlewareT4240.LRM2.Output [af3Id=18260]> (ID: 7)
  Virtual Link <VL_8_Component Architecture.ROSACE-MPEG.MiddlewareT4240.LRM3.Output->Component Architecture.ROSACE-MPEG.MiddlewareT4240.LRM3.Output [af3Id=18290]> (ID: 8)
  Virtual Link <VL_9_Component Architecture.ROSACE-MPEG.MiddlewareT4240.LRM4.Output->Component Architecture.ROSACE-MPEG.MiddlewareT4240.LRM4.Output [af3Id=18320]> (ID: 9)
  Virtual Link <VL_10_Component Architecture.ROSACE-MPEG.MiddlewareT4240.LRM5.Output->Component Architecture.ROSACE-MPEG.MiddlewareT4240.LRM5.Output [af3Id=18350]> (ID: 10)
  Virtual Link <VL_11_Component Architecture.ROSACE-MPEG.MiddlewareT4240.LRM6.Output->Component Architecture.ROSACE-MPEG.MiddlewareT4240.LRM6.Output [af3Id=18380]> (ID: 11)
  Virtual Link <VL_12_Component Architecture.ROSACE-MPEG.MiddlewareT4240.LRM7.Output->Component Architecture.ROSACE-MPEG.MiddlewareT4240.LRM7.Output [af3Id=18410]> (ID: 12)
  Virtual Link <VL_13_Component Architecture.ROSACE-MPEG.MiddlewareT4240.LRM8.Output->Component Architecture.ROSACE-MPEG.MiddlewareT4240.LRM8.Output [af3Id=18440]> (ID: 13)
  Virtual Link <VL_14_Component Architecture.ROSACE-MPEG.MiddlewareT4240.LRM9.Output->Component Architecture.ROSACE-MPEG.MiddlewareT4240.LRM9.Output [af3Id=18470]> (ID: 14)
  Virtual Link <VL_15_Component Architecture.ROSACE-MPEG.ROSACE.aircraft_dynamics.QoS->Component Architecture.ROSACE-MPEG.ROSACE.aircraft_dynamics.QoS [af3Id=18500]> (ID: 15)
    [Resource] System software.Cluster.T4240.Hypervisor.PROSACE [af3Id=5382]
      [Output]
        [Transceiver] System software.Cluster.T4240.Hypervisor.PROSACE.InterPartitionComPort [af3Id=12322]
          [PartitionPort] VL_15_Component Architecture.ROSACE-MPEG.ROSACE.aircraft_dynamics.QoS->Component Architecture.ROSACE-MPEG.ROSACE.aircraft_dynamics.QoS [af3Id=18500]
        [Resource] System software.Cluster.T4240.Hypervisor.PMPEG2Server [af3Id=5395]
          [Input]
            [Transceiver] System software.Cluster.T4240.Hypervisor.PMPEG2Server.InterPartitionComPort [af3Id=12335]
              [PartitionPort] VL_15_Component Architecture.ROSACE-MPEG.MPEG.Mpeg2Server.QoS->Component Architecture.ROSACE-MPEG.MPEG.Mpeg2Server.QoS [af3Id=18503]
```

Figure 42: Virtual Links of the ROSACE initial Deployment.

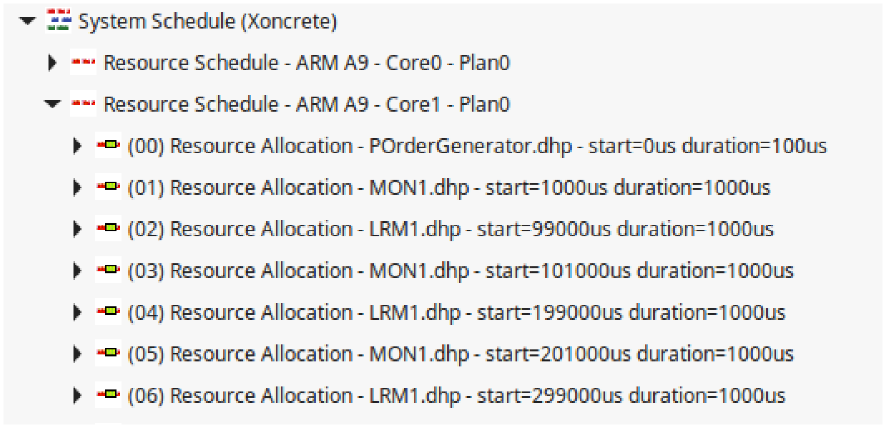
4.4 Configuration of Schedulers

4.4.1 Nominal Task and Partition scheduling

The generation of schedules for partitions and tasks for the nominal mode, corresponds to step 7A of the Tool Chain Use Case 2, see [1]. The goal is to provide a basis from which GRec (Section 4.4.2) generates new scheduling plans to mitigate core failures. As we have seen before, the periods of the resource management tasks are chosen to be equal to the MAF (= smallest common multiple) of application task periods. In this use-case, the MAF is 1s, which is a rather long time interval and leads to complexity problems when using GRec to find new scheduling plans for different core failure scenarios.

To work around this problem, we temporarily change the resource task period to 100ms, before letting Xoncrete generate the nominal scheduling plans. In order to perform this generation with Xoncrete, one needs to follow the steps already described in Section 3.4.2.

An examination of the resulting System Schedule reveals that the first location of the execution of every task is located within the first 100ms, as illustrated with “POrderGenerator” in Figure 43. The period of “OrderGenerator” is 1s. This allows exporting only the first 100ms of the nominal scheduling plan to GRec, reducing this way the complexity of the problem to be solved by GRec. When importing the result of GRec into AF3, the original resource management period of 1s will be restored, see Section 4.4.2.



```
System Schedule (Xoncrete)
    Resource Schedule - ARM A9 - Core0 - Plan0
    Resource Schedule - ARM A9 - Core1 - Plan0
        (00) Resource Allocation - POrderGenerator.dhp - start=0us duration=100us
        (01) Resource Allocation - MON1.dhp - start=1000us duration=1000us
        (02) Resource Allocation - LRM1.dhp - start=99000us duration=1000us
        (03) Resource Allocation - MON1.dhp - start=101000us duration=1000us
        (04) Resource Allocation - LRM1.dhp - start=199000us duration=1000us
        (05) Resource Allocation - MON1.dhp - start=201000us duration=1000us
        (06) Resource Allocation - LRM1.dhp - start=299000us duration=1000us
```

Figure 43 – ROSACE First execution of "OrderGenerator" within 100ms.

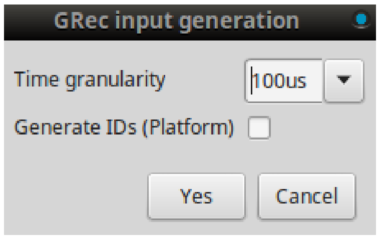
4.4.2 Failure modes

Two failure modes are considered in the building blocks, namely permanent core failure and deadline overrun but only the first one is modelled in the DREAMS tool chain. When a core of a node fails, the local LRM can apply a local reconfiguration and the GRM can apply a global reconfiguration (see D2.2.2[9]). All these local and global reconfiguration graphs are computed by GRec and stored in the meta-model (D4.4.1 [1]).

The generation of a reconfiguration graph for tasks scheduling plans to allow ensuring the continuity of service for critical tasks in case of core failures, corresponds to step 8 of the Tool Chain Use Case 2, see [1].

In order to generate the input file for GRec, the “Generate GRec input” entry from the context menu of the System Schedule created with Xoncrete needs to be executed.

In order to further reduce the size of the solution space to be explored by GRec, a configuration dialog asks for the granularity of time to use (Figure 44). Since in the current use case, all periods and WCETs are multiples of 100us, we chose a granularity of 100us.


GRec input generation

Time granularity: 100us

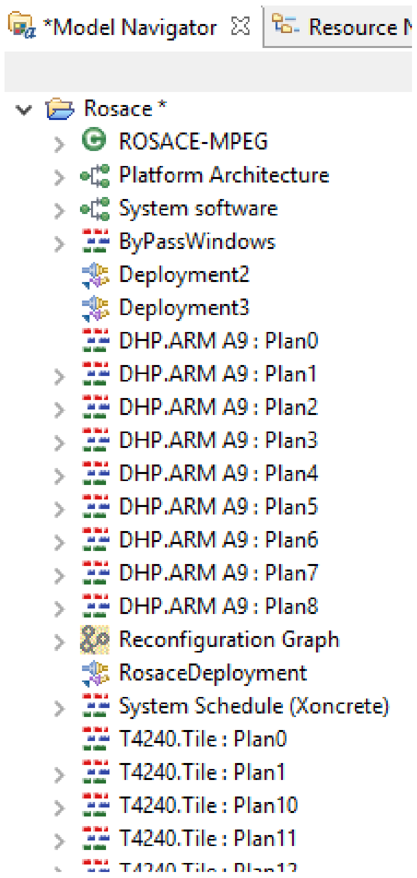
Generate IDs (Platform): ☐

Figure 44 - GRec input generation option.

Remember from Section 4.4.1 that we have reduced the period of the resource management tasks to 100ms. But GRec considers that the period of these tasks should be the MAF. Before importing the output of GRec, we need to set their periods back to 1s, by using the dedicated context menu (see Figure 36, Section 4.2.2).

To import into AF3 the output of GRec, the "Import GRec ouput" entry from the context menu of the System Schedule created with Xoncrete needs to be executed. The following entities are created:

- For each plan defined for the "ARM A9" Tile on the DHP node and 12 core Tile of T4240 node, a separate system schedule is created, see Figure 45.
- A Reconfiguration Graph is created that specifies local (LRM) and global (GRM) scheduling plan changes. Figure 46 shows the transitions of the global reconfiguration graph.
- A set of additional Deployments that correspond to the combinations of the scheduling plans (one for each Tile) which may occur when the LRMs and GRM follow the local and global reconfigurations defined the Reconfiguration Graph.


*Model Navigator
Resource N

- Rosace *
  - ROSACE-MPEG
  - Platform Architecture
  - System software
  - ByPassWindows
  - Deployment2
  - Deployment3
  - DHP.ARM A9 : Plan0
  - DHP.ARM A9 : Plan1
  - DHP.ARM A9 : Plan2
  - DHP.ARM A9 : Plan3
  - DHP.ARM A9 : Plan4
  - DHP.ARM A9 : Plan5
  - DHP.ARM A9 : Plan6
  - DHP.ARM A9 : Plan7
  - DHP.ARM A9 : Plan8
  - Reconfiguration Graph
  - RosaceDeployment
  - System Schedule (Xoncrete)
  - T4240.Tile : Plan0
  - T4240.Tile : Plan1
  - T4240.Tile : Plan10
  - T4240.Tile : Plan11
  - T4240.Tile : Plan12

Figure 45 - ROSACE: Deployments, System Schedules and Reconfiguration Graph created by the GRec output importer.

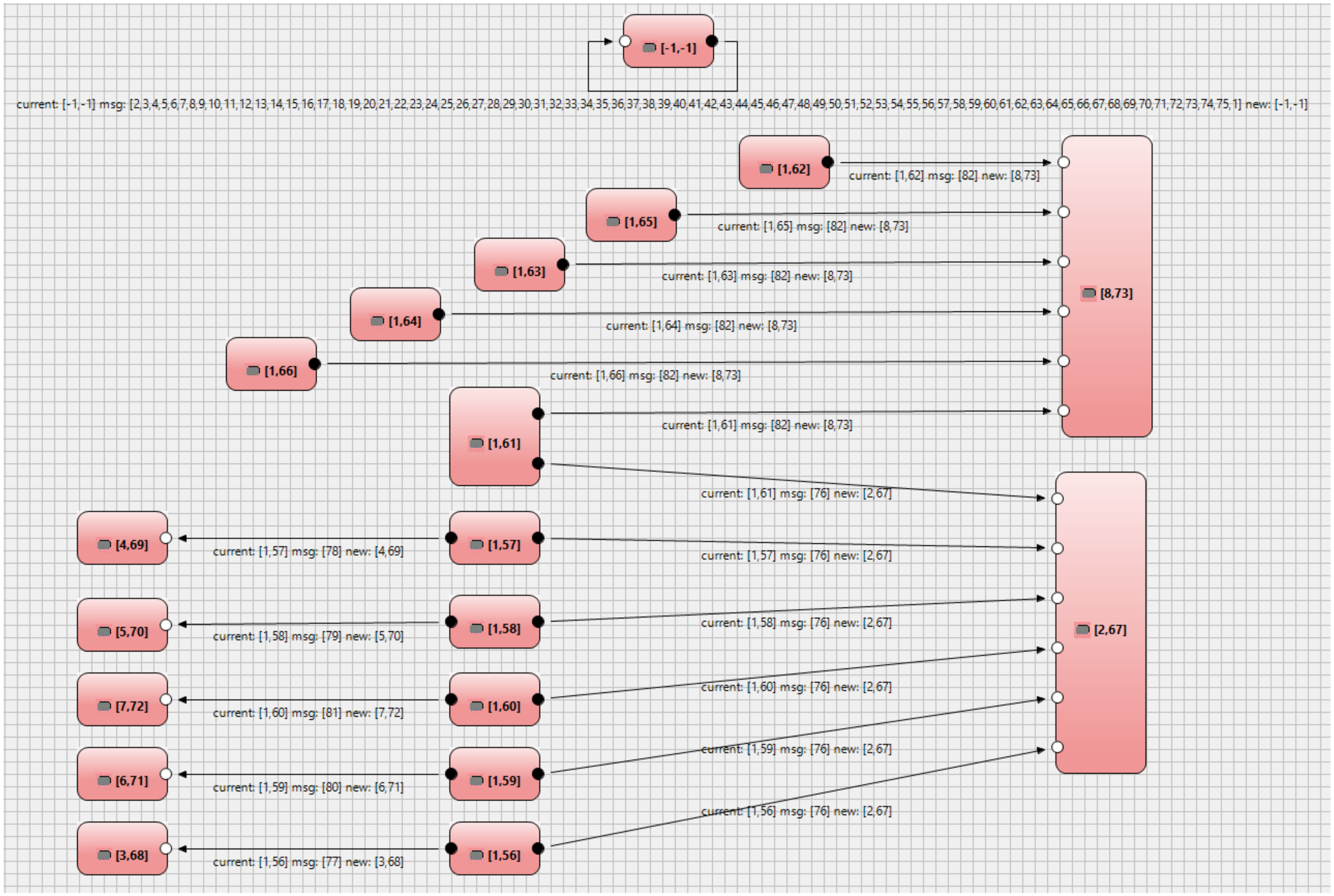
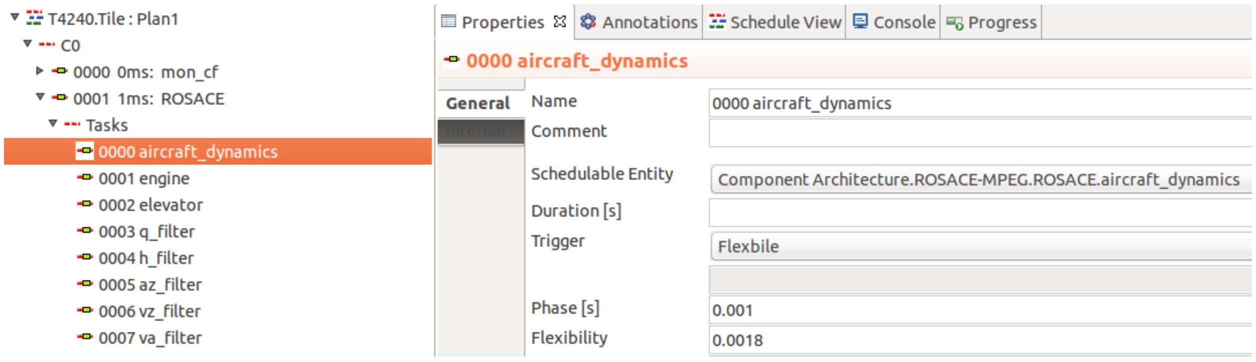

[-1,-1]
current: [-1,-1] msg: [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,1] new: [-1,-1]
[1,62]
current: [1,62] msg: [82] new: [8,73]
[1,65]
current: [1,65] msg: [82] new: [8,73]
[1,63]
current: [1,63] msg: [82] new: [8,73]
[8,73]
[1,64]
current: [1,64] msg: [82] new: [8,73]
[1,66]
current: [1,66] msg: [82] new: [8,73]
[1,61]
current: [1,61] msg: [82] new: [8,73]
current: [1,61] msg: [76] new: [2,67]
[4,69]
current: [1,57] msg: [78] new: [4,69]
[1,57]
current: [1,57] msg: [76] new: [2,67]
[5,70]
current: [1,58] msg: [79] new: [5,70]
[1,58]
current: [1,58] msg: [76] new: [2,67]
[2,67]
[7,72]
current: [1,60] msg: [81] new: [7,72]
[1,60]
current: [1,60] msg: [76] new: [2,67]
[6,71]
current: [1,59] msg: [80] new: [6,71]
[1,59]
current: [1,59] msg: [76] new: [2,67]
[3,68]
current: [1,56] msg: [77] new: [3,68]
[1,56]
current: [1,56] msg: [76] new: [2,67]

Figure 46 - ROSAE: transitions of the global reconfiguration graph

4.4.3 On-line flexibility

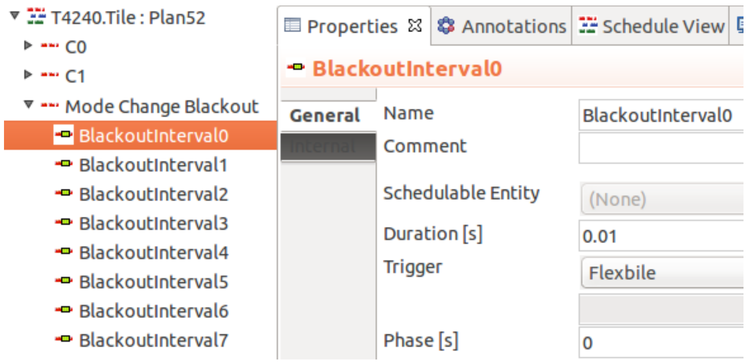
The estimation of online flexibility coefficient corresponds to step 9 of the Tool Chain Use Case 2, see D4.4.1 [1]. The flexibility parameter for a job defines the maximum delay which can be tolerated by the job without missing any deadline in the system. This information can be utilized by the partition LRS to admit aperiodic tasks online with the least response-time. The flexibility parameter for each job in the plan is calculated as defined in [12]. In order to generate flexibility parameter from the model, the input file for MCOSF needs to be generated using “Generate MCOSF input” entry from the context menu of the ‘Reconfiguration Graph’ created with GRec output import. The tool MCOSF is executed with the generated MCOSF input file as mentioned in Section 4.6.5 of D4.4.1 [1]. Once the MCOSF exits without any error, the MCOSF output file can be imported back in AF3 model by invoking “Import MCOSF output’ entry from the context menu of the ‘Reconfiguration Graph’ created with GRec output import. The MCOSF output import modifies the task triggers of each plan from Periodic to Flexible and defines the flexibility parameter as shown in Figure 47 below.



Name	0000 aircraft_dynamics
Comment	
Schedulable Entity	Component Architecture.ROSACE-MPEG.ROSACE.aircraft_dynamics
Duration [s]	
Trigger	Flexbile
Phase [s]	0.001
Flexibility	0.0018

Figure 47 – ROSACE: Flexibility parameter generated using MCOSF.

Other than the flexibility parameter, MCOSF also generates blackout intervals and transition modes as defined in Section 10 of D4.1.3 [6]. The blackout intervals are calculated for each source of the mode transition defined by the local reconfiguration graphs of each node. The blackout intervals can be utilized by the hypervisor to see if an immediate mode switch at current time will lead to a deadline miss, i.e. not to switch immediately if the plan at current time specifies a blackout interval. An example of the blackout interval for the ROSACE case study is shown in Figure 48 (Note that the trigger for a blackout interval is not applicable/don’t-care as it does not trigger anything. Instead, it merely stops from triggering a mode change).



Name	BlackoutInterval0
Comment	
Schedulable Entity	(None)
Duration [s]	0.01
Trigger	Flexbile
Phase [s]	0

Figure 48 - ROSACE: Blackout intervals generated using MCOSF.

4.4.4 On-chip Scheduling of TT Virtual Links

After the creation of additional Deployments through the import of the Reconfiguration Graph generated by GRec (Section 4.4.2), all different communication scenarios implied by the reconfigurations can be derived. Thus the scheduling of the on-chip communication can be configured, which corresponds to step 11 of Tool Chain Use Case 2, see Section 2.

As can be seen in the upper part of Figure 37, only the processor Tile “ARM9” of the DHP is used in this use case. Thus, all communication from and to that Tile goes actually over the off-chip network. The communication between the processor Tile and the on-chip/off-chip gateway is allowed within so-called bypass windows which are opened and closed by the NI LRS according to a static schedule (see D2.1.3 [13]). The sending and receiving through the bypass window is managed by the DRAL and not by the NI LRS. It implies that the generation of transmission offset with RTaW-Timing cannot be used here.

But a bypass window needs to be defined and translated into the configuration files for the on-chip communication (Section 4.5.2), since the NI LRS is responsible for opening (and closing) the bypass window. Otherwise the DRAL would not be able to communicate with the on-chip/off-chip gateway.

The following virtual links are (potentially) sent or received over the off-chip network:

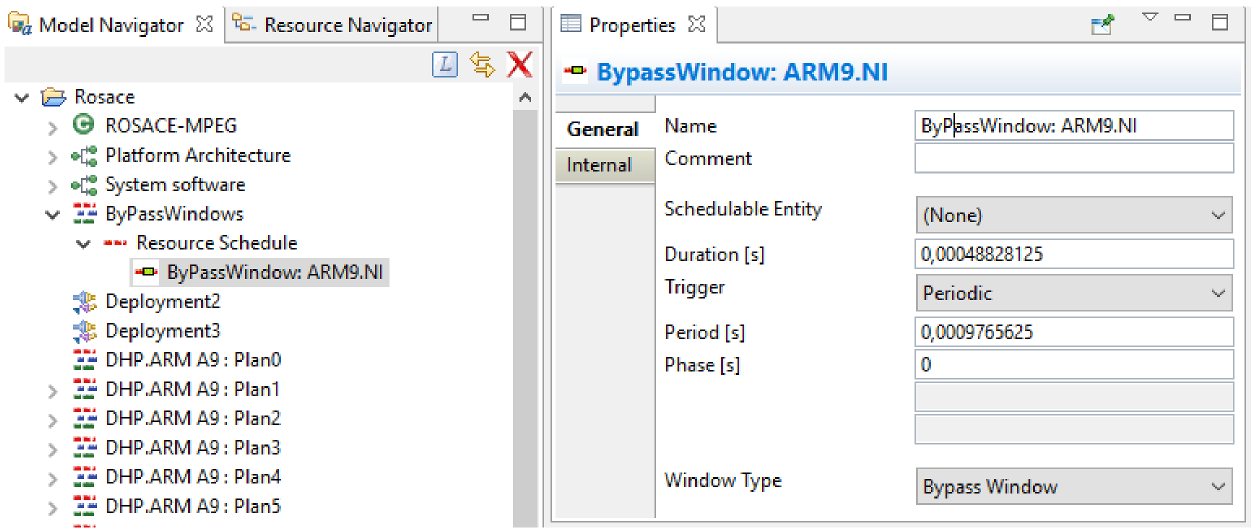
- GRM order message sent every 100ms
- 12 T4240 LRM status message sent every 100ms
- 1 message sent every 1s by OrderGenerator to ROSACE
- 1 message sent every 1s by ROSACE to MPEG2server

The required bandwidth is low with respect to the throughput of the NoC at 100 Mhz, but the time between two consecutive bypass window instances must be limited, in order to limit waiting times at the entry of the NoC. Remember that periods, length and offset must be (negative) power of 1s (see D2.1.3 [13]).

We propose to use one bypass window with the following characteristics:

- open time = 0s
- length = 2^{-11} s = 0,00048828125 s ≈ 488,28µs
- period = 2^{-10} s = 0,0009765625 s ≈ 976,56µs

If the bandwidth is sufficient, then the resulting NoC traversal time will be lower than 1ms. In Section 4.4.6 we will verify this choice by computing a rough upper bound on the NoC traversal.



Name	ByPassWindow: ARM9.NI
Comment	
Schedulable Entity	(None)
Duration [s]	0,00048828125
Trigger	Periodic
Period [s]	0,0009765625
Phase [s]	0
Window Type	Bypass Window

Figure 49 - ROSACE: Bypass Window

The bypass window must be defined in a dedicated System Schedule based on the nominal Deployment "Rosace Deployment" (Figure 49) and specified in the configuration dialog of the "On-chip TT Schedule Generation" menu entry which must be executed from the context menu of the Reconfiguration Graph. The configuration dialog asks for the System Schedule with the bypass window and creates for each Deployment referenced by the Reconfiguration Graph a System Schedule with the specified bypass window (see Figure 50).

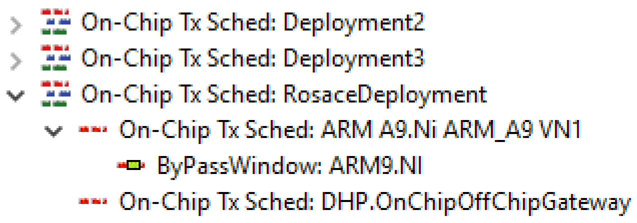

On-Chip Tx Sched: Deployment2
On-Chip Tx Sched: Deployment3
On-Chip Tx Sched: RosaceDeployment
On-Chip Tx Sched: ARM A9.Ni ARM_A9 VN1
ByPassWindow: ARM9.NI
On-Chip Tx Sched: DHP.OnChipOffChipGateway

Figure 50 – ROSACE: System Schedules for the on-chip communication

4.4.5 Off-chip Scheduling of TT Virtual Links

After the creation of additional Deployments through the import of the Reconfiguration Graph generated by GRec (Section 4.4.2), all different possible communication scenarios implied by the reconfigurations are known and thus the scheduling of the off-chip communication can be defined, which corresponds to step 10 of Tool Chain Use Case 2, see Section 2.

To be able to use the dedicated tool TTE-Plan, the "Generate TTEthernet Network Description" entry from the context menu of the "Reconfiguration Graph" needs to be executed. This produces a ".network_description" file, which is the input for TTE-Plan. Notice that also TTE-Build needs to be executed in order to generate the C header files that contain the port ids, which are needed in the platform configuration file (PCF), see Section 4.5.1. In order to import the defined schedules and port ids, the "Import TTEthernet Scheduling" entry from the context menu of the Reconfiguration Graph" needs to be executed.

The result of the import is the creation of dedicated System Schedules, one for each of the possible Deployments implied by the reconfiguration (Figure 51).

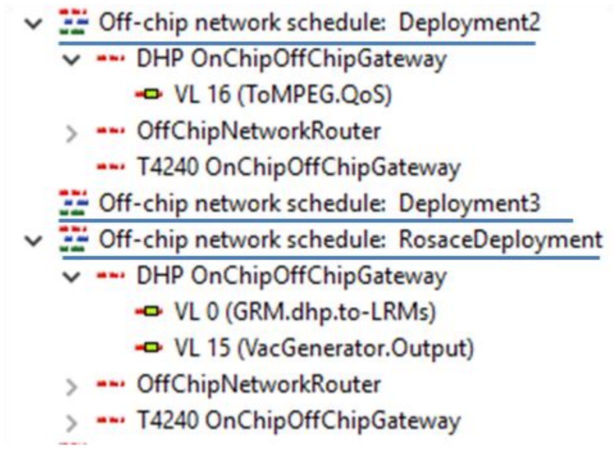
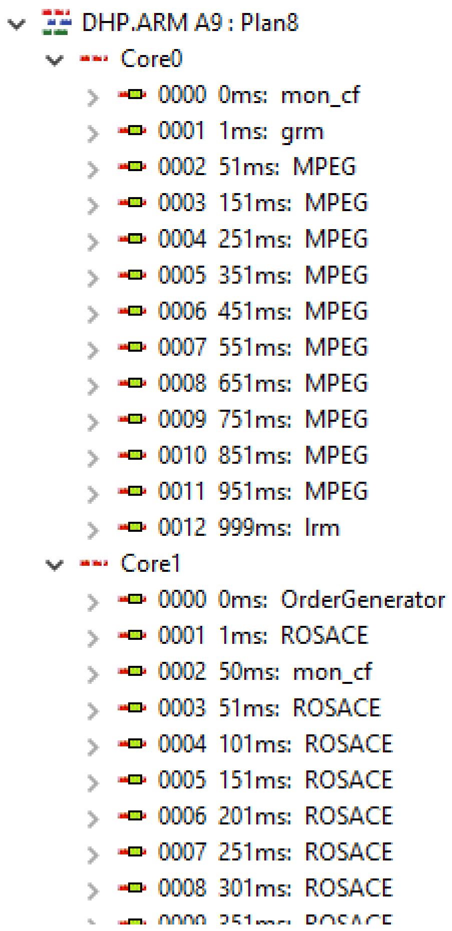

Off-chip network schedule: Deployment2
DHP OnChipOffChipGateway
VL 16 (ToMPEG.QoS)
OffChipNetworkRouter
T4240 OnChipOffChipGateway
Off-chip network schedule: Deployment3
Off-chip network schedule: RosaceDeployment
DHP OnChipOffChipGateway
VL 0 (GRM.dhp.to-LRMs)
VL 15 (VacGenerator.Output)
OffChipNetworkRouter
T4240 OnChipOffChipGateway

Figure 51 - ROSACE: System Schedules for the off-chip communication.

Since the deployment of resource management tasks does never change, the scheduling of their virtual links is only defined in the nominal System Schedule. The Deployment specific schedules of the virtual links between applications are defined in the corresponding System Schedule. As can be seen, there is no off-chip communication between applications in “Deployment3”. This is because all applications are executed on the same Tile. This is, for instance, the case with the scheduling plan 8 of the DHP: ROSACE, Mpeg2Server and OrderGenerator are executed on the two cores of the DHP (see Figure 52).

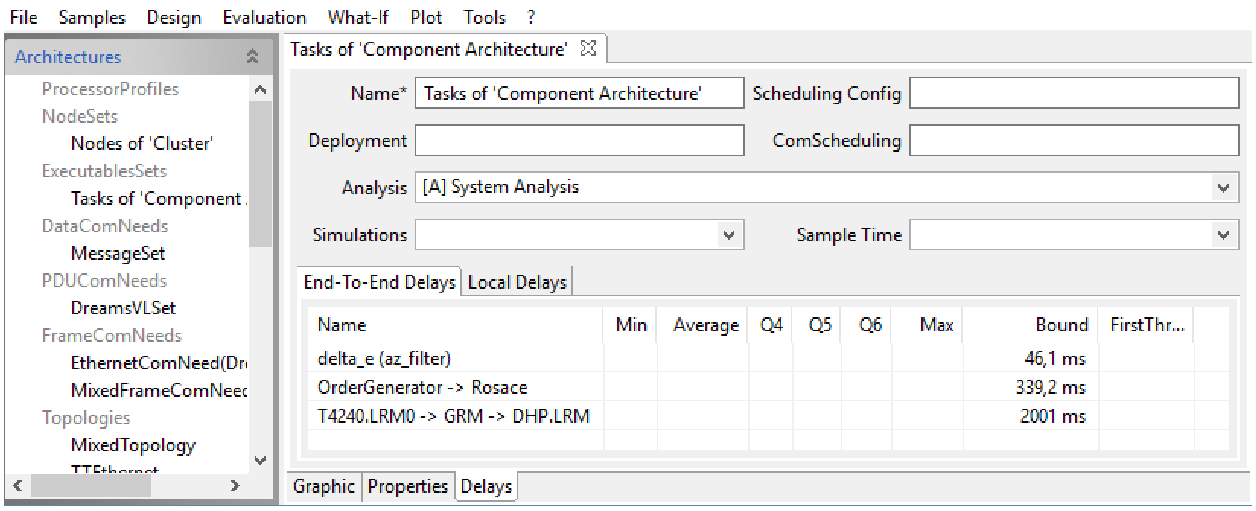


```
DHP.ARM A9 : Plan8
  Core0
    0000 0ms: mon_cf
    0001 1ms: grm
    0002 51ms: MPEG
    0003 151ms: MPEG
    0004 251ms: MPEG
    0005 351ms: MPEG
    0006 451ms: MPEG
    0007 551ms: MPEG
    0008 651ms: MPEG
    0009 751ms: MPEG
    0010 851ms: MPEG
    0011 951ms: MPEG
    0012 999ms: lrm
  Core1
    0000 0ms: OrderGenerator
    0001 1ms: ROSACE
    0002 50ms: mon_cf
    0003 51ms: ROSACE
    0004 101ms: ROSACE
    0005 151ms: ROSACE
    0006 201ms: ROSACE
    0007 251ms: ROSACE
    0008 301ms: ROSACE
    0009 351ms: ROSACE
```

Figure 52 - ROSACE: Scheduling plan 8 of the DHP, execution all three applications.

4.4.6 Timing Analysis

Performing worst case analysis of end-to-end delays, in order to verify that latency constraints on the timing chains are satisfied, corresponds to step 12 of the Tool Chain Use Case 2, see [1]. In order to execute the algorithm implemented in RTaW-Timing, the “Timing Evaluation” entry of the context menu of the “Reconfiguration Graph” must be selected, since it contains the references to all possible SystemScheules. As a result, the complete description of designed system is exported to RTaW-Timing, which runs the worst-case analysis and displays the results in its GUI, as shown in Figure 53.

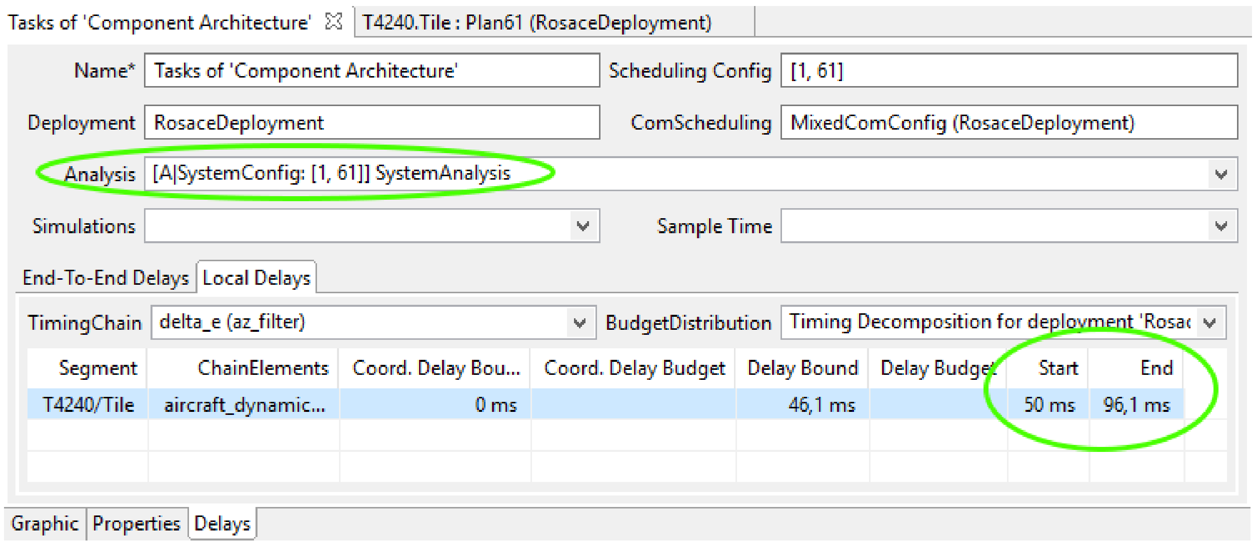


Name* Tasks of 'Component Architecture'
Scheduling Config
Deployment
ComScheduling
Analysis [A] System Analysis
Simulations
Sample Time

Name	Min	Average	Q4	Q5	Q6	Max	Bound	FirstThr...
delta_e (az_filter)							46,1 ms	
OrderGenerator -> Rosace							339,2 ms	
T4240.LRM0 -> GRM -> DHP.LRM							2001 ms	

Figure 53 - ROSACE: WP7: Bounds on worst-case delays of timing chains (RTaW-Timing).

The first line “delta_e(az_filter)” corresponds to the first timing chain described in Section 4.2.2. The computed upper bound on the worst-case delay is 46.1 ms. Remember that in case of reconfiguration (Section 4.4.2) several scheduling plans are defined for each Tile. By clicking on the line and selecting the "Show local delays bound of maximal end-to-end bound" entry from the context menu, the tool shows more details in the “Local Delays” tab, see Figure 54.


Tasks of 'Component Architecture'
T4240.Tile : Plan61 (RosaceDeployment)
End-To-End Delays
Local Delays
Graphic
Properties
Delays

Name* Tasks of 'Component Architecture'
Scheduling Config [1, 61]
Deployment RosaceDeployment
ComScheduling MixedComConfig (RosaceDeployment)
Analysis [A|SystemConfig: [1, 61]] SystemAnalysis
Simulations
Sample Time
TimingChain delta_e (az_filter)
BudgetDistribution Timing Decomposition for deployment 'Rosac

Segment	ChainElements	Coord. Delay Bou...	Coord. Delay Budget	Delay Bound	Delay Budge	Start	End
T4240/Tile	aircraft_dynamic...	0 ms		46,1 ms		50 ms	96,1 ms

Figure 54 - ROSACE: details about the bound on the timing chain “delta_e(az_filter)”.

As shown by the “Segment” column in Figure 54, tasks of the chain are all executed on the T4240. The name of the “Analysis” has been changed by the tool indicated to that of a scenario where the worst case delay can be observed: the scenario is based on plan1 for the DHP and plan 61 for the T4240.

Furthermore, the execution of the first task of the chain starts at 50ms (“Start” column) and the execution of the last task ends at 96.1 ms (“End” column). Figure 55 shows the corresponding Gantt chart. It can be seen that the delay is due in part to different periods and non-optimal execution orders of the involved task: the output of “aircraft_dynamics” may have to wait 10ms before being read by the next instance of “vz_filter”. On the other side, the distance between two executions of the task “vz_control” may be larger than its nominal period of 20ms in the scheduling table of plan 61.

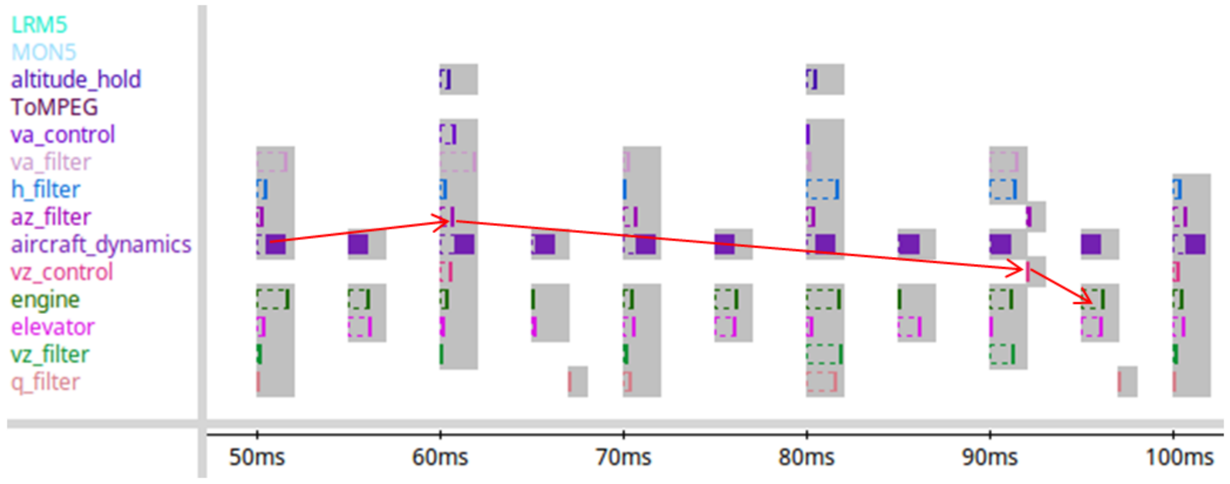
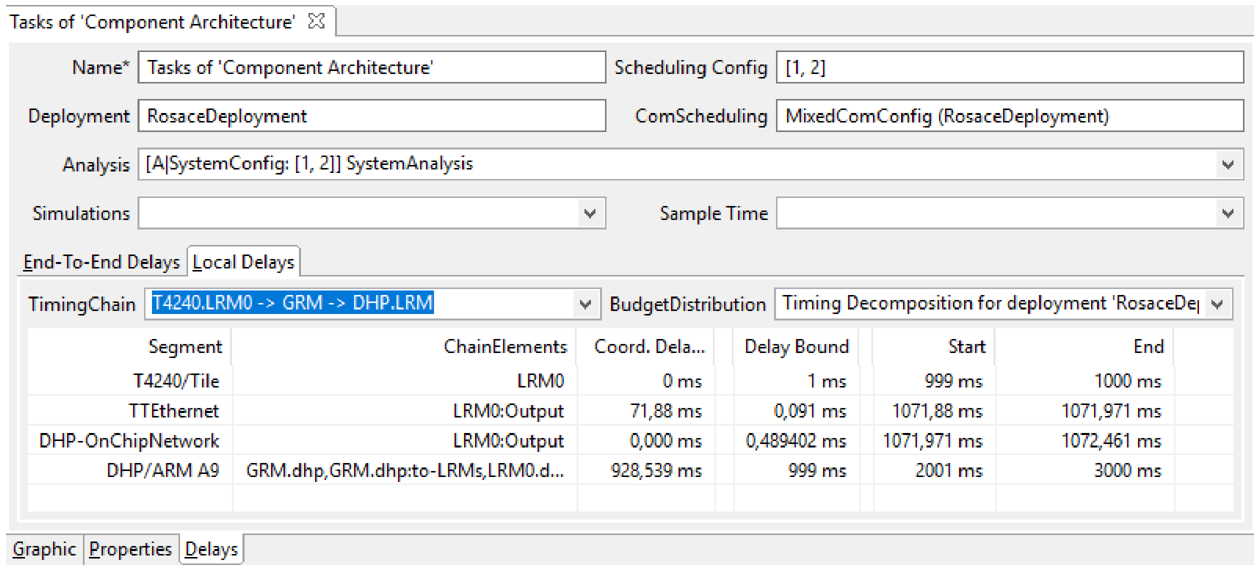


Figure 55 - ROSACE: Gantt chart of worst delay of the timing chain “delta_e(az_filter)” on the T4240.

Figure 56 shows the details of the worst case delay of the timing chain “T4240.LRM0 -> GRM -> DHP.LRM”. Recall that the delay spans from the execution start of LRM0 on T4240, which potentially detects a core failure, until the next execution end of LRM0 on the DHP that would receive a corresponding reconfiguration command from the GRM.

Have a look at the last line of the table, which corresponds to the task execution segment on the DHP. Figure 57 shows the corresponding scheduling table, which has a period of 1s: the GRM is executed almost at the beginning and the LRM at the end. Recall furthermore that the schedule tables of the nodes are based on the same global time (DREAMS platform service). Thus, when the message arrives at t=1072,461 ms, i.e. at 72,461 ms after the start of the schedule table, the GRM has already finished to execute. Therefore, a “coordination delay” of 928,539 ms occurs until the execution of the next instance of the GRM. Since furthermore the LRM is executed at the end of the table, the result is a delay of 999 ms from the start of the GRM until the end of the LRM.



Name*	Tasks of 'Component Architecture'	Scheduling Config	[1, 2]
Deployment	RosaceDeployment	ComScheduling	MixedComConfig (RosaceDeployment)
Analysis	[A|SystemConfig: [1, 2]] SystemAnalysis		
Simulations		Sample Time	
TimingChain	T4240.LRM0 -> GRM -> DHP.LRM	BudgetDistribution	Timing Decomposition for deployment 'RosaceDe

Segment	ChainElements	Coord. Dela...	Delay Bound	Start	End
T4240/Tile	LRM0	0 ms	1 ms	999 ms	1000 ms
TTEthernet	LRM0:Output	71,88 ms	0,091 ms	1071,88 ms	1071,971 ms
DHP-OnChipNetwork	LRM0:Output	0,000 ms	0,489402 ms	1071,971 ms	1072,461 ms
DHP/ARM A9	GRM.dhp,GRM.dhp:to-LRMs,LRM0.d...	928,539 ms	999 ms	2001 ms	3000 ms

Figure 56 - ROSACE: Details about the bound on the timing chain “T4240.LRM0 -> GRM -> DHP.LRM”

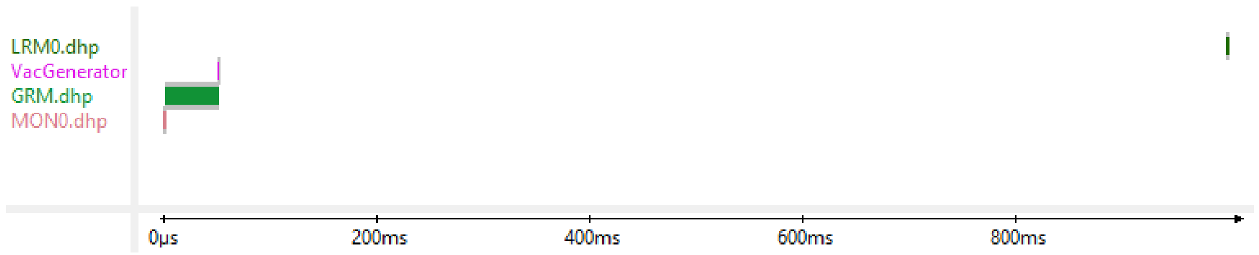


Figure 57 - ROSACE: Scheduling table of core 0 of the DHP node in plan 2.

4.5 Platform Building Block Configuration File Generation

4.5.1 DREAMS Resource Management Configuration files

The platform configuration file (PCF) is generated by invoking the “Generate DRMS Configuration files” entry form the context menu of the “Reconfiguration Graph”. It generates for each Tile a configuration file in YAML format as defined in [7] which contains

- the declaration of applications and their partitions
- the channels for the inter partition communication
- the declaration of off-chip communication ports with their parameters (see Figure 58)
- a set of partition and task scheduling plans (see Figure 58)
- the local reconfiguration table
- the global reconfiguration table (see Figure 59)
- the remapping of DRAL ports to partition of on-chip/off-chip communication ports depending on the scheduling plan

```
…
hw_desc:
  num_cores: 2
  devices:
    uart:
    - {name: Uart, id: 1, baud_rate: 115200}
    tte:
    - name: TTEthernet_1
      ports:
      - {name: OrderGenerator_VacGenerator_Output_RosaceDeployment,
        type: TT, portDirection: source, tte_port_id: 9, tte_port_ap: 3}
      - {name: ROSACE_ToMPEG_QoS_Deployment2, type: TT,
         portDirection: source, tte_port_id: 8, tte_port_ap: 1}
  processor_table:
  - id: 0
    freq: 400000000
    plan:
    - id: 0
      major_frame: 1000
      slots:
      - {id: 0, start: 0, duration: 100, part: OrderGenerator}
      - {id: 1, start: 100, duration: 100, dlrm: lrm}
      - {id: 2, start: 100, duration: 100, dlrm: grm}
      - {id: 3, start: 100, duration: 100, dlrm: mon_cf}
    - id: 1
      major_frame: 1000
      slots:
```

```
      - {id: 0, start: 0, duration: 1, dlrm: mon_cf}
      - {id: 1, start: 1, duration: 50, dlrm: grm}
      - {id: 2, start: 51, duration: 1, part: OrderGenerator}
      - {id: 3, start: 999, duration: 1, dlrm: lrm}
…
```

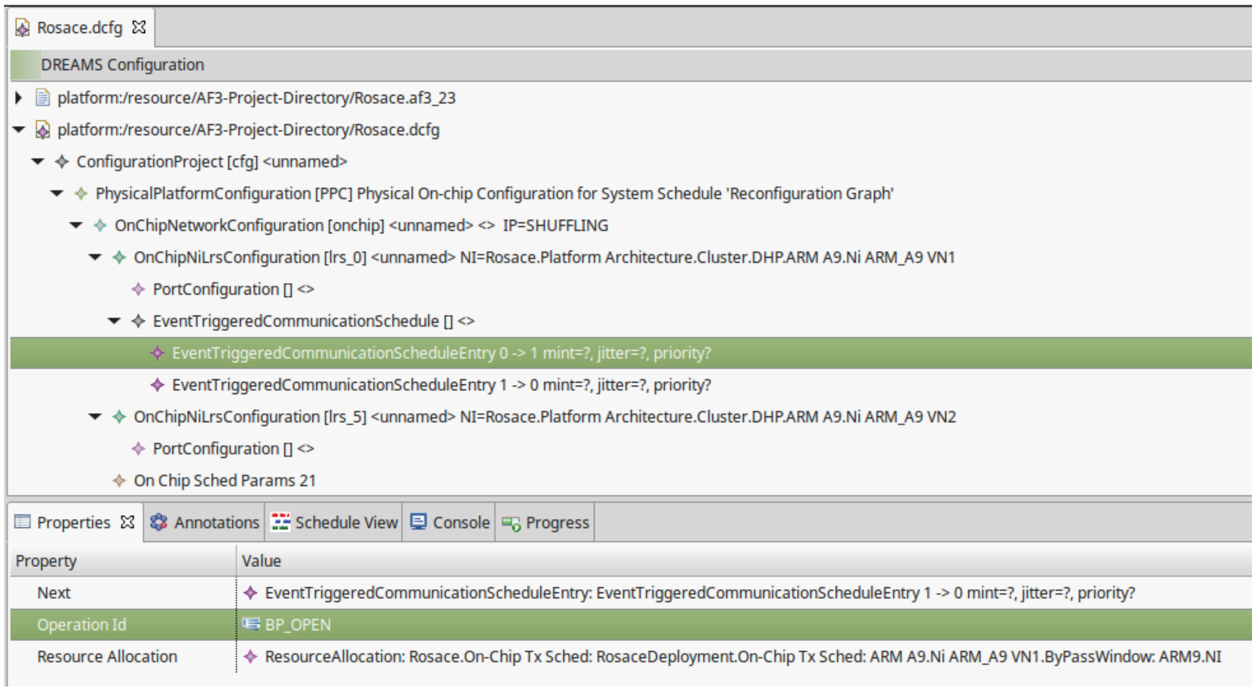
Figure 58 – ROSACE: Snippet1 of the platform configuration file.

```
….
  global_reconfiguration_table:
    - { msg: [2,3,4,5,6,7,8,9], new: [-1,-1], current_configuration: [-1,-1] }
    - { msg:
[10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,3
8,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,
67,68,69,70,71,72,73,74,75], new: [-1,-1], current_configuration: [-1,-1] }
    - { msg: [76], new: [2,67], current_configuration: [1,56] }
    - { msg: [77], new: [3,68], current_configuration: [1,56] }
    - { msg: [76], new: [2,67], current_configuration: [1,57] }
    - { msg: [78], new: [4,69], current_configuration: [1,57] }
    - { msg: [76], new: [2,67], current_configuration: [1,58] }
    - { msg: [79], new: [5,70], current_configuration: [1,58] }
    - { msg: [76], new: [2,67], current_configuration: [1,59] }
    - { msg: [80], new: [6,71], current_configuration: [1,59] }
    - { msg: [76], new: [2,67], current_configuration: [1,60] }
    - { msg: [81], new: [7,72], current_configuration: [1,60] }
    - { msg: [76], new: [2,67], current_configuration: [1,61] }
    - { msg: [82], new: [8,73], current_configuration: [1,61] }
    - { msg: [82], new: [8,73], current_configuration: [1,62] }
    - { msg: [82], new: [8,73], current_configuration: [1,63] }
    - { msg: [82], new: [8,73], current_configuration: [1,64] }
    - { msg: [82], new: [8,73], current_configuration: [1,65] }
    - { msg: [82], new: [8,73], current_configuration: [1,66] }
    - { msg: [1], new: [-1,-1], current_configuration: [-1,-1] }
….
```

Figure 59 – ROSACE: Snippet2 of the platform configuration file.

4.5.2 On-chip network communication configuration files

As described in [1], the generation of the configuration files of the on-chip communication is a two step process, where first a configuration model is generated, by invoking the "Generate 'Physical On-Chip Communication' Configuration Model" entry from the context menu of the Reconfiguration Graph. Figure 60 shows the resulting configuration model. Second, the actual textual configuration files (Figure 61) are generated with the help of the “Run configuration generation framework...” from the context menu of the ROSACE model.



Rosace.dcfg

DREAMS Configuration

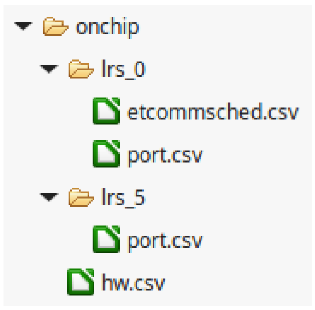
- platform:/resource/AF3-Project-Directory/Rosace.af3_23
- platform:/resource/AF3-Project-Directory/Rosace.dcfg
  - ConfigurationProject [cfg] <unnamed>
    - PhysicalPlatformConfiguration [PPC] Physical On-chip Configuration for System Schedule 'Reconfiguration Graph'
      - OnChipNetworkConfiguration [onchip] <unnamed> <> IP=SHUFFLING
        - OnChipNiLrsConfiguration [lrs_0] <unnamed> NI=Rosace.Platform Architecture.Cluster.DHP.ARM A9.Ni ARM_A9 VN1
          - PortConfiguration [] <>
          - EventTriggeredCommunicationSchedule [] <>
            - EventTriggeredCommunicationScheduleEntry 0 -> 1 mint=?, jitter=?, priority?
            - EventTriggeredCommunicationScheduleEntry 1 -> 0 mint=?, jitter=?, priority?
        - OnChipNiLrsConfiguration [lrs_5] <unnamed> NI=Rosace.Platform Architecture.Cluster.DHP.ARM A9.Ni ARM_A9 VN2
          - PortConfiguration [] <>
        - On Chip Sched Params 21

Properties | Annotations | Schedule View | Console | Progress

Property	Value
Next	EventTriggeredCommunicationScheduleEntry: EventTriggeredCommunicationScheduleEntry 1 -> 0 mint=?, jitter=?, priority?
Operation Id	BP_OPEN
Resource Allocation	ResourceAllocation: Rosace.On-Chip Tx Sched: RosaceDeployment.On-Chip Tx Sched: ARM A9.Ni ARM_A9 VN1.ByPassWindow: ARM9.NI

Figure 60 - ROSACE: Configuration model of the on-chip scheduling (AF3 Resource Navigator).

Recall that in this use case, there are only two network interfaces on the NoC and thus only two LRS configurations are generated. Furthermore, all virtual links go through the bypass window and thus no PortConfigurations are needed, only the scheduling of the bypass window, which is defined by the "EventTriggeredCommunicationSchedule" entity and stored in the "etcommsched.csv" configuration file.



```
onchip
  lrs_0
    etcommsched.csv
    port.csv
  lrs_5
    port.csv
  hw.csv
```

Figure 61 - ROSACE: Generated configuration files of the on-chip scheduling

4.5.3 Off-chip network communication configuration files

The parameters of the time triggered scheduling for the off-chip communication have been imported into AF3 from the off-chip network configuration files generated with TTE-Plan and TTE-Build in step 10 (Section 4.4.5). Since these configuration files cannot be generated from a System Schedule defined in AF3, we (have to) use the files created in step 10.

5 Application to the Healthcare Demonstrator

The final design of the healthcare demonstrator is described in D8.3.1 [14]. In the following sections we describe the minimal system model needed in order to allow the configuration of the off-chip communication through the tool chain.

5.1 Applications and Resources

5.1.1 Logical Architecture

First, the application tasks and their data exchange must be modelled. There are two couples of tasks, one related to the ECG data (critical) and one related to video streaming (non critical), see Figure 62.

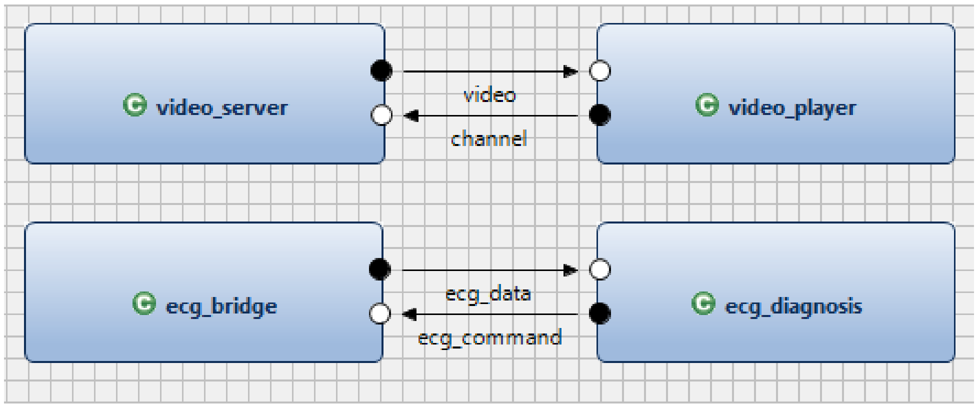
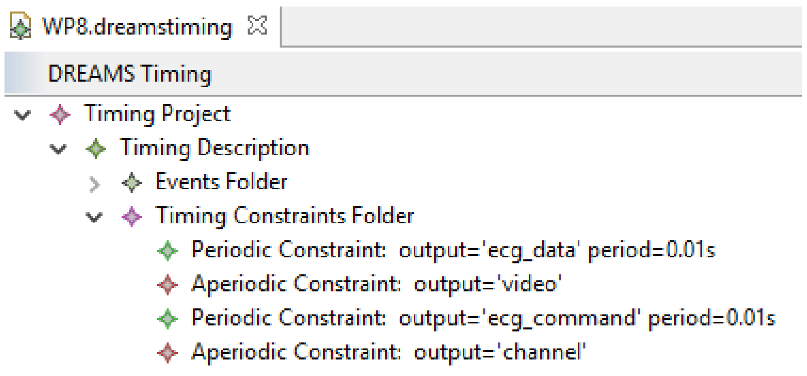


Figure 62 - WP8: application level tasks with exchange of data.

5.1.2 Timing Requirements

Since we only consider off-chip communication, we associate timing constraints directly with the sending ports, see Figure 63.

For the critical ECG communication, the “Time Triggered” traffic class shall be used The criticalwith a period of 10ms. This is specified by attaching an appropriate PeriodicConstraints to each of the concerned sender ports. . For the non critical communication between video server and player, the “Best Effort” traffic class shall be used. This is specified by attaching anAperiodicConstraint to each of the concerned sender ports..


WP8.dreamstiming

DREAMS Timing

- Timing Project
  - Timing Description
    - Events Folder
    - Timing Constraints Folder
      - Periodic Constraint: output='ecg_data' period=0.01s
      - Aperiodic Constraint: output='video'
      - Periodic Constraint: output='ecg_command' period=0.01s
      - Aperiodic Constraint: output='channel'

Figure 63 - WP8: timing constraint for the off-chip communication

5.1.3 Platform Architecture

To be able to configure the off-chip communication, the topology of the off-chip network with the connected nodes needs to be described, see Figure 64.

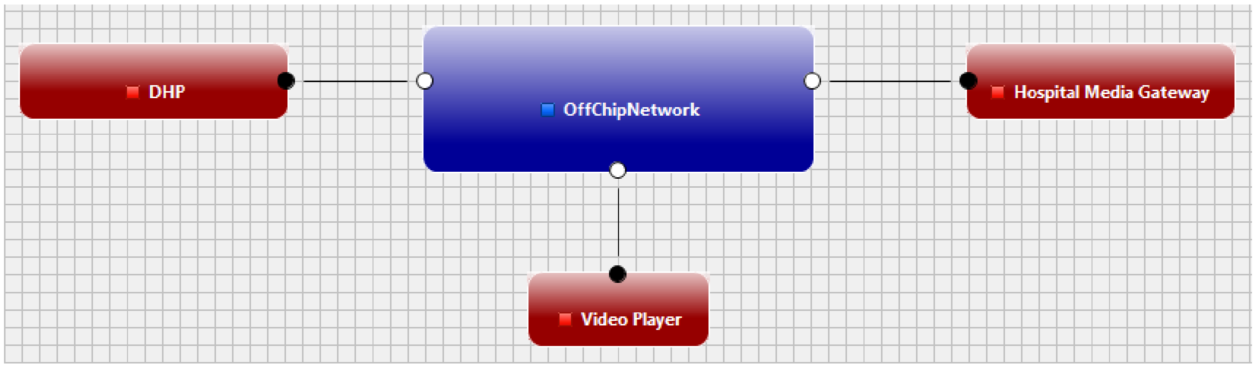


Figure 64 - WP8: off-chip network topology.

The model of a Node connected to the off-chip network must contain at least one processor Tile, containing at least one core to which partition may be mapped, see Section 5.2.1. Figure 65 shows the minimal model used for “Hospital Media Gateway”

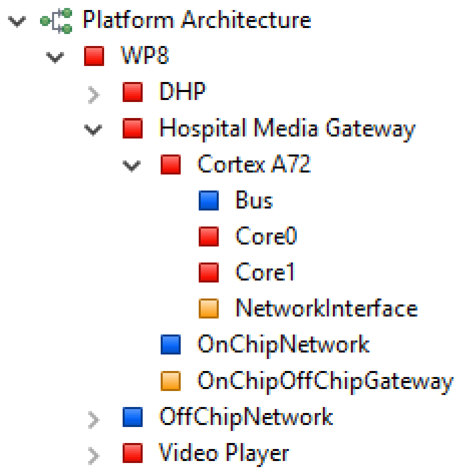

- Platform Architecture
  - WP8
    - DHP
    - Hospital Media Gateway
      - Cortex A72
        - Bus
        - Core0
        - Core1
        - NetworkInterface
      - OnChipNetwork
      - OnChipOffChipGateway
    - OffChipNetwork
    - Video Player

Figure 65 - WP8: Platform Architecture model.

5.2 Deployment

5.2.1 System Software

The deployment of the task components to Tiles determines which communications are crossing the off-chip network. To be able to deploy task components, a minimal System Software entity needs to be defined, consisting of a Hypervisor for each Tile and Partitions to which the tasks can be deployed. Figure 66 shows the details for the “Hospital Media Gateway”, with the hypervisor on the Cortex A72 Tile, and two partitions, one for the critical application and one for the non-critical one.

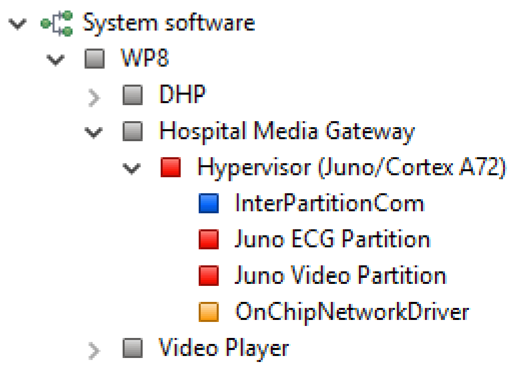
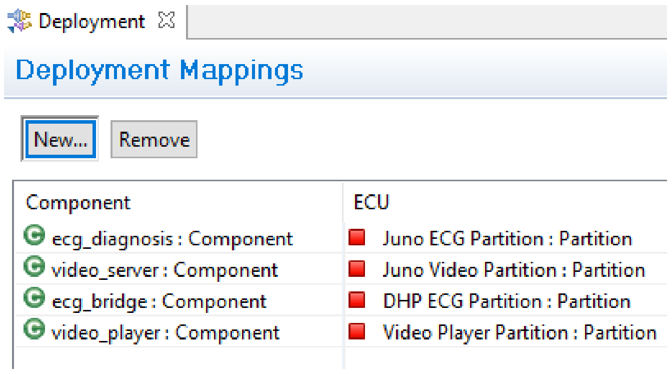

```
System software
  WP8
    DHP
    Hospital Media Gateway
      Hypervisor (Juno/Cortex A72)
        InterPartitionCom
        Juno ECG Partition
        Juno Video Partition
        OnChipNetworkDriver
    Video Player
```

Figure 66 - WP8: System Software.

5.2.2 Deployment

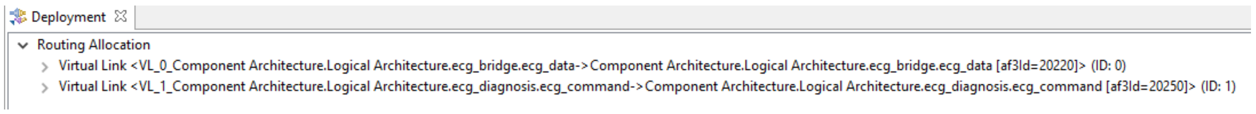
After having defined the System Software, the Deployment can be defined, which allocates each task to a Partition on some processor Tile, see Figure 67.

Next, virtual links need to be generated in the “Virtual Links” tab of the Deployment. The result consists of two TT virtual links, for the ECG related communication (see Figure 68). No virtual links are created for the best effort communication, since the corresponding frames are not scheduled and their routing path is determined dynamically at run-time.



Component	ECU
ecg_diagnosis : Component	Juno ECG Partition : Partition
video_server : Component	Juno Video Partition : Partition
ecg_bridge : Component	DHP ECG Partition : Partition
video_player : Component	Video Player Partition : Partition

Figure 67 - WP8: Deployment of tasks to partitions.



```
Routing Allocation
  Virtual Link <VL_0_Component Architecture.Logical Architecture.ecg_bridge.ecg_data->Component Architecture.Logical Architecture.ecg_bridge.ecg_data [af3Id=20220]> (ID: 0)
  Virtual Link <VL_1_Component Architecture.Logical Architecture.ecg_diagnosis.ecg_command->Component Architecture.Logical Architecture.ecg_diagnosis.ecg_command [af3Id=20250]> (ID: 1)
```

Figure 68 - WP8: TT VLs.

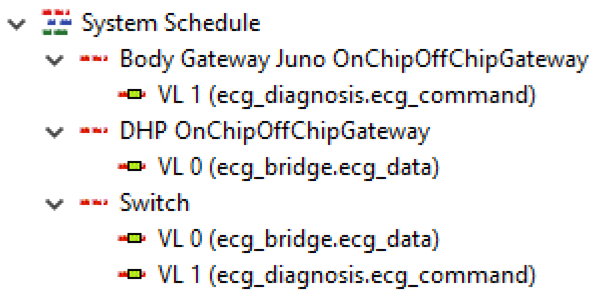
5.3 Configuration of Schedulers

5.3.1 Off-chip Scheduling of TT Virtual Links

At this point, sufficient information is available for generating the off-chip network communication.

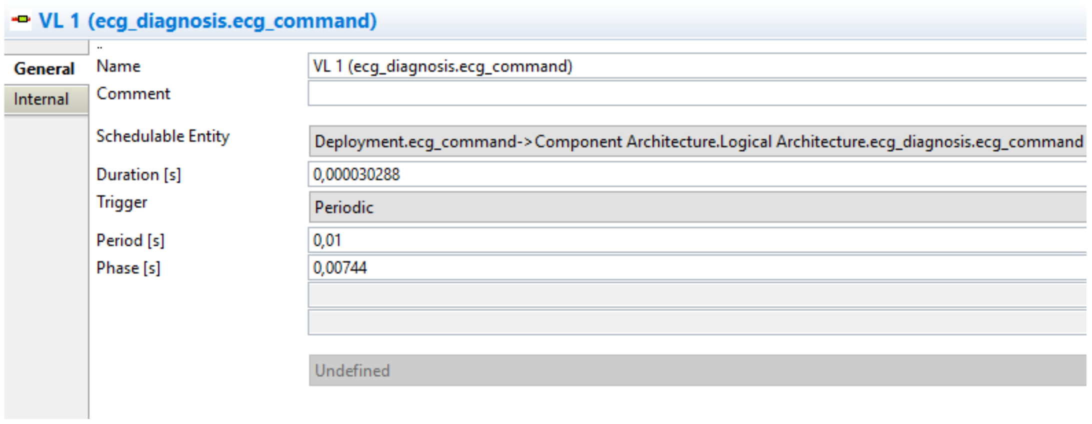
With the help of the "Generate TTEthernet Network Description" entry from the context menu of the "System Schedule" the ".network_description" file can be produced, which is the input for TTE-Plan. In order to import the defined schedules, the "Import TTEthernet Scheduling" entry from the context menu of the "System Schedule" must be executed.

The result of the import is the adding of scheduling entries for the time triggered communication of the TT VLs, as shown in Figure 69 and Figure 70.



- System Schedule
  - Body Gateway Juno OnChipOffChipGateway
    - VL 1 (ecg_diagnosis.ecg_command)
  - DHP OnChipOffChipGateway
    - VL 0 (ecg_bridge.ecg_data)
  - Switch
    - VL 0 (ecg_bridge.ecg_data)
    - VL 1 (ecg_diagnosis.ecg_command)

Figure 69 - WP8: off-chip scheduling parameter set.



VL 1 (ecg_diagnosis.ecg_command)

Name	VL 1 (ecg_diagnosis.ecg_command)
Comment	
Schedulable Entity	Deployment.ecg_command->Component Architecture.Logical Architecture.ecg_diagnosis.ecg_command
Duration [s]	0,000030288
Trigger	Periodic
Period [s]	0,01
Phase [s]	0,00744
	Undefined

Figure 70 - WP8: off-chip scheduling parameters for VL 1 in the Juno board.

5.4 Platform Building Block Configuration File Generation

5.4.1 Off-chip network communication configuration files

The generation of the textual configuration files for the off-chip network has already been performed in the previous step, at the same time as the generation of the scheduling parameters for the import into AF3, see Section 5.3.1.

6 Bibliography

[1] «D4.4.1 - Tools feature map and interoperability capabilities,» DREAMS Consortium, 7/2016.

[2] S. Barner, A. Diewald, F. Eizaguirre, A. Vasilevskiy et F. Chauvel, «Building Product-lines of Mixed-Criticality Systems,» *Proceedings on the Forum on Specification and Design Languages,* 2016.

[3] «D4.3.2 - First implementation and improvement of variability analysis and testing techniques for mixed critical systems,» DREAMS Consortium, 11/2015.

[4] «D4.3.3 – Final implementation and improvement of variability analysis and testing techniques for mixed critical systems,» DREAMS Consortium, 7/2016.

[5] «D7.3.2 - Wind power assessment report,» DREAMS Consortium, 2017.

[6] «D4.1.3 – Final implementation and improvement of the offline adaptation strategies for mixed criticality,» DREAMS Consortium, 7/2016.

[7] «D4.2.2 - Final implementation of a platform configuration files generator,» DREAMS Consortium, 7/2016.

[8] C. Pagetti, D. Saussié, R. Gratia et E. &. S. P. Noulard, «The ROSACE case study: From simulink specification to multi/many-core execution,» *20th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS),* 2014.

[9] «D2.2.2 - Report on monitoring, local resource scheduling and reconfiguration services for mixed criticality and security.,» DREAMS Consortium, 9/2015.

[10] «D1.4.1 - Meta-models for Application and Platform,» DREAMS Consortium, 3/2015.

[11] «D1.6.1 - Meta-models for platform-specific modelling,» DREAMS Consortium, 5/2016.

[12] A. Syed, G. Fohler et D. Gracia Pérez, «Online Admission of Non-Preemptive Aperiodic Mixed-Critical Tasks in Hierarchic Schedules,» *The 23rd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA),* August 2017.

[13] «D2.1.3 - RT-level design specifications of a) virtualization and memory interleaving support of the Spidergon STNoC backbone at the network interface layer and b) a bus-to-noc bridge for seamlessly interconnecting Spidergon STNoC and network gateways,» DREAMS Consortium, 9/2015.

[14] «D8.3.1 - Preliminary assessment report related to improving or calibrating the technological results,» DREAMS Consortium, 2017.

[15] «D1.5.1 - Intermediate integration of DREAMS platform with virtual platform prototype,» DREAMS Consortium, 4/2015.

[16] «D1.3.1 - Description of Development Process with Model Transformations,» DREAMS Consortium, 7/2014.

[17] «D4.4.2 - Integration and Support Report,» DREAMS Consortium, 2017.

[18] «D4.2.1 - Specification and first implementation of a platform configuration files generator,» DREAMS Consortium, 11/2015.

[19] «D5.4.1 - Guidelines for process and tool integration,» DREAMS Consortium, 7/2016.

[20] «D4.3.1 - Variability Analysis and Testing Techniques for Mixed-Criticality Systems,» DREAMS Consortium, 7/2015.

[21] «D4.1.2 - Definition of Offline Adaptation Strategies for Mixed-Criticality and Initial Implementation,» DREAMS Consortium, 3/2015.

[22] «D5.5.3 - Method for certifying mixed-criticality product lines,» DREAMS Consortium, 7/2016.

[23] «D5.2.2 - Prototype implementation of simulation framework for DREAMS architecture,» DREAMS Consortium, 11/2015.

[24] «D5.2.3 - Fault injection framework,» DREAMS Consortium, 7/2016.

[25] «D6.3.2 - Avionics assessment report,» DREAMS Consortium, 2017.

[26] «D8.3.2 - Assessment report for mixed-criticality healthcare and entertainment use cases,» DREAMS Consortium, 2017.

[27] «D2.3.4 - Hypervisor adaptation and drivers for local resource manager,» DREAMS Consortium, 2016.

[28] T. C. AG, «TTEPlan - The TTEthernet Scheduler.,» TTTech Computertechnik AG, 2015.

[29] «D1.2.1 - Architectural style of DREAMS,» DREAMS Consortium, 7/2014.

[30] «D5.2.1 Specification of Simulation Framework,» DREAMS Consortium, 3/2015.

[31] «ARINC Report 664P7-1. Aircraft Data Network, Part 7: Avionics Full Duplex Switched Ethernet (AFDX) Network,» Sept. 2009.

[32] «D2.3.1 - XtratuM support of enhanced hypervisor layer services. Description and interfaces,» DREAMS Consortium, 9/2015.

[33] «TTEPlan - The TTEthernet Scheduler / TTE-Plan User Manual v4.3, Document Number: D-TTE-G-01-010,» TTTech, July 2015.

[34] «SAE AS6802 Time-Triggered Ethernet Networking, Issuing Committee: As-2d2 Deterministic Ethernet And Unified,» SAE, November 2011.

[35] «D1.4.1 - Meta-models for Application and Platform,» DREAMS Consortium, 11/2015.

[36] IEEE, IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, IEEE Instrumentation and Measurement Society, July 24, 2008.

[37] «D1.7.1 - Final DREAMS Platform,» DREAMS Consortium, 8/2016.

[38] «D2.3.4 - Hypervisor adaptation and drivers for local resource,» DREAMS Consortium, 06/2016.

[39] Fent Innovative Software Solutions S.L., «Software User Manual: XM-ARM (14-035-03.005.sum.02),» January, 2016.

[40] DREAMS Consortium, «D1.5.1: Intermediate integration of DREAMS platform with virtual platform prototype».

[41] «D3.3.3 - Final Implementation of Cluster-Level Safety,» DREAMS Consortium, 7/2016.