



CHES Software Development Guide

14 April 2020

1 Table of Contents

1	Table of Contents	2
2	List of Tables	6
3	Document history	7
4	Introduction.....	7
5	Model the software components.....	7
5.1	Modeling Data Types.....	7
5.2	ComponentTypes.....	12
5.3	Interfaces.....	12
5.4	ComponentImplementations	14
5.5	Operations behavior	14
5.6	Modeling the Data Flow	15
5.7	Modeling the SW instances	17
5.8	Interactions.....	20
5.9	Modeling Operational Modes	21
5.10	Modelling Timing properties	22
5.10.1	Querying the timing properties.....	25
5.10.2	Instance View.....	25
6	Modelling the target platform and allocations	30
6.1	Modeling the HW entities	30
6.2	Defining the number of cores for a processor	31
6.3	Allocating SW instances to HW instances	31
6.3.1	Multicore Support Wizard	32
6.3.1.1	Assign Components to Cores.....	32
6.3.1.2	Assign Tasks to Cores.....	32
6.3.1.3	Generate Task to Core Assignments.....	33
6.3.2	Manually creating the Assign entities	33

6.3.3	Specifying the Operational Mode.....	33
7	Model validation.....	34
7.1	Configuring the CHES model validation features	34
8	Running Schedulability Analysis	34
9	End2End Response Time Analysis	39
10	Domain specific modeling	42
10.1	Avionics.....	42
10.1.1	Creating functional partitions.....	42
10.1.2	IMA Partitions Support Wizard.....	43
10.1.2.1	Assign Components to Partitions	43
10.1.2.2	Assign Partitions to Cores.....	43
10.1.3	ARINC Processes	44
11	Appendix.....	46
11.1	Tutorial: the CHES process - protected operation example	46
11.1.1	Create data types.....	46
11.1.2	Create interfaces	46
11.1.3	Create component types	48
11.2	Constraints.....	49
11.2.1	Functional View (included in "Core Constraints").....	49
11.2.1.1	Provided/Required ports of Interfaces (Connector_01).....	49
11.2.1.2	Connector: FlowPorts compatibility of direction (Connector_02)	49
11.2.1.3	Connector: FlowPorts compatibility of type (Connector_03)	49
11.2.1.4	FlowPorts not mapped on req/prov operations (FlowPorts_01)	49
11.2.1.5	FlowPorts parameters type compatibility (FlowPorts_02).....	50
11.2.1.6	FlowPorts parameters direction compatibility (FlowPorts_03)	50
11.2.1.7	ComponentType Interface (FV_02)	50
11.2.1.8	ComponentImplementation realization (FV_03).....	50
11.2.1.9	ClientServerPort (FV_05).....	50
11.2.1.10	ClientServerPort SPEC_KIND (FV_06)	50

11.2.1.11	ClientServerPort with kind PROREQ (FV_07).....	50
11.2.1.12	Interfaces (FV_08).....	50
11.2.1.13	Operations (FV_09).....	51
11.2.2	Extra-Functional View (included in “Core Constraints”, for schedulability analysis)	51
11.2.2.1	CHRTSpecification Context (EFVRT_01).....	51
11.2.2.2	CHRTSpecification occKind(EFVRT_02).....	51
11.2.2.3	CHRTSpecification PeriodicPattern (EFVRT_03)	51
11.2.2.4	CHRTSpecification PeriodicPattern (EFVRT_04)	51
11.2.2.5	CHRTSpecification partWithPort (EFVRT_05).....	51
11.2.2.6	CHRTSpecification SporadicPattern (EFVRT_20)	51
11.2.2.7	CHRTSpecification priority (EFVRT_30)	51
11.2.2.8	CHRTSpecification localWCET (EFVRT_40)	52
11.2.3	Deployment View (included in “Core Constraints”)	52
11.2.3.1	Component Implementation deployment (DV_01)	52
11.2.3.2	Assign must have exactly one “from” and one “to” (DV_02).....	52
11.2.3.3	HwBus packett (DV_03).....	52
11.2.3.4	HwBus attributes (DV_04)	52
11.2.3.5	Used Functional Partition allocated on exactly one Core (DV_05)	52
11.2.3.6	Un-used Functional Partition allocated on exactly one Core (DV_06).....	52
11.2.4	Analysis View	53
11.2.4.1	AnalysisView unique package AV_01	53
11.2.4.2	SaAnalysisContext Platform (AV_03).....	53
12	References.....	53

List of Figures

Figure 1	CHES Funct View - Modeling Data Types.....	7
Figure 2:	Modeling default value for attribute.....	7
Figure 3	Functional View - Property Default Value Setting.....	8
Figure 4	Modeling Data Ranges using MARTE BoundedSubtype (1).....	8
Figure 5:	Modeling Data Ranges using MARTE BoundedSubtype (2)	9
Figure 6	Modeling Arrays using MARTE CollectionType (1)	9
Figure 7	Modeling Arrays using MARTE CollectionType (2)	10
Figure 8	Modeling Arrays using MARTE CollectionType (3)	10
Figure 9	Modeling Structures using the MARTE VSL TupleType (1).....	11
Figure 10	Modeling Structures using the MARTE VSL TupleType (2).....	11
Figure 11	Modeling Structures using the MARTE VSL TupleType (3).....	12

Figure 12: Interface dependencies and realizations	13
Figure 13 Selection of Provided (or Required) Interface for a Client Server Port	14
Figure 14: Modeling Activity Diagram	15
Figure 15 Mapping a Flow Port to a Client Server Port	16
Figure 16 CHES Build Instances command for Sw System	18
Figure 17 CHES Build Instances command for HW System	19
Figure 18: Instances specifications	20
Figure 19 Functional View - Sequence Diagram palette	20
Figure 20 Sequence Diagram	21
Figure 21 Naming convention for message numbering	21
Figure 22: Working with the CHRtSpecification in the CSD (as instance view)	24
Figure 23: CHES tab for the CHRtSpecification	24
Figure 24: CHRtSpecification property tab	25
Figure 25: InstanceView	26
Figure 26: Reading with the InstanceView	27
Figure 27: Reading with the InstanceView - hyperlinks	28
Figure 28: Creating with the InstanceView	29
Figure 29: Updating with the InstanceView	29
Figure 30 HW Types in a Class Diagram in the Deployment View	30
Figure 31 Deployment View Class Diagram Palette	30
Figure 32: Composite Structure Diagram in the Deployment View	30
Figure 33 Build instances for HW System	31
Figure 34 Assign Components to Cores Wizard	32
Figure 35 Assign Tasks to Cores Wizard	32
Figure 36: Modeling the SW to HW allocation	33
Figure 37: modelling Allocations constrained to OperationalModes	34
Figure 38: Schedulability Analysis Context	35
Figure 39 CHES Schedulability analysis	35
Figure 40 CHES Select Analysis Context window	36
Figure 41 Analysis in execution	36
Figure 42 Schedulability analysis results	37
Figure 43: Analysis context tab	37
Figure 44 CHES Compare Analysis Results Command	38
Figure 45 CHES Analysis Results Comparison	38
Figure 46 End2End Activity in End2End Analysis Package	39
Figure 47 Setting the Activity properties for End2End Analysis	39
Figure 48 Activity Diagram for End2End Analysis	40
Figure 49 End2End Analysis Context	40
Figure 50 End2End Analysis Command	40
Figure 51 End2End Analysis Context selection	41
Figure 52 End2End Analysis Result pop-up window	41
Figure 53 End2End Analysis results in Activity diagram SaEnd2EndFlow properties	42
Figure 54 End2End Context updated with results	42
Figure 55 Assign Components to Partitions Wizard	43

Figure 56 Assign Partitions to Cores Wizard..... 44

Figure 57: ARINCComponentImpl..... 44

Figure 58: providing real time properties for ARINCProcess and ARINCFunction operations 45

Figure 59 End2End Analysis Package..... 45

Figure 60 Creating data types..... 46

Figure 61 Creating interfaces 46

Figure 62 Creating operations 47

Figure 63 Defining operation's parameters..... 47

Figure 64 Assigning a type to operation parameters 48

Figure 65 Modeling interfaces and operations 48

Figure 66 Creating component types 49

2 List of Tables

<empty>

3 Document history

Date	Changes
14 April 2020	First version

4 Introduction

This guide illustrates the usage of CHES tool related to the modelling of software architectures and model based schedulability analysis. [Views used for these activities are the Component, Deployment and Analysis Views.](#)

5 Model the software components

5.1 Modeling Data Types

⚠️ OPTIONAL: primitive data types are supported by default in UML

Primitive Types are supported: they can be created in a Class Diagram in the CHES ComponentView, as illustrated in Figure 1.

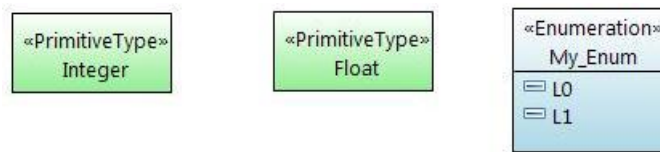


Figure 1 CHES Funct View - Modeling Data Types

In a given Component, **Constants** can be modeled by selecting the ‘read-only’ attribute of the given Property (see Figure 2).

The screenshot shows a UML class diagram with a component 'MyComponentType' containing a constant 'MyConstant: MyInteger'. Below the diagram, the 'Properties' window for 'MyConstant: MyInteger' is open. The 'UML' tab is selected, and the 'Is read only' property is checked. A tooltip for the 'Is read only' property reads: 'isReadOnly: States whether the feature's value may be modified by a client.'

Figure 2: Modeling default value for attribute

A **Default value** can be specified by setting the 'default value' attribute of the property, for instance by using a Literal-Integer (see figure Figure 3).

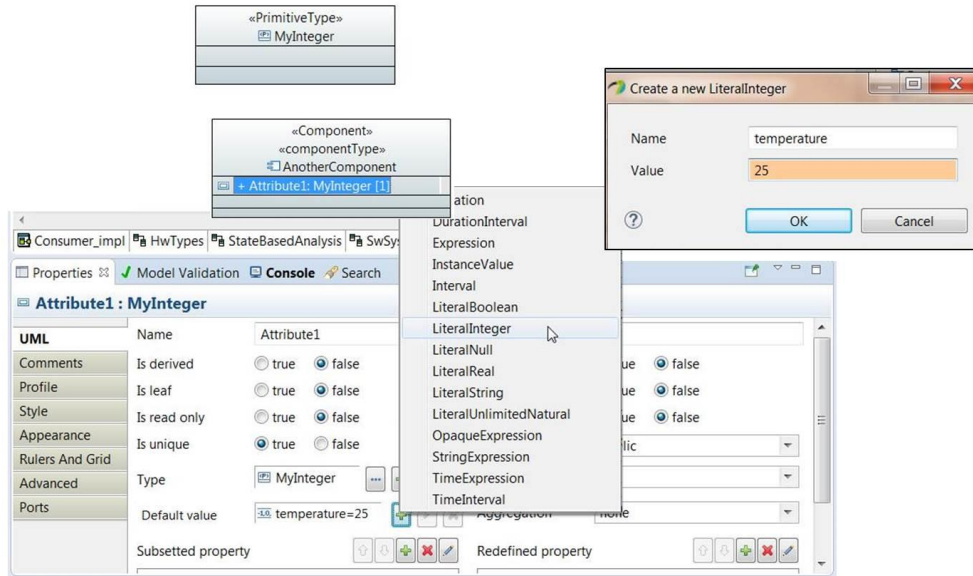


Figure 3 Functional View - Property Default Value Setting

Data ranges: to specify a data type that is an interval, use the MARTE::VSL::DataTypes::BoundedSubtype stereotype, from the palette (see Figure 4).

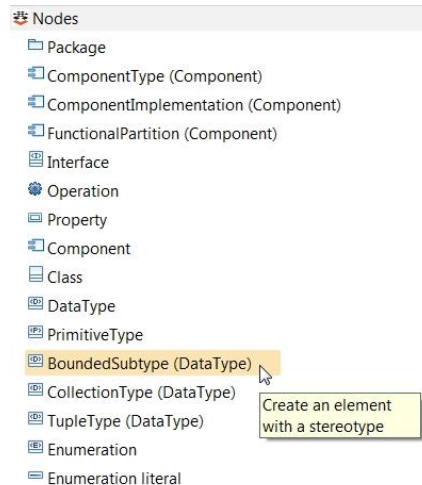


Figure 4 Modeling Data Ranges using MARTE BoundedSubtype (1)

From the profile tab below, assign values to the following fields: baseType, minValue, maxValue, isMinOpen, isMaxOpen, as depicted below.

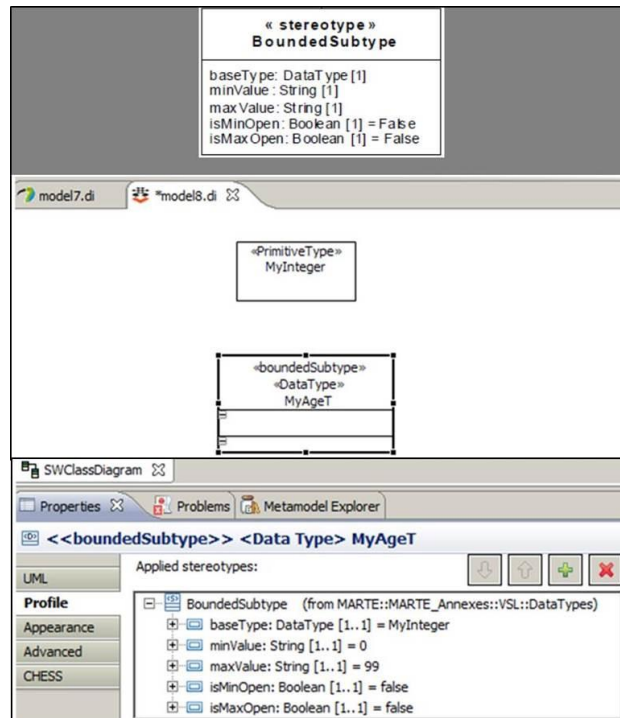


Figure 5: Modeling Data Ranges using MARTE BoundedSubtype (2)

Another possibility for data ranges would be to use the IntervalType from MARTE VSL, but this is currently not properly implemented in Papyrus.

Arrays can be modeled selecting from the palette the CollectionType from MARTE VSL, as illustrated in Figure 6.

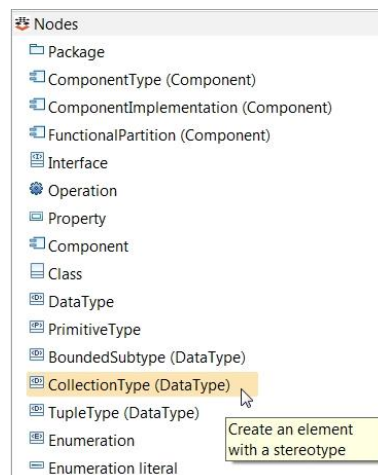


Figure 6 Modeling Arrays using MARTE CollectionType (1)

Once you have created a CollectionType (e.g. myArray), you must create its property (e.g. myArrayElement) (selecting the property icon from the palette) and assign it the desired type and multiplicity using the UML tab in the properties below.

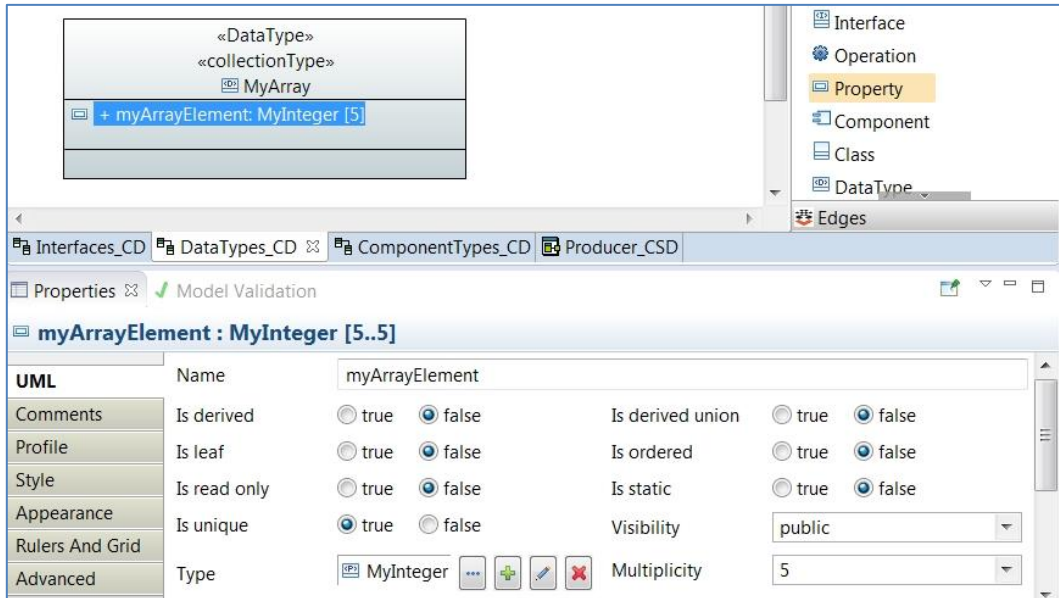


Figure 7 Modeling Arrays using MARTE CollectionType (2)

Then, from the properties below, select the Profile tab and set the collectionAttrib to the array’s element that has just been created as illustrated in Figure 8.

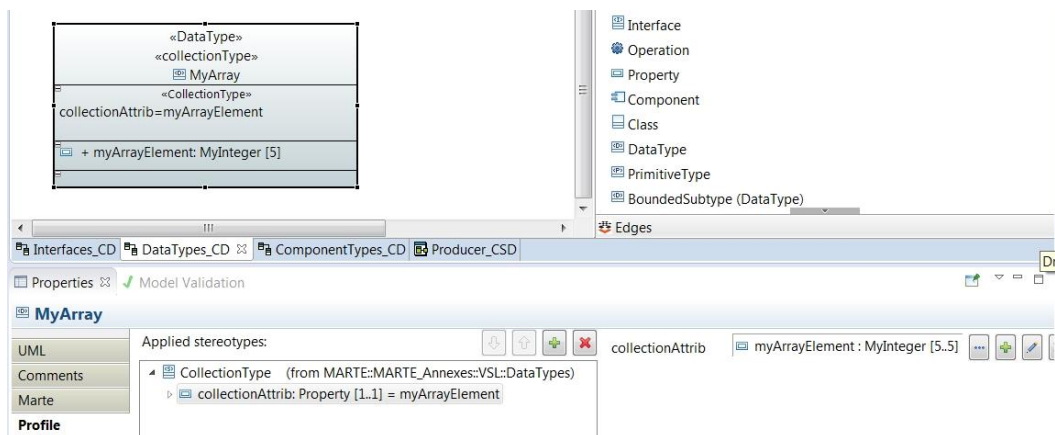


Figure 8 Modeling Arrays using MARTE CollectionType (3)

Structures can be modeled by selecting from the palette the TupleType from MARTE VSL as illustrated in Figure 9.

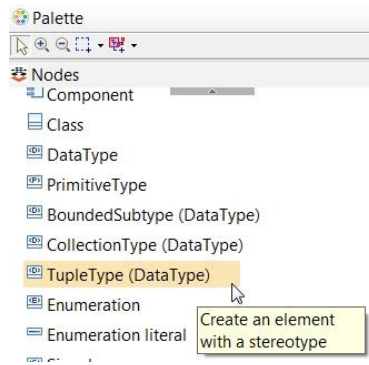


Figure 9 Modeling Structures using the MARTE VSL TupleType (1)

Once you have created a TupleType (e.g. MyStructure), you must create its fields (Properties), and assign them the correct type from the UML tab in the properties below, as depicted in Figure 10.

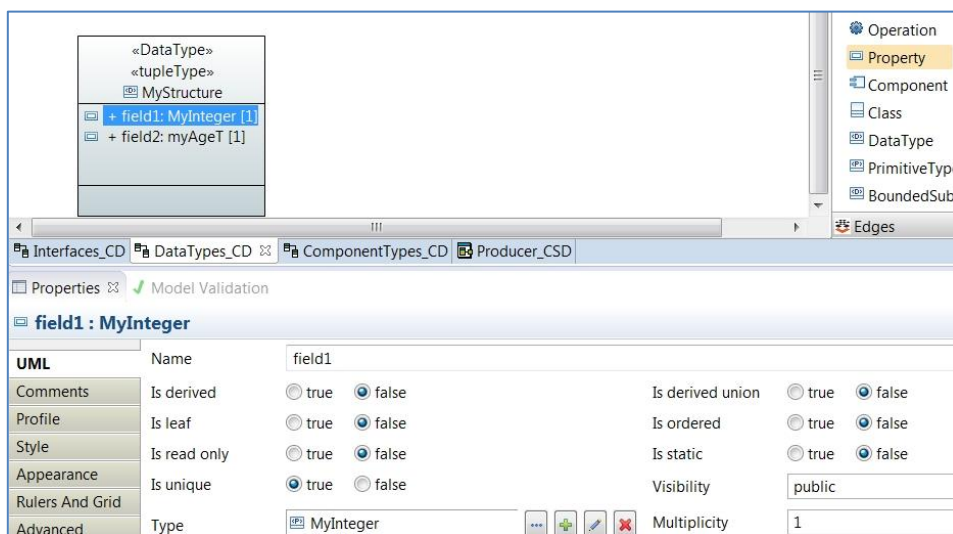


Figure 10 Modeling Structures using the MARTE VSL TupleType (2)

Finally, once you have created the needed fields, assign the tupleAttrib to the newly created fields from the Profile tab in the properties below, as depicted in Figure 11.

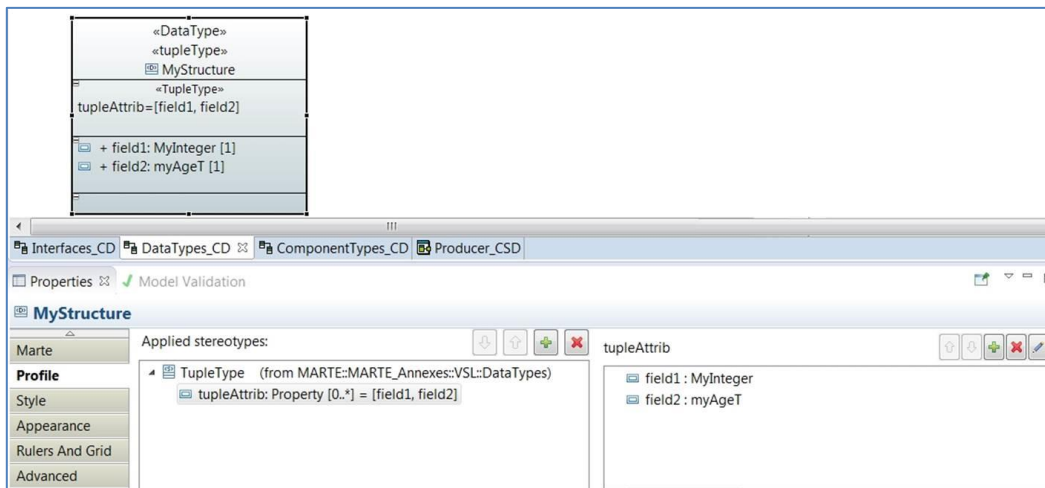


Figure 11 Modeling Structures using the MARTE VSL TupleType (3)

5.2 ComponentTypes



OPTIONAL: required to perform Ada code generation

ComponentTypes , as part of the CHES component model, can be modelled in the ComponentView; by the way they are not mandatory.

To create a ComponentType:

- Open a Class Diagram
- Select ComponentType from the CHES Funct View palette and click on the diagram
 - Give a proper name to the ComponentType

Needed FlowPorts are created and typed for the Component Type in its Composite Structure Diagram.

To create a FlowPort for a ComponentType:

- Select the ComponentType from the Model Explorer
- Create its Composite Structure Diagram (right-click->Create new Diagram->Composite Structure Diagram)
- Select IN Flow Port or OUT Flow Port icon from the CHES FunctView palette
- Click on the border of the ComponentType

5.3 Interfaces

For each Component Type (or ComponentImplementation, if the ComponentType is not modelled), provided and/or required Interfaces are also modeled in the Component View with the offered operations and parameters.

To create an Interface for a ComponentType:

- Open a Class Diagram For instance the one where the ComponentType is depicted
- Select the Interface icon from the CHES FunctView palette

- Click in the Class Diagram
- Give a proper name to the Interface

To create a ClientServerPort for a ComponentType:

- Open the Composite Structure Diagram of the ComponentType
- Select the Provided Port or the Required Port icon from the CHES FunctView palette
- Click in the Composite Structure Diagram inside the ComponentType
- Select “Profile” Tab, select the ClientServerPort applied stereotype
- Set the value of property “provInterface” or “reqInterface” (according to the value of the “kind” property) selecting an interface from the dialog window as shown in Figure 13.



The last step will **automatically create** and link in the model the ComponentType and the Interface with a **Dependency** (required Interface) or with an **InterfaceRealization** (provided Interface). Dependency and InterfaceRealization can be shown in the diagram by drag and drop them from the model browser to the diagram.

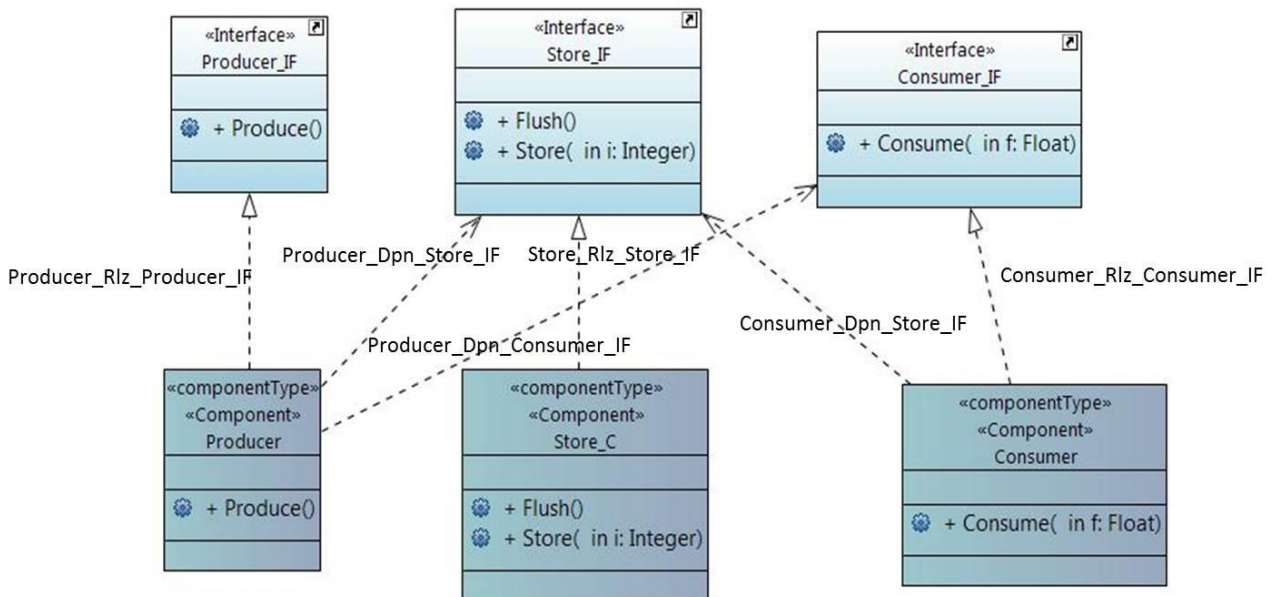


Figure 12: Interface dependencies and realizations

Operations are defined for the Interfaces by selecting the “Operation” icon from the CHES FunctView palette and clicking in the operation area of the Interface. If a ComponentType is linked with an InterfaceRealization to that Interface, the Operation is also automatically added to the ComponentType. Changes in the Operation name and parameters (name, type and number) are propagated to the linked ComponentType’s Operation.

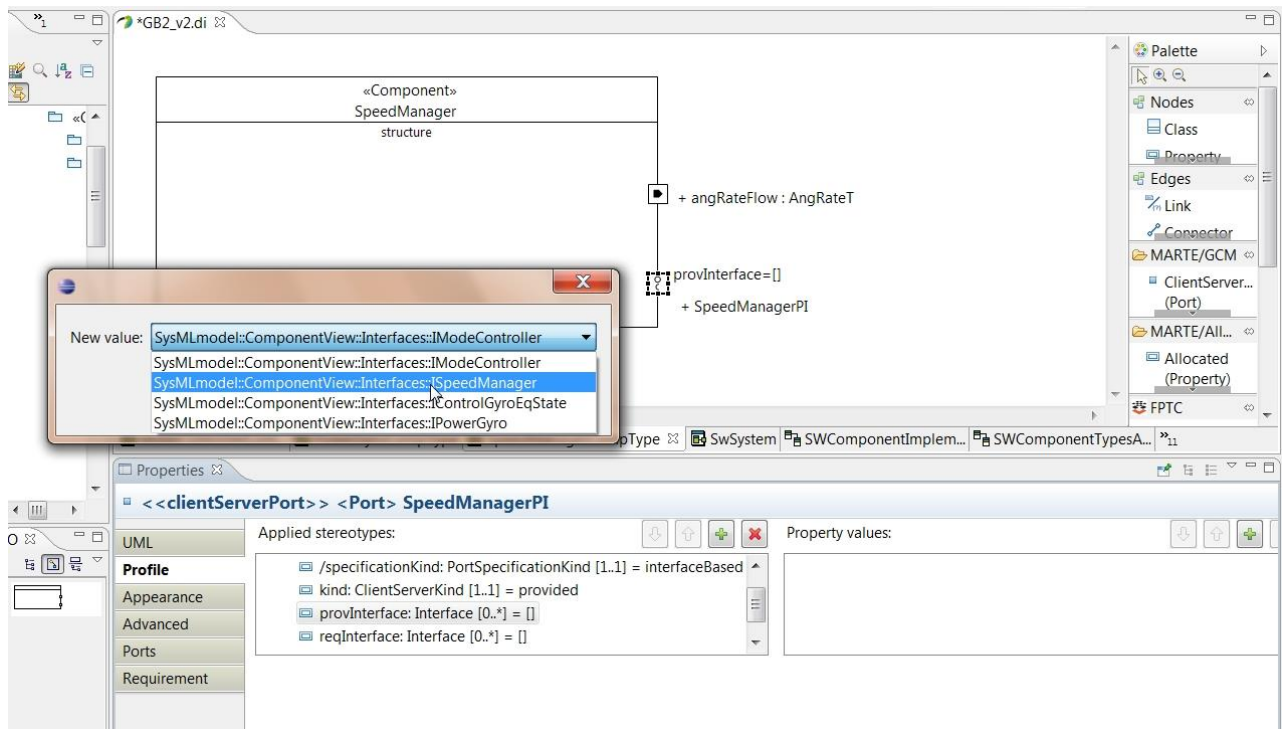


Figure 13 Selection of Provided (or Required) Interface for a Client Server Port

5.4 ComponentImplementations

To create a ComponentImplementation:

- Open a Class Diagram
- Select ComponentImplementation from the CHESS FunctView palette and click on the diagram
 - Give a proper name to the ComponentImplementation
- Select Realization from the CHESS FunctView palette and trace the link from the ComponentImplementation to the ComponentType that it is supposed to implement.

The last step **will automatically propagate the Ports and the Operations** of the ComponentType to the ComponentImplementation. Changes in the Operations of the Interface are propagated also to the ComponentImplementation.

5.5 Operations behavior

⚠ OPTIONAL: required to analyse blocking time for threads execution

An Activity Diagram can be created for a given Operation of a ComponentImplementation to model intra ComponentImplementation bindings, i.e. the called Operations (information used by Schedulability Analysis).

An Activity diagram can be created through the Eclipse Model Explorer by right-clicking on the Operation of the ComponentImplementation and selecting the *New Diagram -> Create a New Activity Diagram* command.

Next figure shows an example of activity diagram for the Produce operation implemented by the Producer_impl Component Implementation; the calls to the operations Store and Consume are modeled in

the diagram. In particular see the information provided for CallOperationAction named Call Store_IF.Store: the OnPort refers the required port Store_IF_RI, defined for Producer_impl, while the Operation field refers the Store operation which is defined in the Store_IF interface which in turn is required through the Store_IF_RI port.

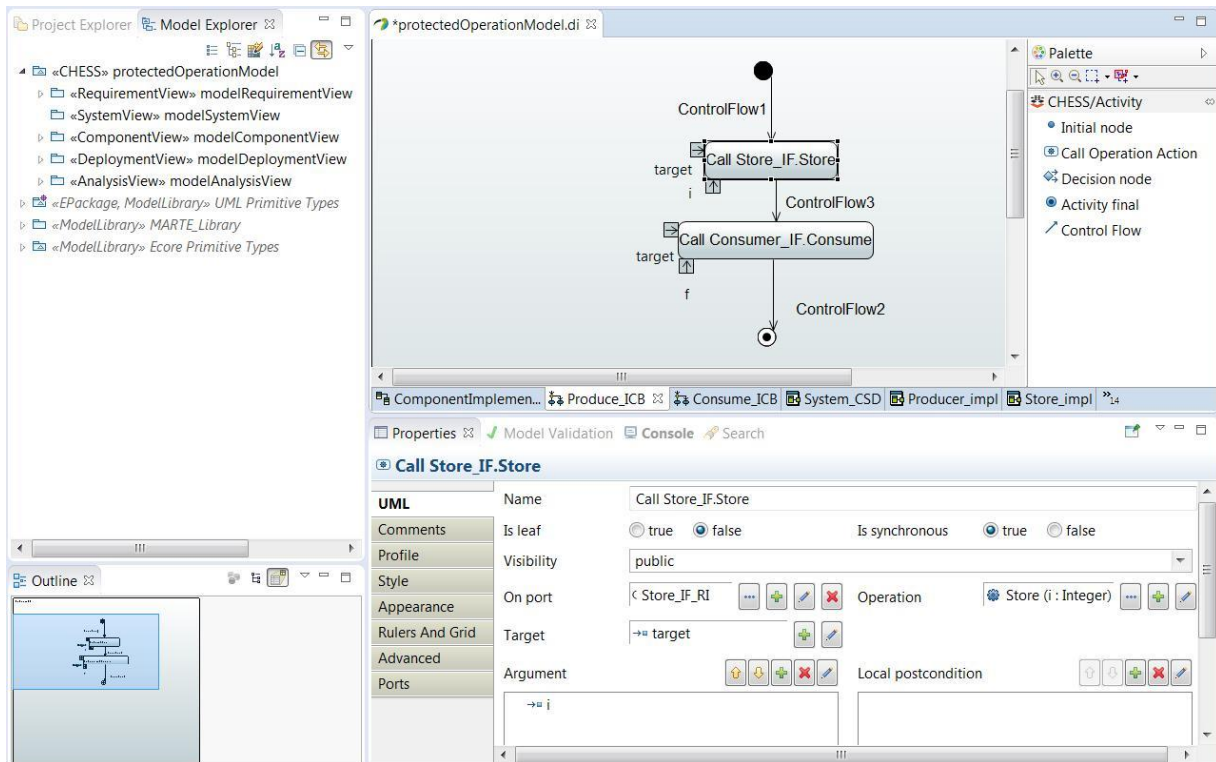


Figure 14: Modeling Activity Diagram



Currently Decision nodes are not supported by the Schedulability analysis.

5.6 Modeling the Data Flow



DataFlow between system, software and hardware components can be modeled in the SystemView, ComponentView, DeploymentView respectively by using ports stereotyped as FlowPorts (from SysML, in the SystemView, or MARTE, in the Component and DeploymentView).



A special case regards the usage of flow ports in the Component View to model data-flow between ComponentImplementations.

According to the current CHES component model definition, data flow ports are not considered during the PSM generation, i.e. the generation of Containers and Connectors; in other words data-flow ports do not have support at code generation level. However the design of data-flow for ComponentImplementations is allowed in CHES because it can be a useful modelling step, especially in the preliminary phase of the software design, for instance as derived from a system one; then, in order to complete the software design, the modeller has to create provided/required operations ports for the ComponentImplementations. The

operations parameters allow to actually implement the data flow originally modelled for the ComponentImplementations through the data-flow ports.

So CHES requires that data-flow ports implementing data flow between ComponentImplementation must be allocated to operations parameters; a FlowPort owned by a ComponentImplementation should be mapped to an existing ClientServerPort of the component itself in order to model how the flow of data for the component is actually realized through its provided and required operations.



The mappings can be modeled using the “Port” tab in the Property View: Select a FlowPort in the editor, choose an operation from a provided or required interface and select a parameter that shares the same type as the FlowPort. The mapping will be created and displayed in the tab.

Note that the latest version of the CHES component model includes event data ports, this kind of ports does not need to be allocated to operations parameters.

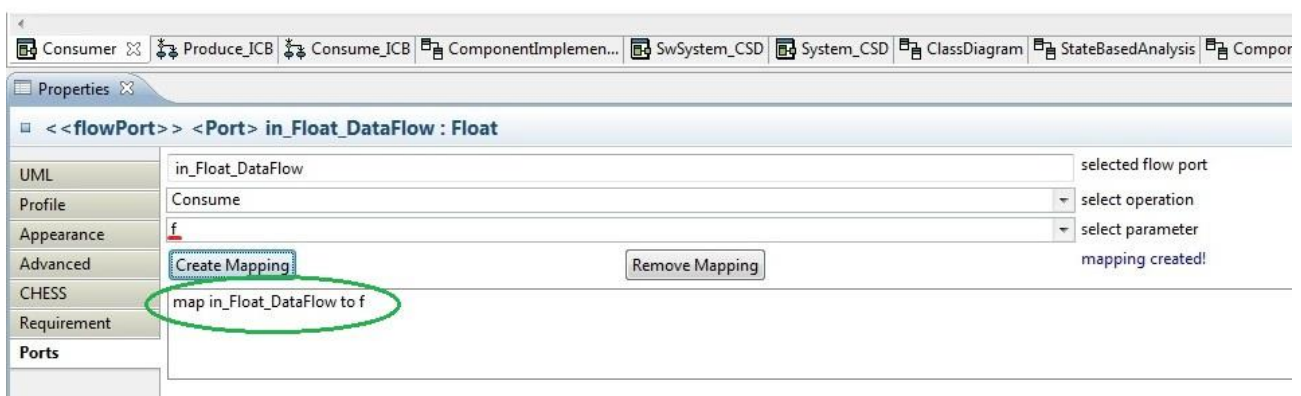
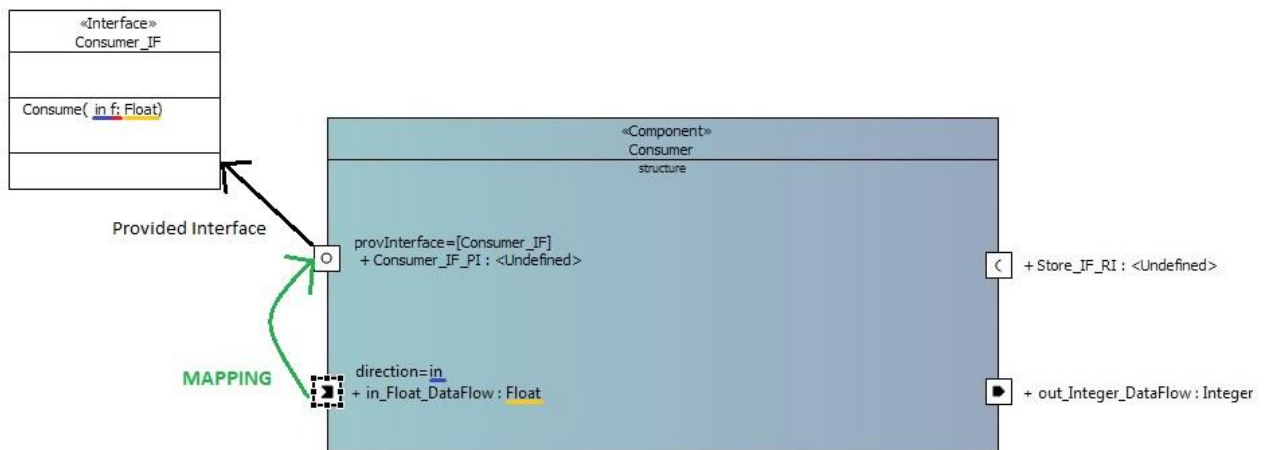



Figure 15 Mapping a Flow Port to a Client Server Port

5.7 Modeling the SW instances

 UML defines the concept of InstanceSpecification to represent an instance of a given entity (e.g. SW or HW component). Currently in the tool, components instances can be automatically obtained starting from the specification of the components themselves. In particular instances owned by a given component can be represented through UML Composite structure diagrams, so by using the concept of UML part (as part of a composite component). So starting from a given component, the hierarchy of component instances can be automatically derived according to the decomposition modelled with the composite diagrams. In the following we will use the terms “instances” and “parts” to refer to the same concept, where not otherwise specified.

ComponentImplementations are instantiated to model a specific software system; it is worth noting that the latter can be represented by a ComponentImplementation as well. Once ComponentImplementations are instantiated, they can be connected together through the available provided and required ports.

According to the inheritance of UML definition, it is possible to model ComponentImplementation instances owned by a given ComponentImplementation in the following two ways:

- in a Composite Structure Diagram: drag a ComponentImplementation from the Papyrus Model Explorer into the Composite Structure Diagram of the containing entity (Component Implementation) to model its instantiation as a part of it. Then press **F4** to choose the ports (border items) to be showed for the selected instance.
- In a Class Diagram: drag a ComponentImplementation and the containing entity (Component Implementation) from the Papyrus Model Explorer into a Class Diagram and draw an Association link between them, directed from the container to the component implementation. From the UML tab select the member end corresponding to the component implementation and set Owner to “Classifier” (and Aggregation to “composite”). This will instantiate the component implementation in its container.

However in CHESS we recommend the usage of Composite Structure Diagram to model instances, given that composite diagrams are mandatory to model the connections between the ComponentImplementation instances and to model some extra functional properties.

 Note that **only one level of decomposition for a given component can be designed by using the composite diagram.**

When ComponentImplementation instances and their ports are available in the Composite Structure Diagram, links between ComponentImplementation ports can be created by using the Connector tool.

Finally, when the internal structure of each component has been defined, it is possible to derive the Instance Model, i.e. the corresponding InstanceSpecification representation, by using the dedicated command in the CHESS main menu in the toolbar.

! Note that the InstanceModel allows to properly represent the instances hierarchy at any level of decomposition, and in particular to represent the proper extra functional information attached to each instances/ports, at any level of the hierarchy (this is not possible by using the composite diagram only).

! Note that the creation of the InstanceModel is a mandatory step in CHESS, given that several design steps (e.g. the allocation of software to hardware instances) can only be performed by using the entities available in the InstanceModel.

To invoke the Build Instance command:

- from the Model Explorer, select the CHGaResourcePlatform that represents the SW System (in the Component View), or the CHGaResourcePlatform that represents the HW configuration on which the system is deployed (in the Deployment View) as the root entity for which the instance model has to be created and right click on it,
- select the command CHESS-> Build Instances.

Notice that Build Instances must be invoked both for the SW system (Figure 16) and for the HW system (Figure 17). Different HW systems can be defined (with their corresponding instances created by means of the Build Instances command) in order to model different deployments of the system and perform separate analysis on each of the different deployments, for comparing them and identifying the best deployment configuration.

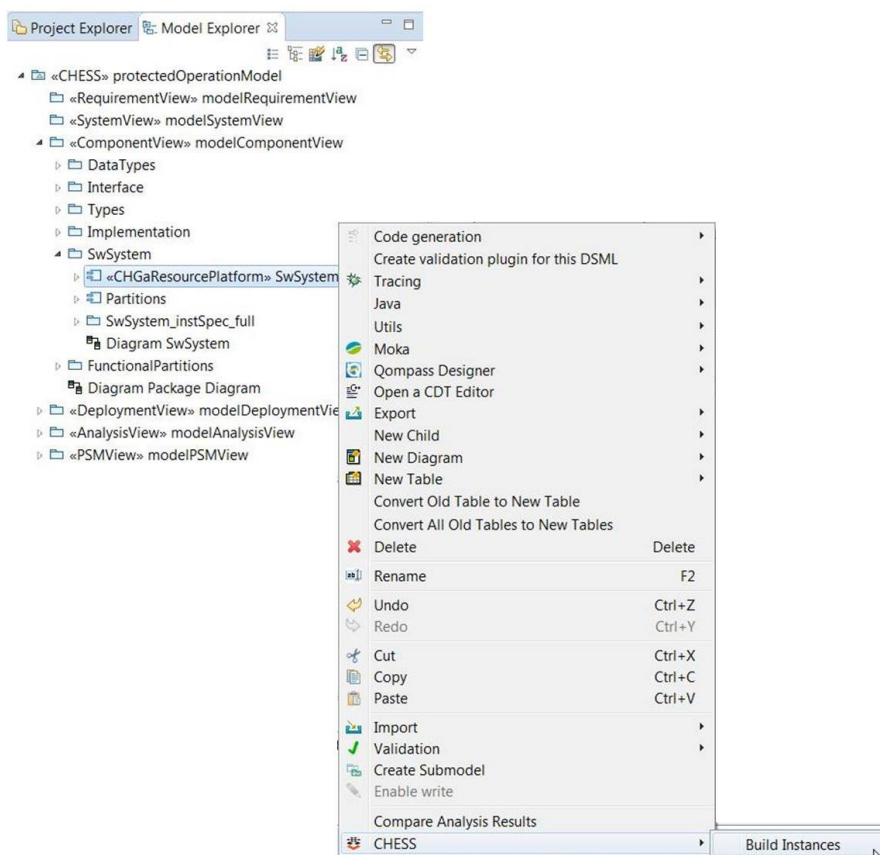


Figure 16 CHESS Build Instances command for Sw System

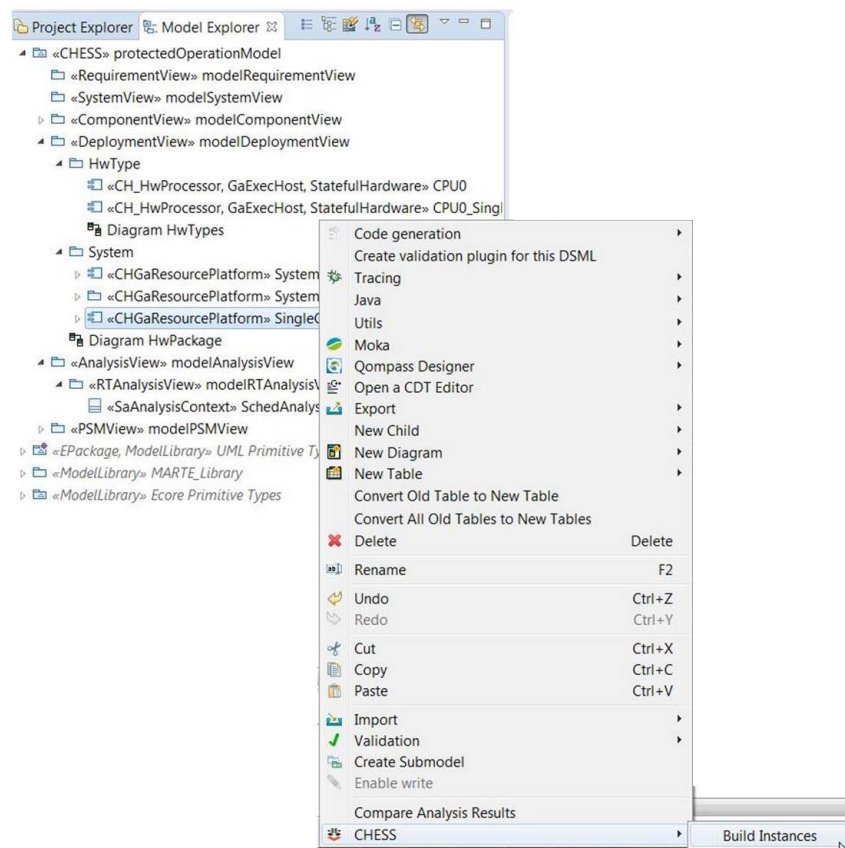


Figure 17 CHES Build Instances command for HW System

then the InstanceSpecifications/Slots are generated in a dedicated package which is created in the model at the same level of the selected root entity.

The Next figure shows the instances generated; in particular see the SwSystem_instSpec package in the Model Explorer. It is possible to see the instance related to the SwSystem entity and the instances related to the owned Properties (i.e. the ones named SwSystem_Producer_inst, SwSystem_Consumer_inst and SwSystem_Store_inst); the unnamed instances are the ones related to the designed connectors.

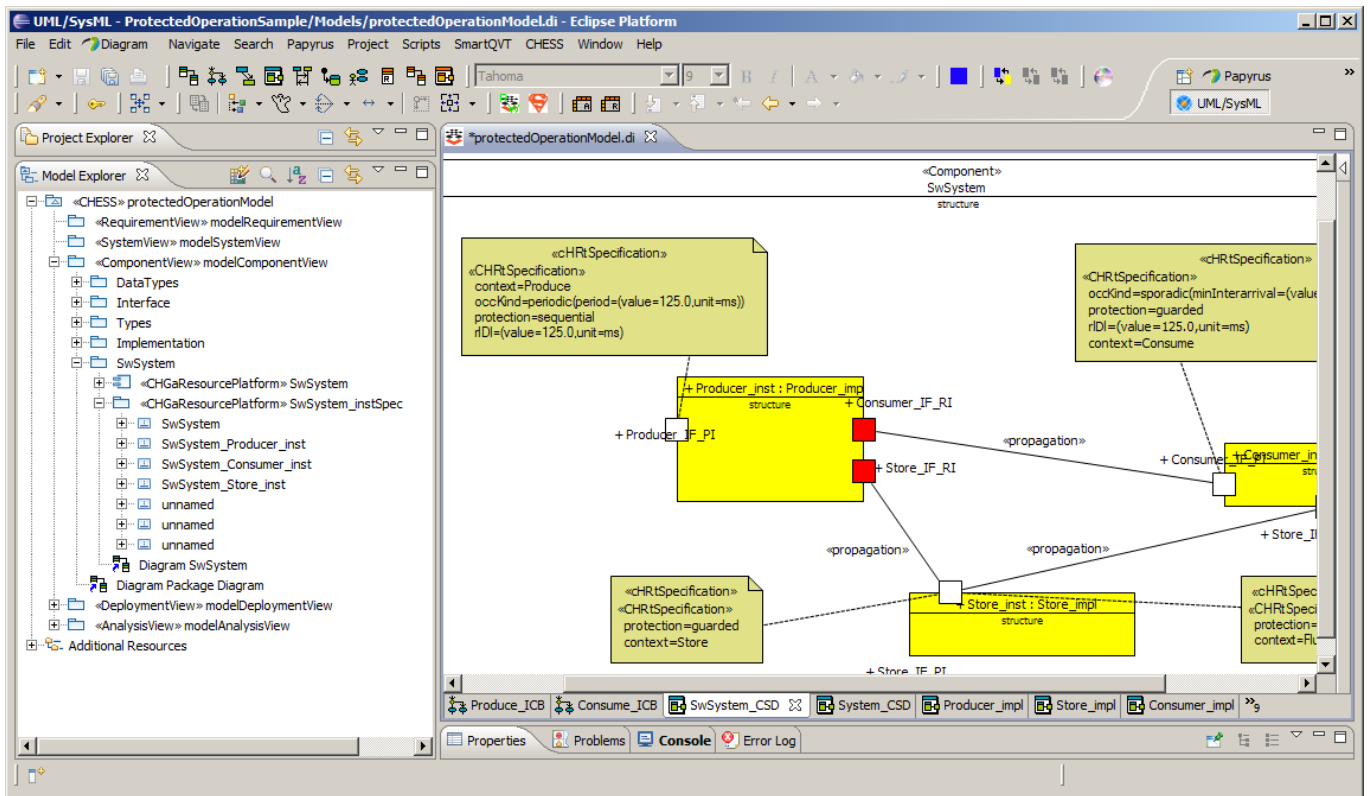


Figure 18: Instances specifications

See section 5.10.2 about the InstanceView.

5.8 Interactions

OPTIONAL - Required to enable end to end response time analysis

This diagram is used for modelling a sequence of operations on which to perform End2End response time analysis (see 9).

A subset of the modeling elements is available from the palette for Sequence Diagrams in the Functional View, as illustrated in the figure below.

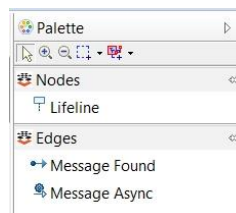


Figure 19 Functional View - Sequence Diagram palette

Message Found is used to represent the starting event of the sequence. It usually corresponds to an event in the environment.

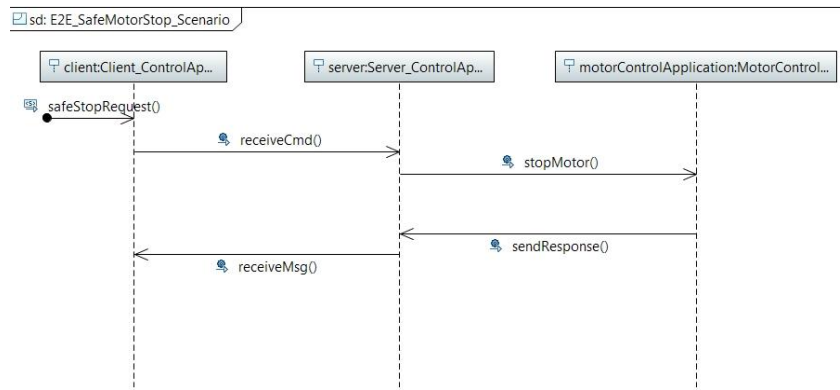


Figure 20 Sequence Diagram

! Sequence diagrams allow to represent ComponentImplementation parts (parts/property owned by composite components) collaborating together; the parts in the sequence diagram are represented by the lifelines. To specify the mapping between a part and a lifeline, the value of the Lifeline *represents* feature must be set to the given part/property (typed with ComponentImplementation): this can be done through the UML tab of the Property view.

! Ordering of messages is not automatically supported due to a limitation in Papyrus 1.0.1. To enforce the ordering of messages, required for the End2End response time analysis, it is necessary to use the simple naming convention for the created messages as illustrated in the figure below.

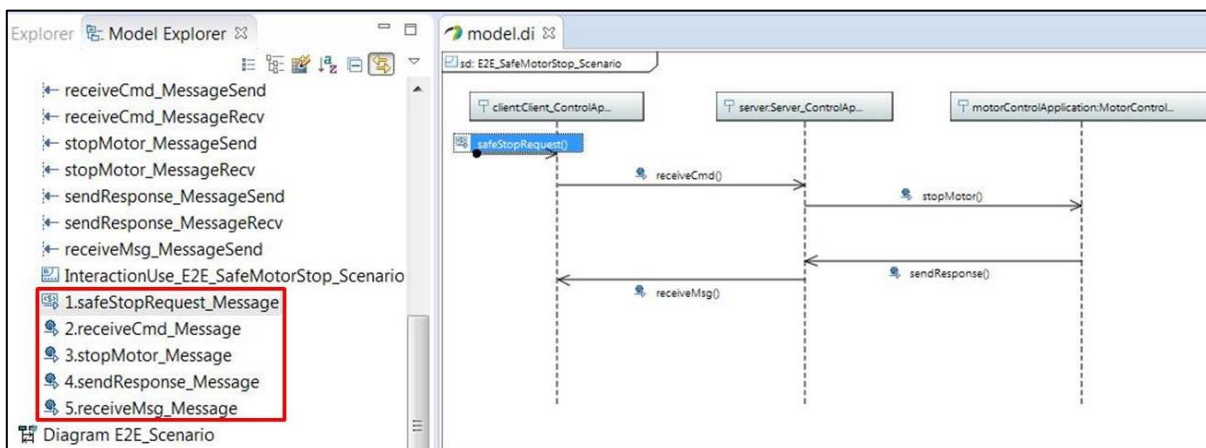


Figure 21 Naming convention for message numbering

5.9 Modeling Operational Modes

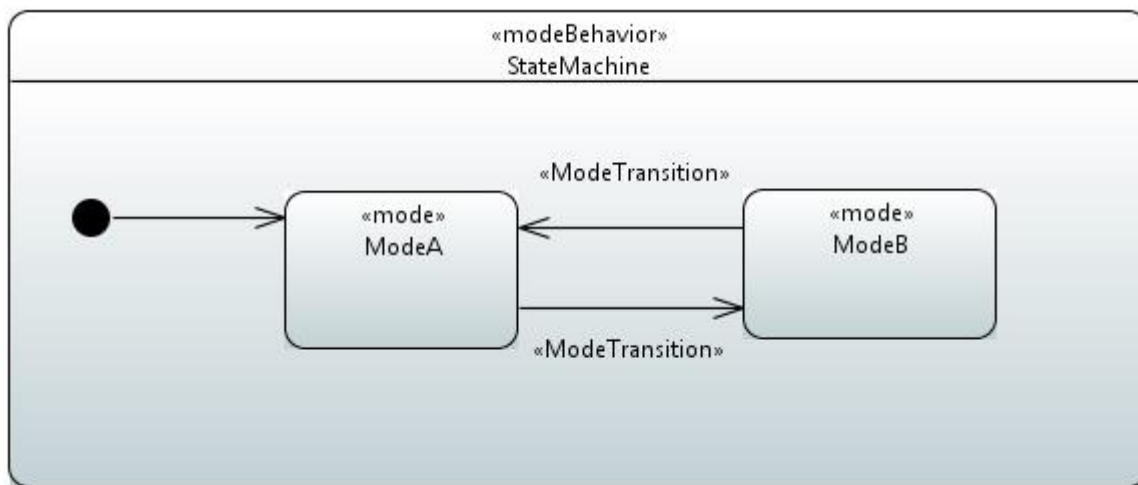
! OPTIONAL

Operation modes can be modelled in the FunctionalView by using stereotyped state machine.

To model operational model, assuming to have a root SW component, perform the following steps:

- Create a state machine for the component (in the ModelExplorer view, right click on the component and select “new diagram-> create a new UML state machine diagram”)
- Stereotype the state machine as ModeBehavior (by using the Profile tab in the Properties view)
- Create and Initial State
- Create additional States, where each State has to be stereotyped with Mode (so each State represents an operational mode)
- Design transition between states; each transition has to be stereotyped with ModeTransition.

Once the modes have been modelled, it is possible to use them for the specification of the extra functional properties (so the properties can have different values depending on the given mode) and to model mode-dependent allocation of SW to HW.



5.10 Modelling Timing properties

CHRTSpecification allows attaching real time properties to an operation provided by a ComponentImplementation instance.

CHESS allows to use Composite Structure Diagrams (CSD) as object view **in the context of a given component** (so without the possibility to focus on a given instance in a well identified level of the instance model hierarchy); in particular in CSD a Property corresponds to a (software or hardware) instance. Then the Build Instances command can be used to automatically generate the instance model, together with their extra functional information eventually modelled in the CSD.

Note that in case of multiple instances of the same decomposed component C, the extra functional information eventually modelled in the composite diagram of C will be replicated for each instances of C: this is the limitation of the composite diagram wrt the usage of the instance model. To overcome this limitation the modeller can use the **InstanceView** (see section 5.10.2).

⚠ The usage of CSDs for extra functional annotation upon instances works well for a given system with only one level of decomposition (i.e. the root system entity together with the owned components instances).

To create a CHRTSpecification in a composite structure diagram the Extra Functional View has to be activated in the ComponentView.

Once the CHRTSpecification has been created, it must be linked to the target port by using the Link item on the palette; then the following properties of the CHRTSpecification stereotype must be provided through the Profile tab in the Properties Eclipse view:

- *partWithPort* (only relevant in CSD): the Property subject of the CHRTSpecification, i.e. the Property owning the Port which has been linked to the CHRTSpecification
- *context*: the operation which has to be decorated with real time properties. The operation must be owned by the ComponentImplementation which types the partWithPort. Moreover the operation must be one which is also defined in the interface which is provided by the port linked to the ChRtSpecification.
- *relativePriority*: an integer that represents the priority of the operation that is decorated with real time properties w.r.t. the other operations. The higher the value, the higher the priority.

Then real time properties can be set through the following CHRTSpecification stereotype fields by using the MARTE (VSL) language:

- *occKind*: specifies the activation event for the current operation. It can be periodic or sporadic, e.g.:
 - *periodic(period=(value=125.0,unit=ms) , phase=(value=10,unit=ms)¹)*
 - *sporadic(minInterarrival=(value=125.0,unit=ms))*
- *localWCET*: the worst case execution time for the given operation, e.g. *(value=45.0,unit=ms)*
- *protection*: must be set to
 - *sequential* (for cyclic or sporadic operation)
 - *guarded* (for protected operation)
- *rDI*: the deadline, e.g. *(value=125.0,unit=ms)*

¹ The “phase” attribute is optional

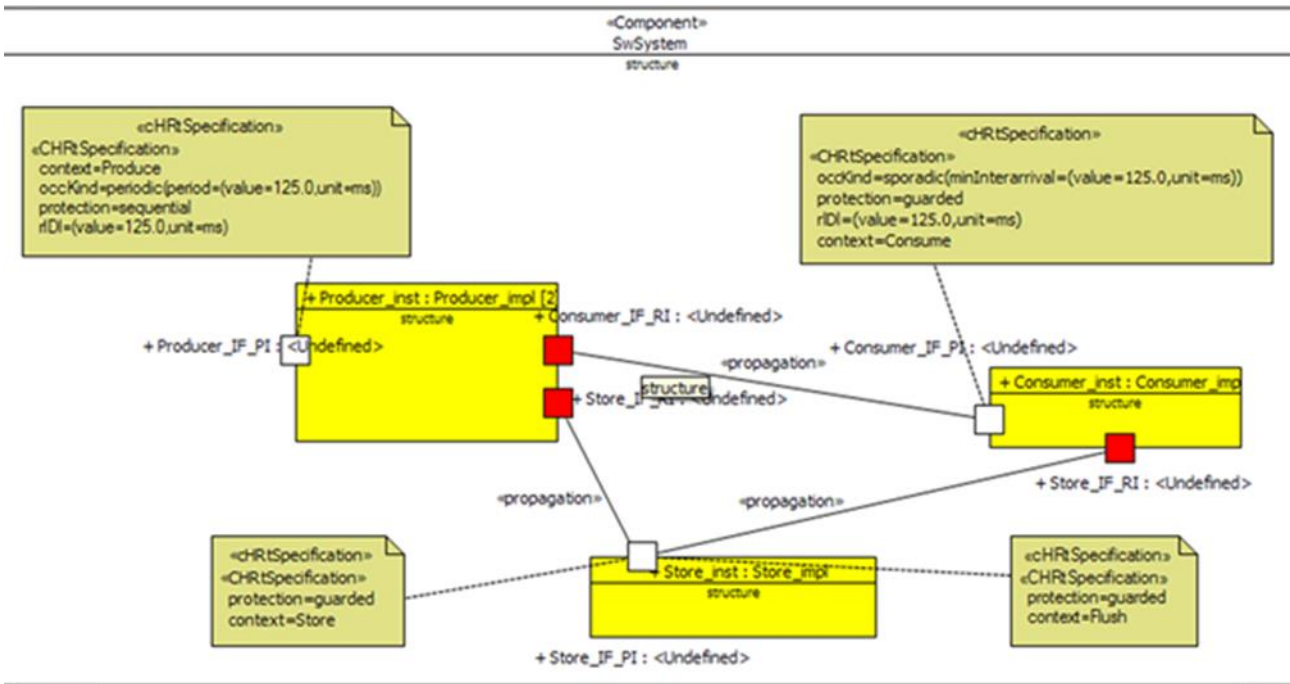


Figure 22: Working with the CHRTSpecification in the CSD (as instance view)

The properties of the CHRTSpecification can be edited through the CHES tab available on the Property view.

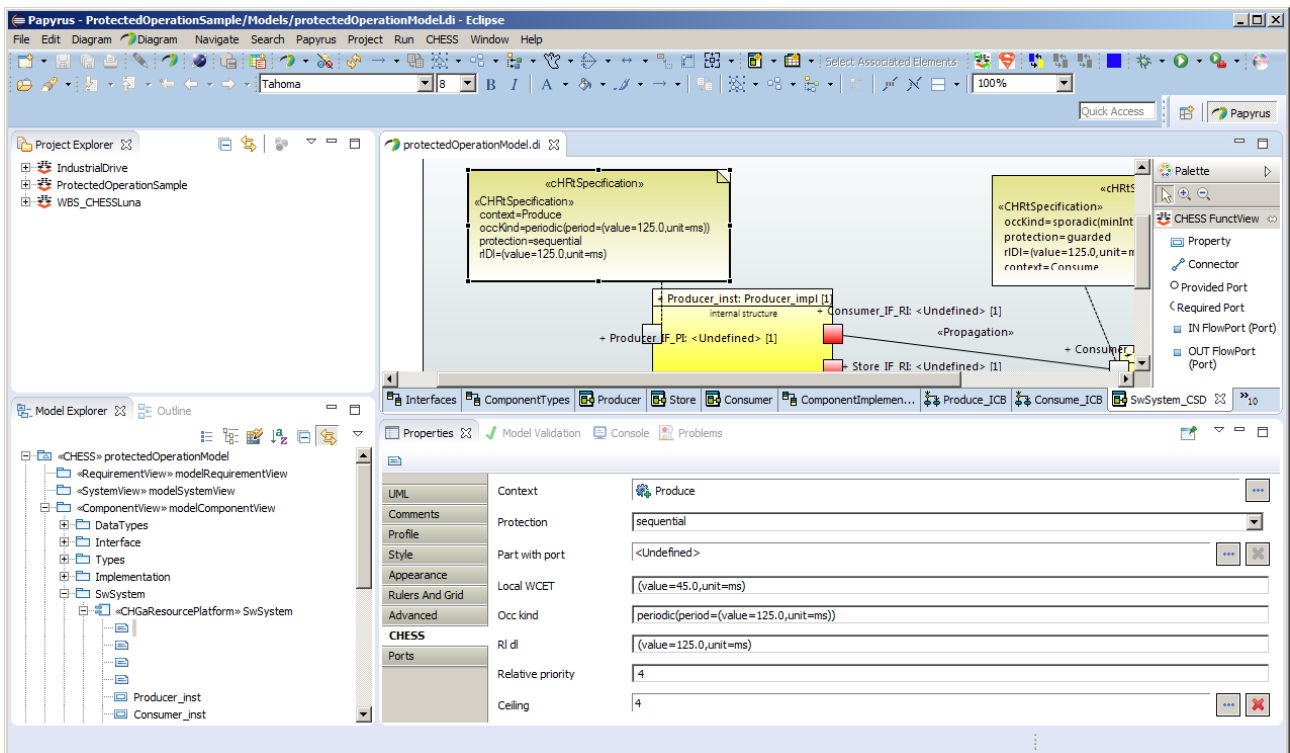


Figure 23: CHES tab for the CHRTSpecification

! Once the Real Time properties have been modelled as described above, the component instances need to be created (or simply updated) through the *Build Instances* Command.

5.10.1 Querying the timing properties

The real time properties attached to the generated InstanceSpecification can be retrieved in a dedicated tab-view by selecting the InstanceModel package (the one automatically generated by the Build Instances Command and stereotyped with <<CHGaResourcePlatform>>) owning the InstanceSpecifications, or selecting one instanceSpecification.

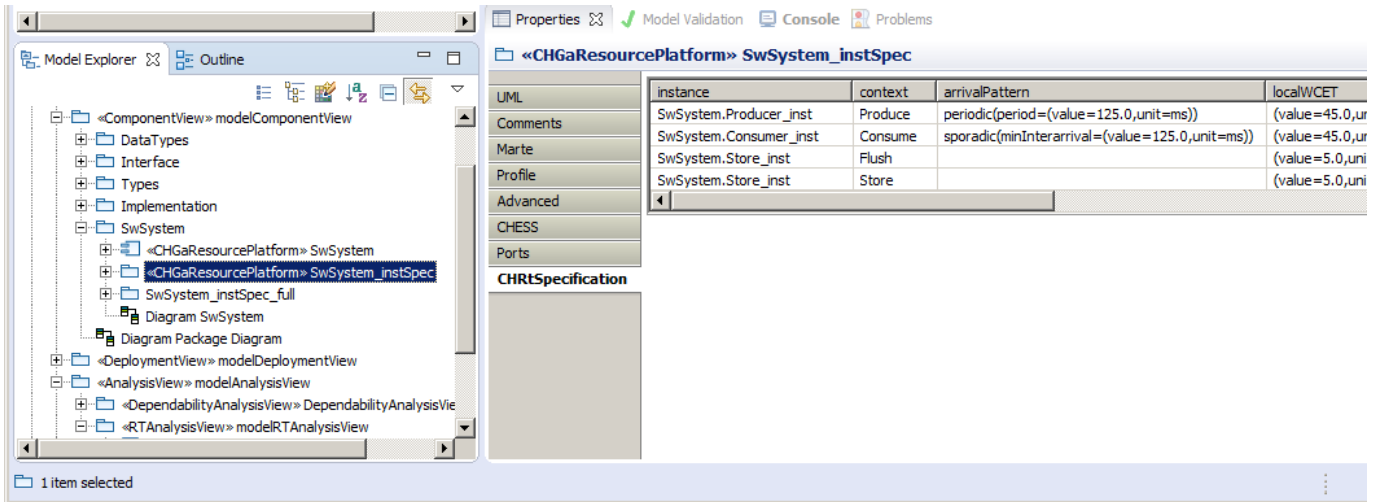


Figure 24: CHRTSpecification property tab

5.10.2 Instance View

The InstanceView can be activated by the Eclipse ShowView menu

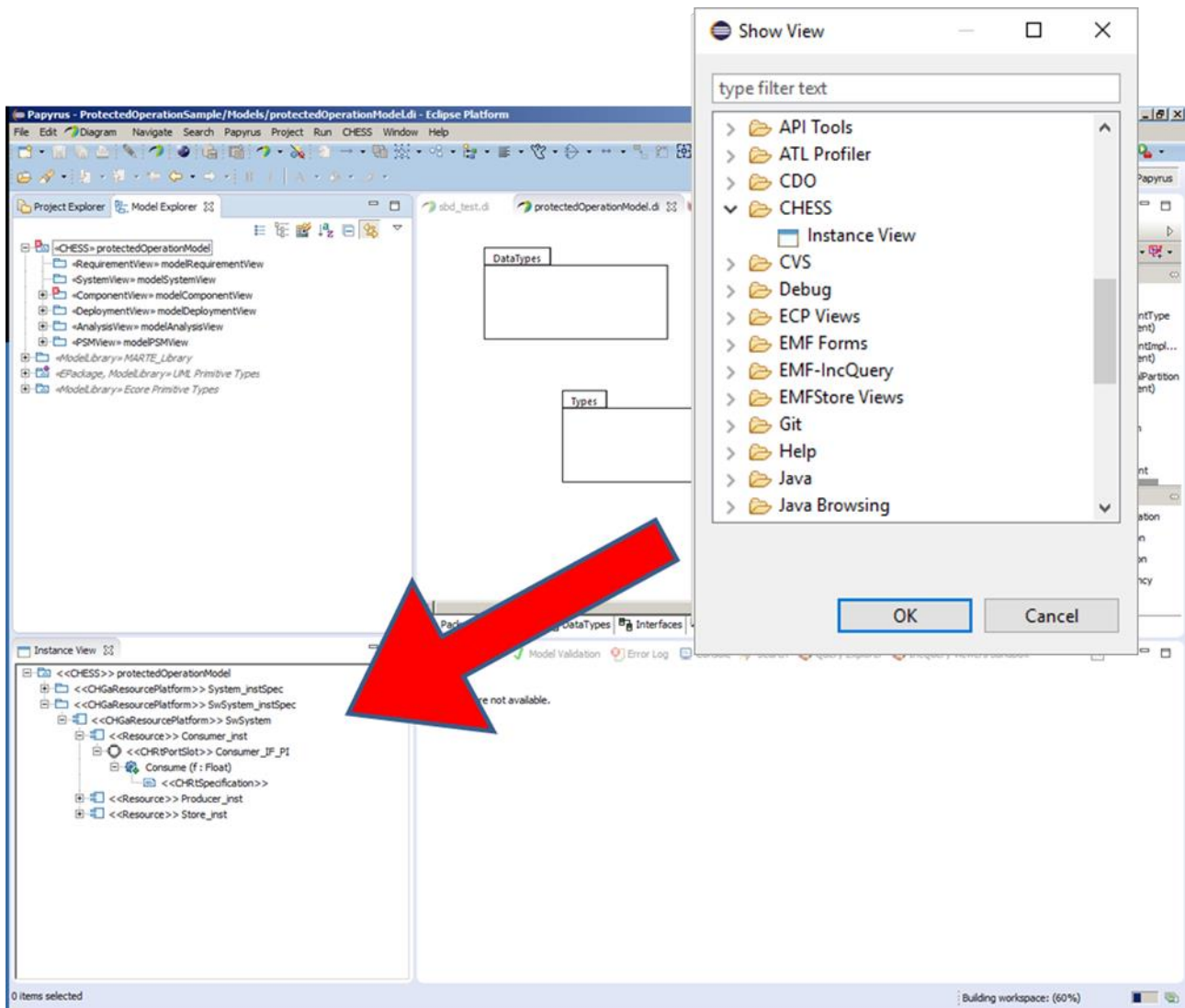


Figure 25: InstanceView

The InstanceView can be used to navigate the instances model, the latter as built with the *Build Model Instances* command. The InstanceView offers a customized view of the UML instances model, allowing to represent components instances hierarchies, component ports instances and referred operations through an intuitive representation Figure 26.

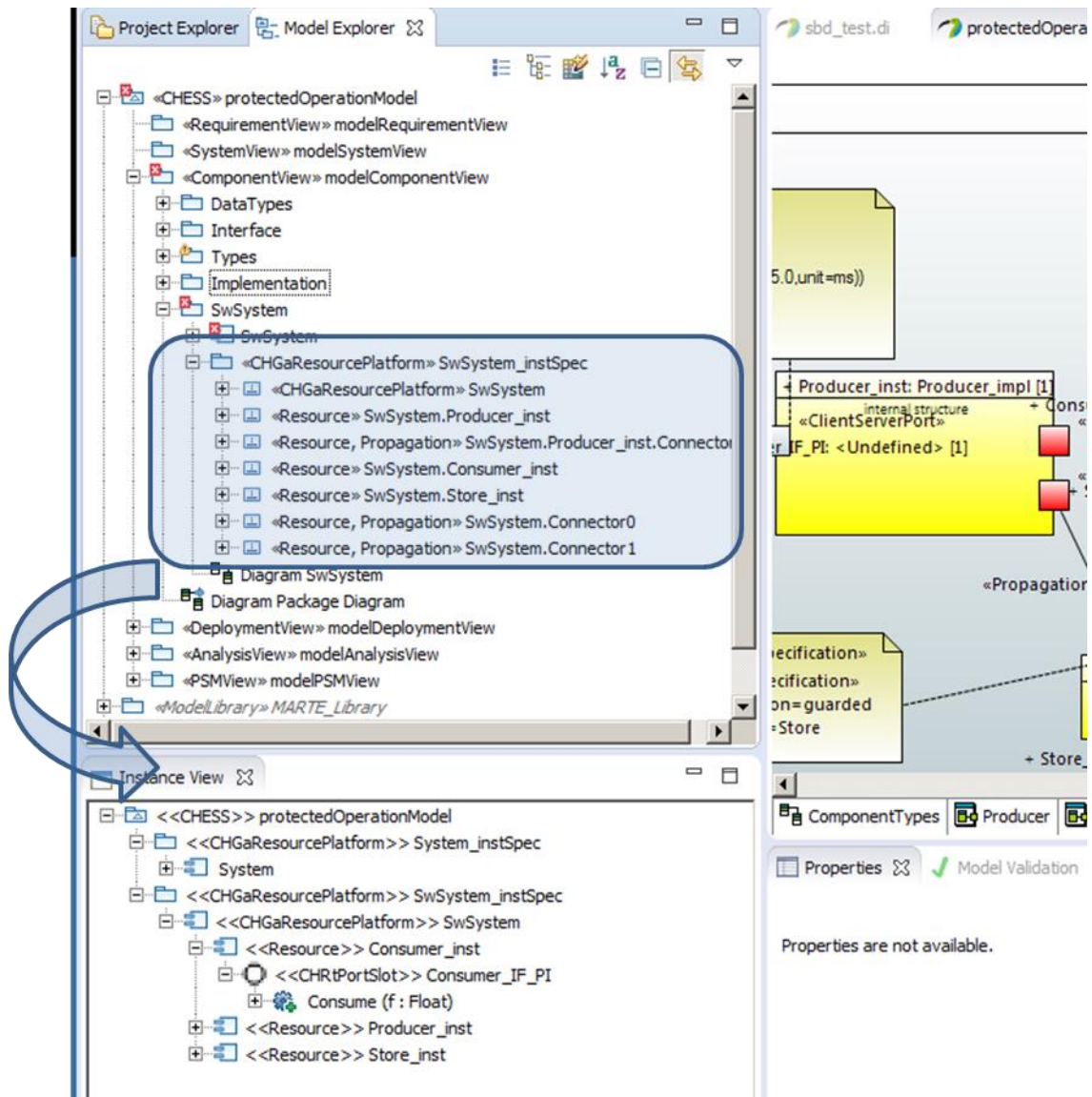


Figure 26: Reading with the InstanceView

Double click an item in the InstanceView to make it the current selection in Papyrus (Model Explorer and diagram, the latter if available) Figure 27.

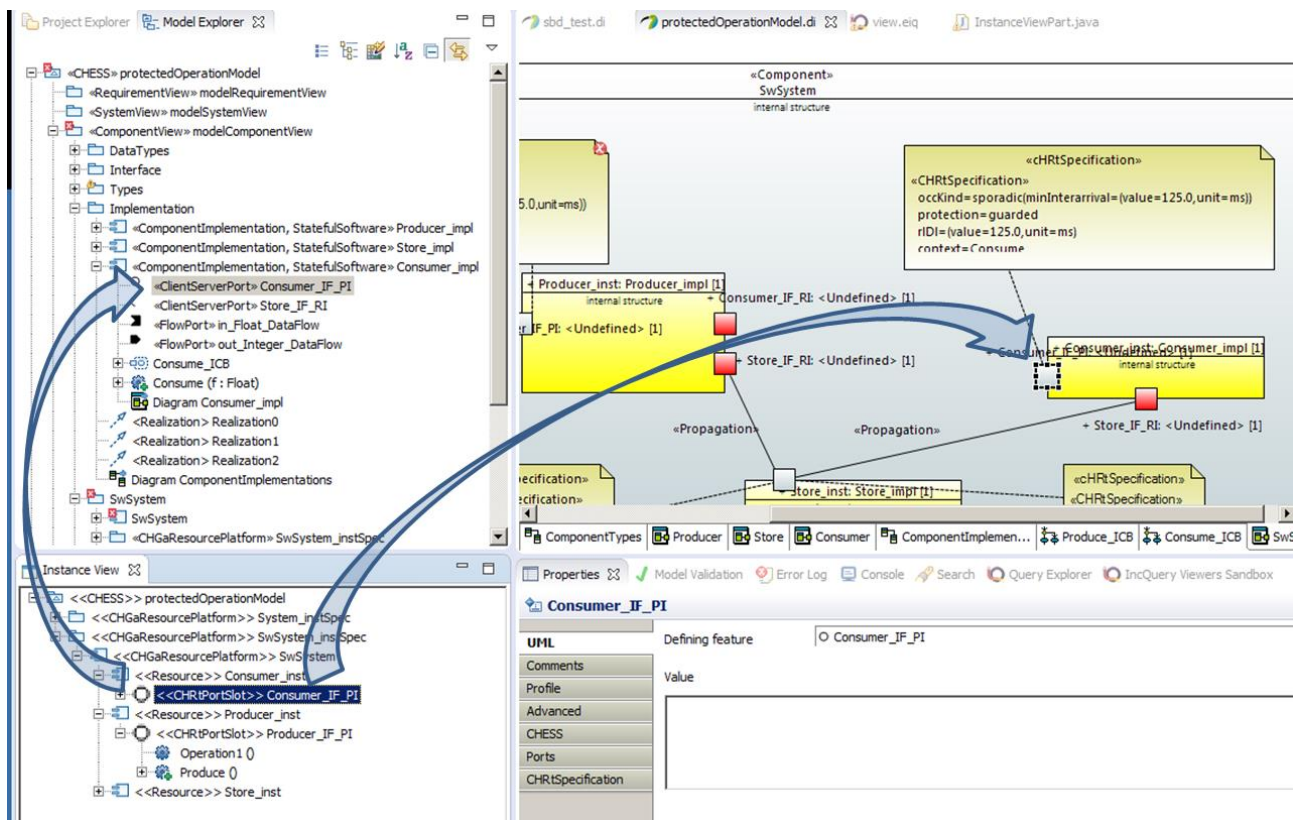


Figure 27: Reading with the InstanceView - hyperlinks

The InstanceView can be used to create CHRTSpecification for a given Operation owned by an Instance (right click on the Operation and select the kind of decoration to be applied (Figure 28); the operation can be public or private. A public operation is shown under the port instance which provides the interface where the operation itself is defined, while private operations appear under the component instance (see Func3 in the following figure).

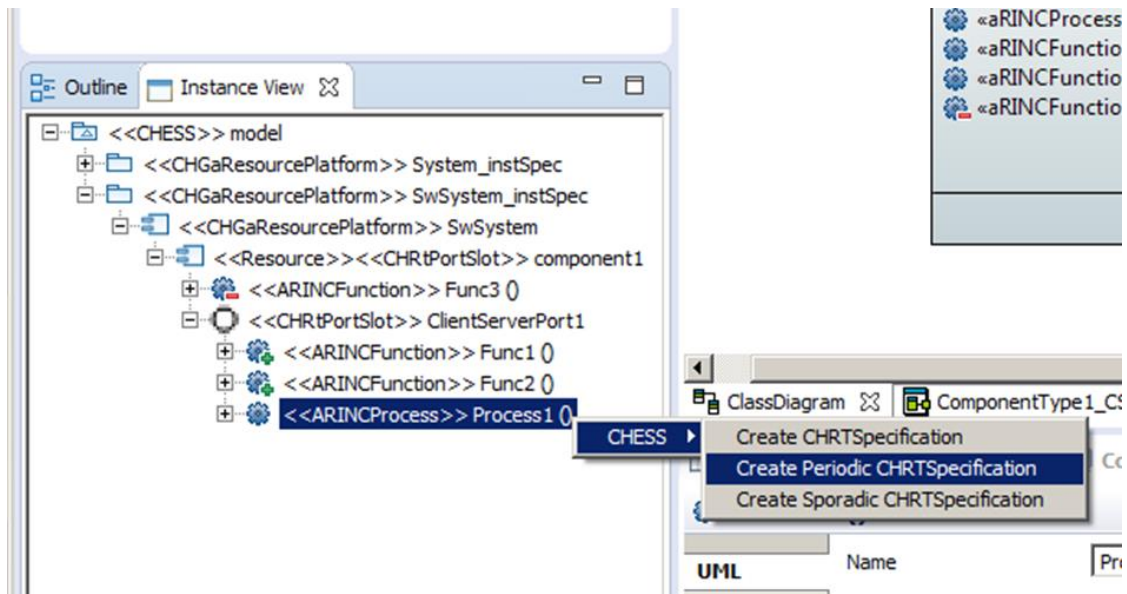


Figure 28: Creating with the InstanceView

A selection made in the InstanceView becomes the current selection for the other Eclipse views; in particular the Properties View is updated as well, to allow viewing/updating the properties for the selected item (Figure 29).

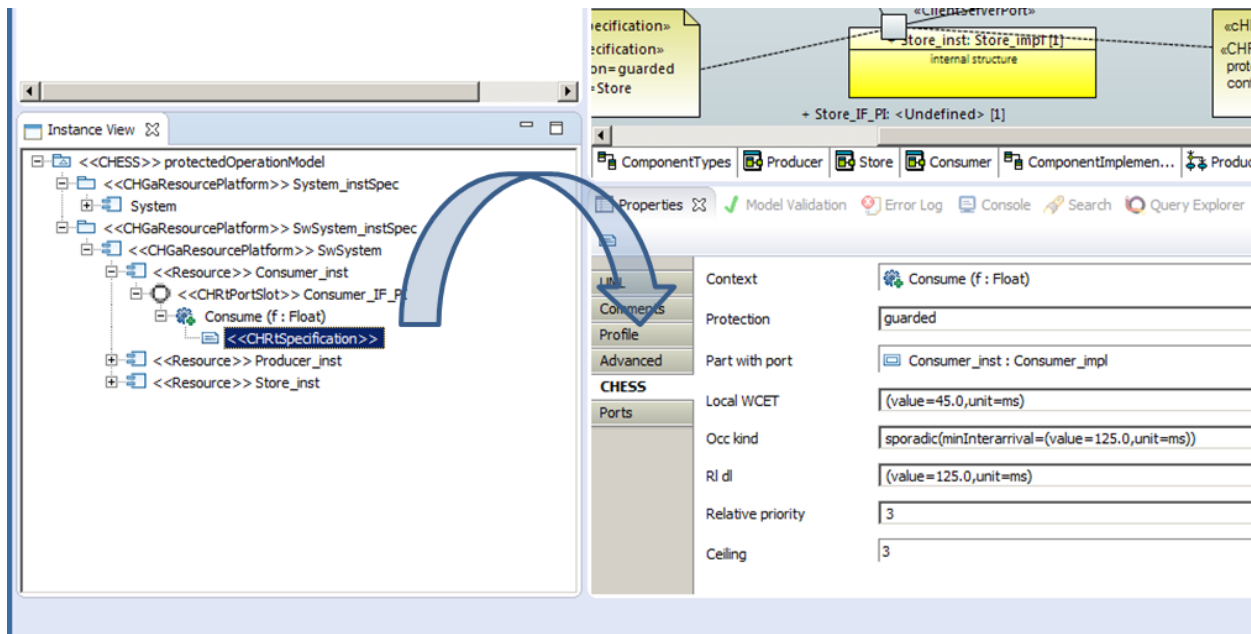


Figure 29: Updating with the InstanceView

6 Modelling the target platform and allocations

6.1 Modeling the HW entities

HW entities are initially defined at classifier level in the DeploymentView (or in its sub-packages) by creating UML::Component in a Class Diagram and applying stereotypes coming from a set composed of some defined in CHES and some others coming from the MARTE::HRM sub-profile, available from the Deployment View Palette (Figure 31) for Class Diagrams.

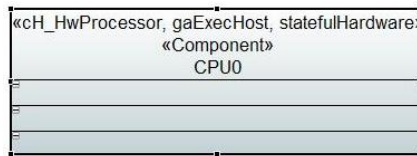


Figure 30 HW Types in a Class Diagram in the Deployment View

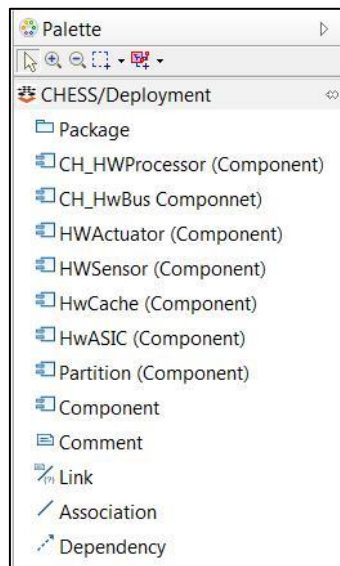


Figure 31 Deployment View Class Diagram Palette

Composite Structure diagrams for a given hardware component are used to model:

- hardware components instances and connectors owned by a given component

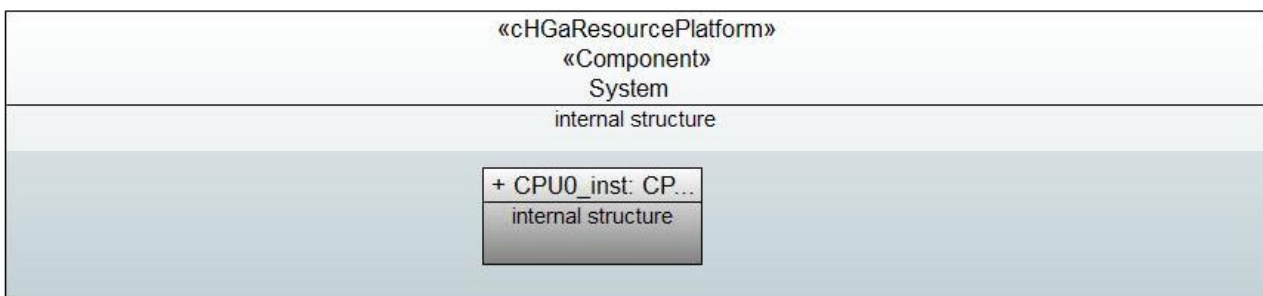


Figure 32: Composite Structure Diagram in the Deployment View

CompositeStructure diagrams in this view are also used by the tool to automatically build the hardware InstanceSpecifications, with the proper extra-functional information attached, through the **Build Instances command** from the CHES Context menu.

6.2 Defining the number of cores for a processor

It is possible to define the number of cores that constitute a processor, by assigning the desired value to the “nbCores” field of the component stereotyped as <<CH_HwProcessor>> in the Deployment View.

This operations is completed when the Build Instance command is invoked from the Context Menu (see Figure 33) and the actual HwSystem Instance Specification is created with its defined cores.

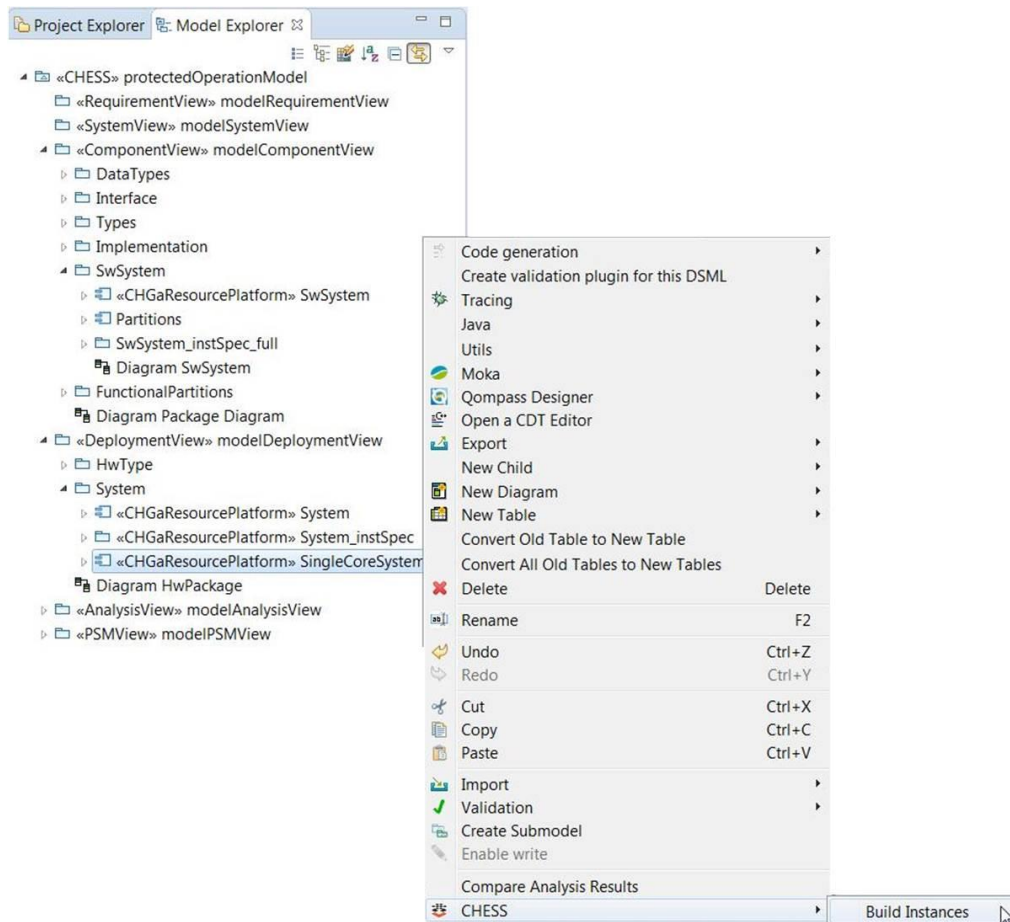


Figure 33 Build instances for HW System

Notice: if the user decreases the number of cores that constitute a processor, active verification is carried out to request user’s confirmation to delete all assignments (Partition to Core Assignment and Task to Core Assignment) to the processor whose number of cores has been decreased.

6.3 Allocating SW instances to HW instances

CHES allows specifying SW to HW deployment by using the Assign MARTE stereotype.

6.3.1 Multicore Support Wizard

The wizard presented in this section allows to model the SW to HW instances allocation using a dedicated wizard; the creation of the Assign entities in the model is done automatically by the tool. The Assign can be showed in a given diagram by drag&drop them from the ModelExplorer to the diagram itself.

6.3.1.1 Assign Components to Cores

In order to assign components to cores: under the “CHES” command button, select “Multicore Support” and then “Assign Components to Cores”. This will open a form that allows to assign the Component Instances (listed on the left side) to the processor’s cores (on the right side), as illustrated in Figure 34.

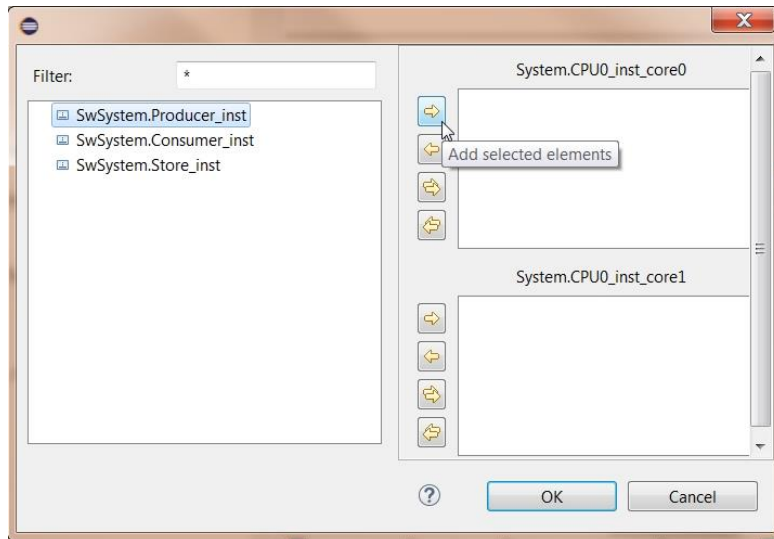


Figure 34 Assign Components to Cores Wizard

6.3.1.2 Assign Tasks to Cores

In order to assign tasks to cores, under the “CHES” command button, select “Multicore Support” and then “Assign Tasks to Cores”. This will open a form that allows to assign the tasks (listed on the left side) to the available instances of processor’s cores (on the right side), as illustrated in Figure 56Figure 55.

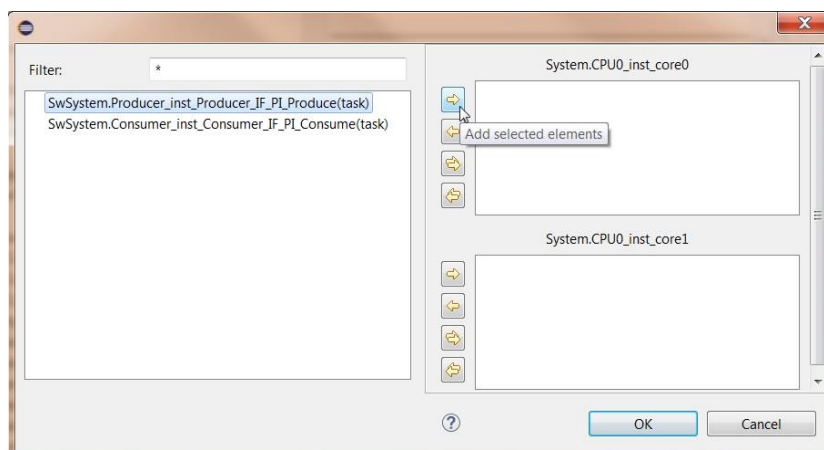


Figure 35 Assign Tasks to Cores Wizard

6.3.1.3 Generate Task to Core Assignments

In order to generate task to core assignments, under the “CHES” command button, select “Multicore Support” and then “Generate Task to Core Assignments”. Task to core assignments are generated according to the Reduction to Uniprocessor algorithm.

6.3.2 Manually creating the Assign entities

Assign entity can be created in the DeploymentView, e.g. by using the composite structure diagram.

Assign comes with *from* and *to* fields which can be set through the MARTE or Profile tab of the Eclipse Properties view. To specify the allocation of a SW instance to an HW instance:

- *from* must be set with the proper SW instance (the one created with the Build Model Instances command)
- *to* must be set the proper HW instance (the one created with the Build Model Instances command)

Next figure shows an example of Assign used to model the allocation of the instances previously showed; note that the Build Model Instance command has been invoked on the System entity in order to have the System_CPU0_inst instance available to be referred in the *Assign.to* field.

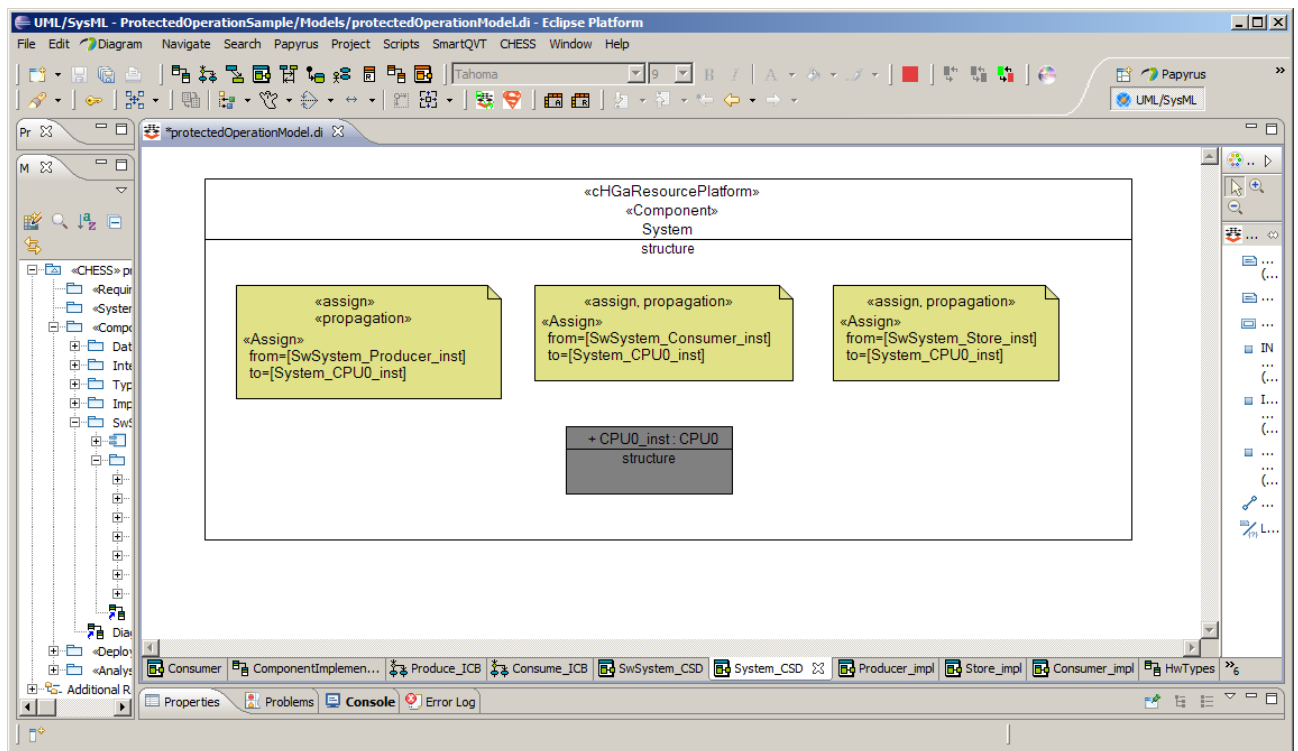


Figure 36: Modeling the SW to HW allocation

6.3.3 Specifying the Operational Mode

In case of a system with different operational modes it is possible to define different software to hardware allocations scenarios, one for each operational mode. In this case the MARTE Assign, already created to model allocations, can be constrained by the MARTE NFPCConstraint construct,

where the latter allow specifying the mode on which the given Assign has to apply. For example, in Figure 5 the allocation of the component instances to the processing resource CPU0 for only the *NormalMode* operational mode is modelled.

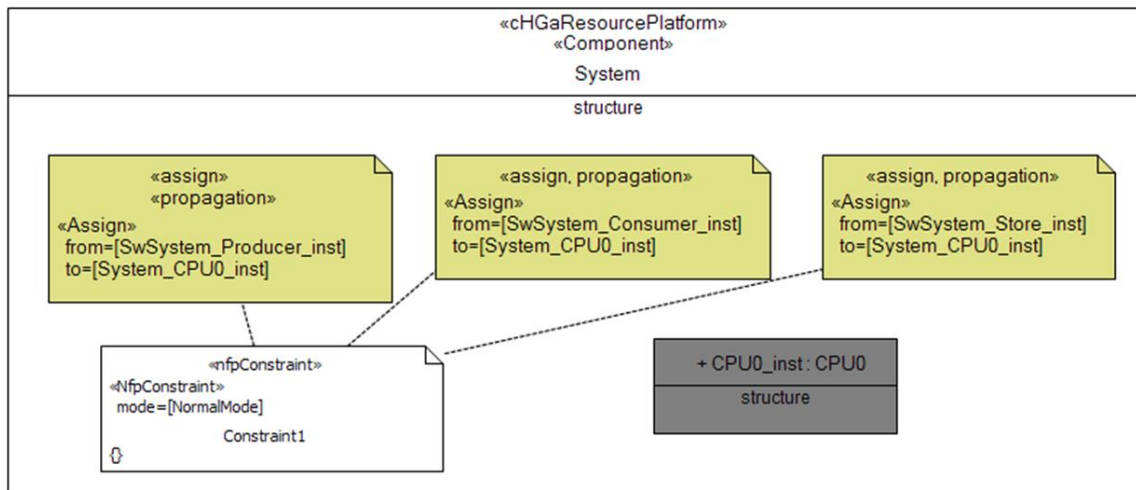


Figure 37: modelling Allocations constrained to OperationalModes

7 Model validation

Different types of model validation are provided by the CHES toolset for what regards the definition of the software system and target platform.

In order to perform model validation starting from a selected entity and all its owned ones, from the Papyrus Model Explorer select the Model or the entity on which the validation must be performed, right click and choose:

- Validation-> CHES->Validate core constraints->: performs checks to enforce the CHES methodology constraints (including specific preconditions as required by the schedulability analysis).

7.1 Configuring the CHES model validation features

It is possible check and configure which model validation features are enabled in CHES by selecting from the Eclipse commands bar “Window”->“Preferences”->“Model Validation”->“Constraints”->“CHES Model Constraints”, and finally selecting the constraints to be enabled.

See also section 11.2 for more information about CHES constraints.

8 Running Schedulability Analysis

To run the schedulability analysis the SaAnalysisContext entity needs to be created in the RTAnalysisView:

- create a class diagram in the RTAnalysisView
- create a SaAnalysisContext
- set the SaAnalysisContext.platform attribute to

- the <CHGaResourcePlatform> Package owning the component instances, as generated by the Build Instance command and
- the <CHGaResourcePlatform> Package owning the HW component instances where the SW is deployed, as generated by the Build Instance command.

Notice that multiple SaAnalysisContexts can be created in order to perform schedulability analysis on different deployment configurations for comparison.

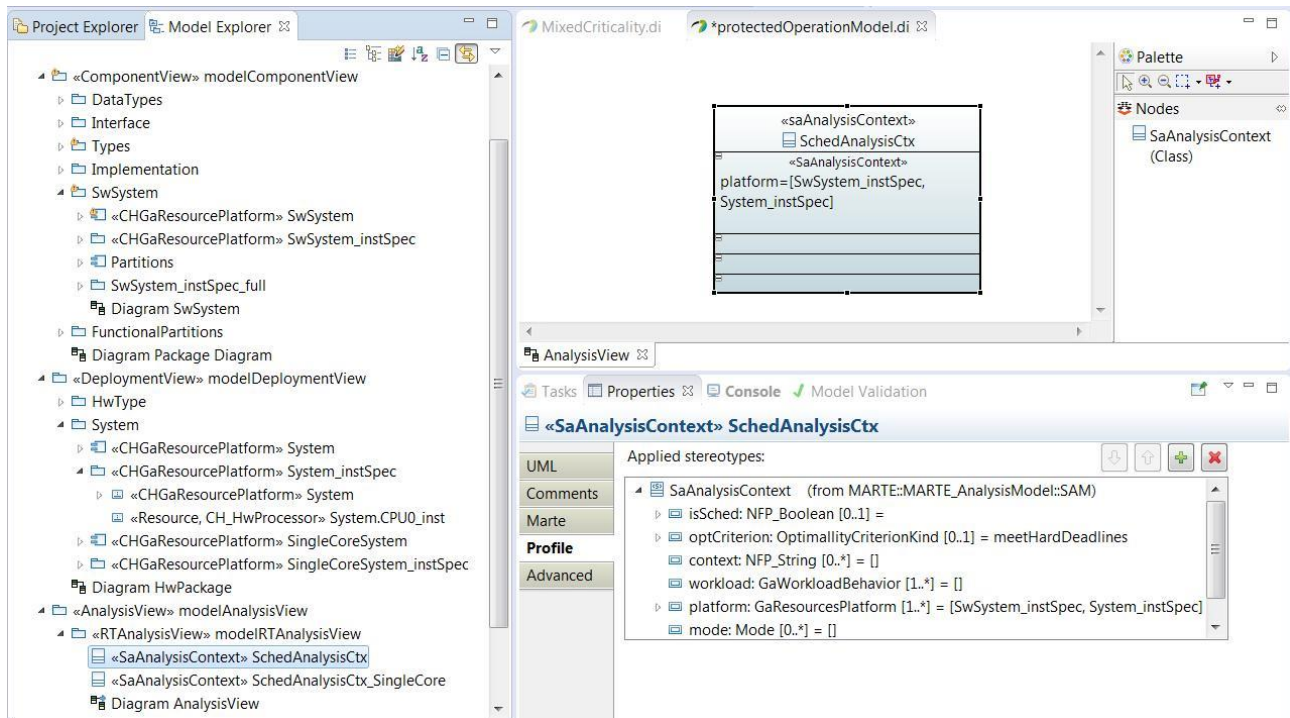


Figure 38: Schedulability Analysis Context

Then the Schedulability Analysis may be invoked from the CHES command menu, by selecting CHES->Analysis->Real Time->Schedulability, as depicted in Figure 39.

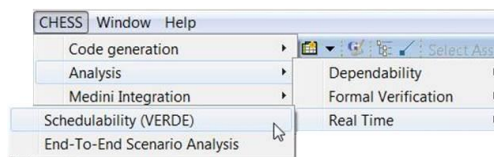


Figure 39 CHES Schedulability analysis

If multiple anasysisContexts are present in the model, the user is prompted to specify the one to be considered for the performing the analysis (see Figure 40).

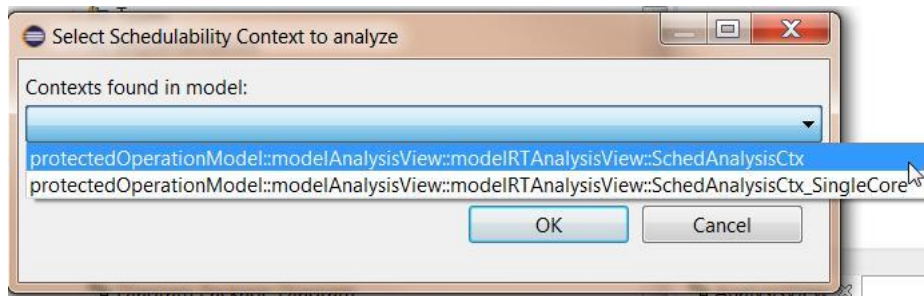


Figure 40 CHES Select Analysis Context window

Wait while the analysis is executed (may take some time)

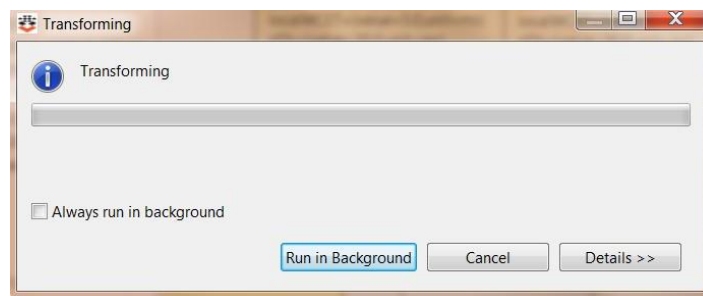


Figure 41 Analysis in execution

Finally the results will be visualized in a pop-up window as illustrated in Figure 42.

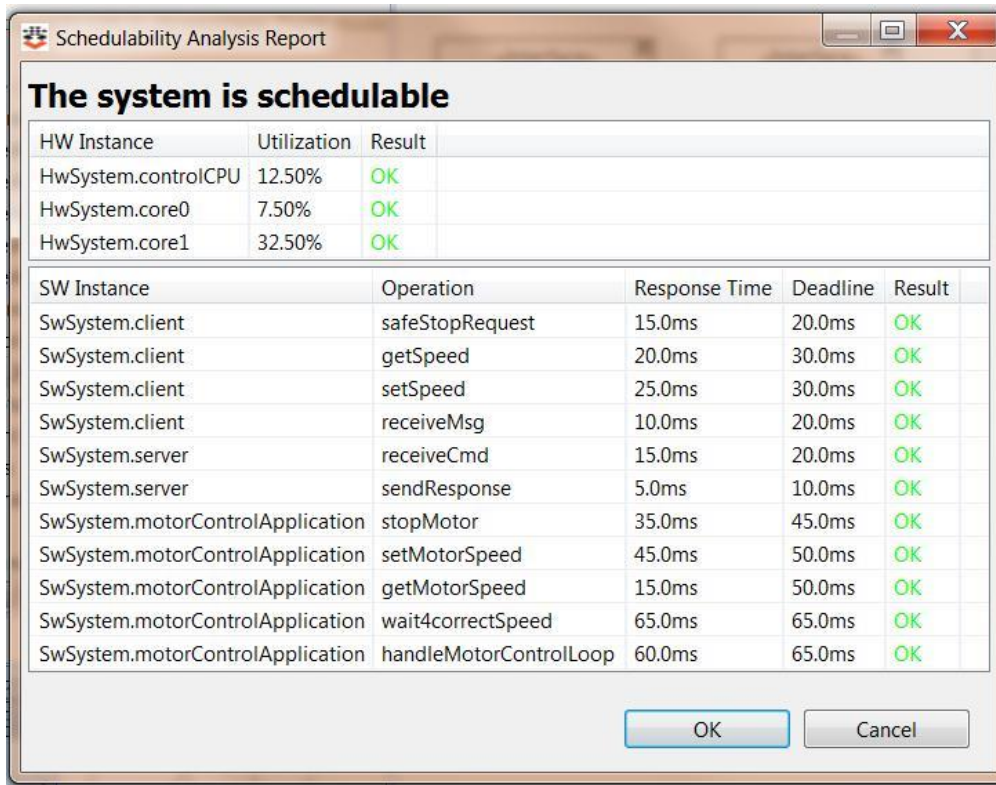


Figure 42 Schedulability analysis results

The first section of the results window represents the utilization of HW resources, while the second section details, for each operation, the SW instance that executes it, the calculated response time and the defined deadline. The Result field indicates whether the deadline will be met or not.

Analysis results can also be showed at any time by selecting the AnalysisContext in the model explorer and then by selecting the AnalysisContext in the Properties view.

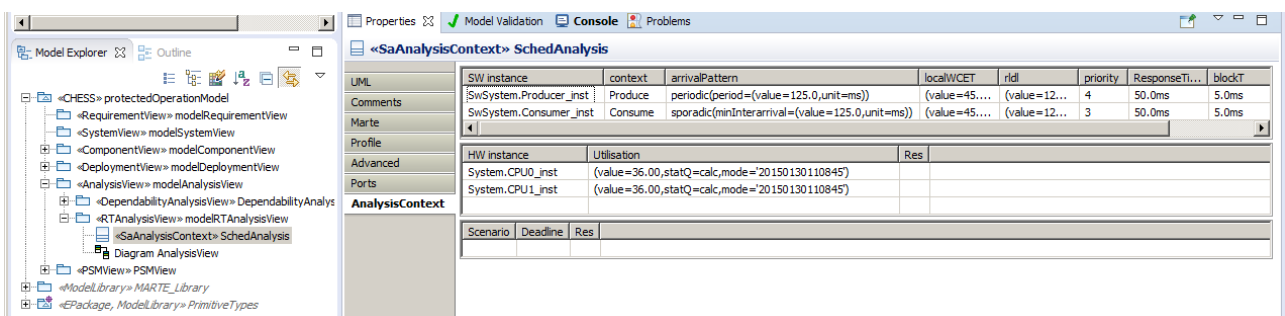


Figure 43: Analysis context tab

It is possible to compare the results obtained by the analysis of two different deployment configurations: once both analysis have been run, selecting the two SaAnalysisContexts and using the Context Menu Command "Compare Analysis Results" as depicted in Figure 44.

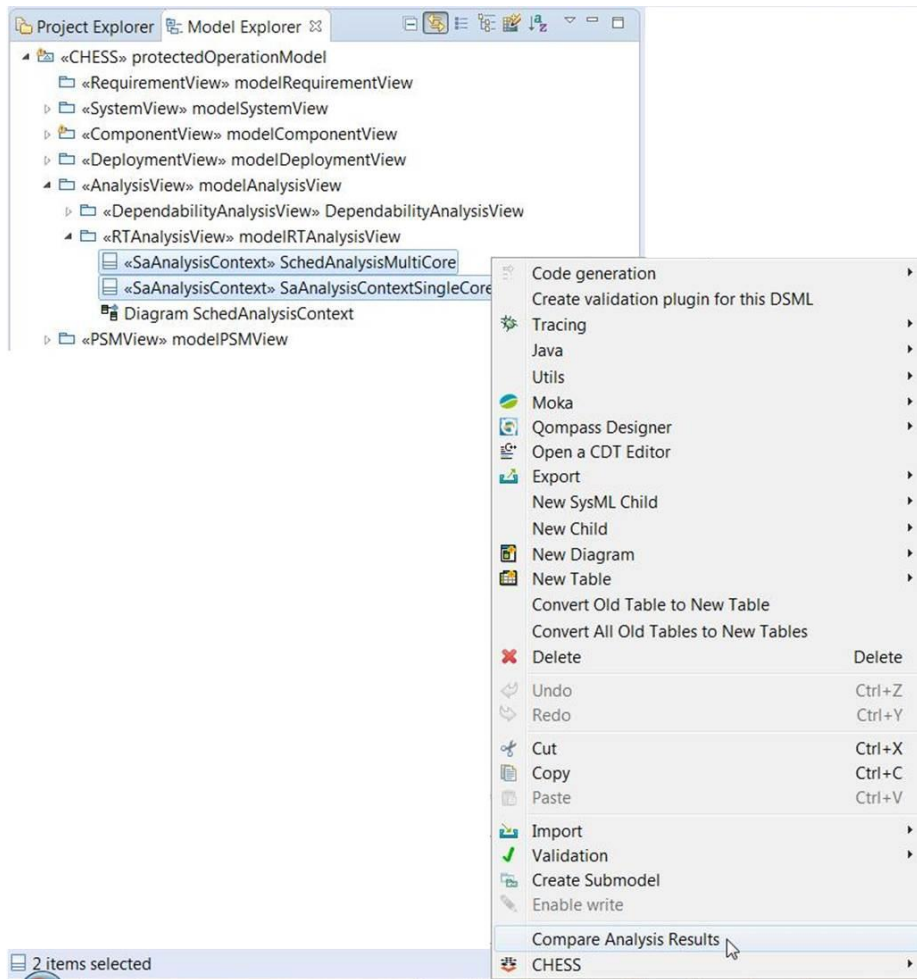


Figure 44 CHES Compare Analysis Results Command

The analysis results obtained for the different deployments are displayed side by side as illustrated in Figure 45.

The 'Compare analysis results' dialog box displays a table with the following data:

HW Instance	Utilization	Result	Analysis Context
System.CPU0_inst_core0	36.00%	OK	SchedAnalysisMultiCore (1)
System.CPU0_inst_core1	36.00%	OK	SchedAnalysisMultiCore (1)
System_SingleCore.singlecore	72.00%	OK	SaAnalysisContextSingleCore (2)

SW Instance	Operation	Resp. T. (1)	Resp. T. (2)	Block. T. (1)	Block. T. (2)	Sched. (1)	Sched. (2)
SwSystem.Consumer_inst	Consume	50.0ms	90.0ms	5.0ms	0.0ms	OK	OK
SwSystem.Producer_inst	Produce	50.0ms	50.0ms	5.0ms	5.0ms	OK	OK

At the bottom of the dialog, there are 'OK' and 'Cancel' buttons.

Figure 45 CHES Analysis Results Comparison

9 End2End Response Time Analysis

The process for the End2End response time analysis, briefly described in the rest of the paragraph, follows the MARTE modeling approach and allows to model and analyze different End2End scenarios.

In order to perform End2End response time analysis on a scenario that has been modeled in a Sequence Diagram in the Component View, the following modeling steps must be carried out in the Analysis View:

1. Under RTAnalysisView, create a Package (e.g. E2E_Analysis_Package in this example) to contain End2End necessary model elements
2. Apply the <<GQAM>> and <<SAM>> MARTE profiles to the newly created package (Figure 59): they are respectively the Generic Quantitative Analysis Modeling and Schedulability Analysis Modeling MARTE packages.
3. Create an Activity Diagram (e.g. E2E_AnalysisActivity) inside the previously created package (E2E_Analysis_Package). This represents the activity of the End2End analysis
4. Apply the <<GaWorkloadBehavior>> and <<SaEndtoEndFlow>> MARTE stereotypes to the newly created Activity.

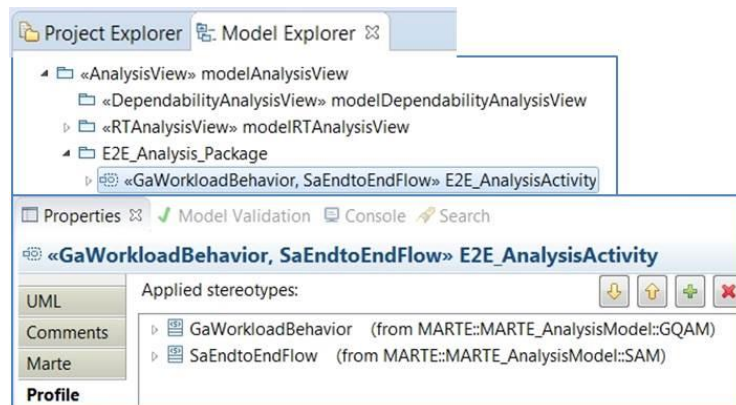


Figure 46 End2End Activity in End2End Analysis Package

5. Specify the end2End Deadline in <<SaEndtoEndFlow>> properties. This represents the timing requirements to be met

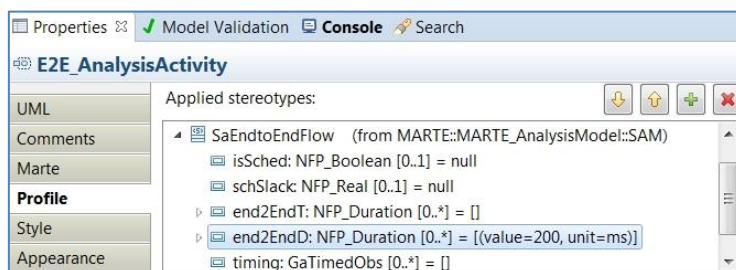


Figure 47 Setting the Activity properties for End2End Analysis

6. To model the analysis activity, the simplest case is to model a Call Behavior Action element, in the newly created Activity Diagram, and specify its Behavior referring to the Sequence Diagram that represents the scenario on which the End2End analysis will be executed (e.g. E2E_SafeMotorStop_Scenario).

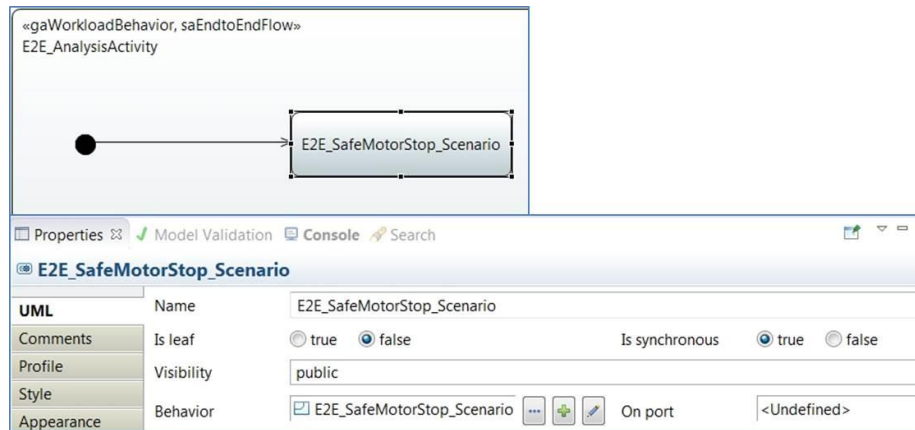


Figure 48 Activity Diagram for End2End Analysis

7. Create a Class Diagram and there model a Class to represent the analysis context (e.g. E2E_AnalysisContext)
8. Stereotype the newly created Class as <<SaAnalysisContext>>
9. Set the <<SaAnalysisContext>> Workload property to refer to the analysis Activity modeled above (e.g. E2E_AnalysisActivity) (Figure 49)

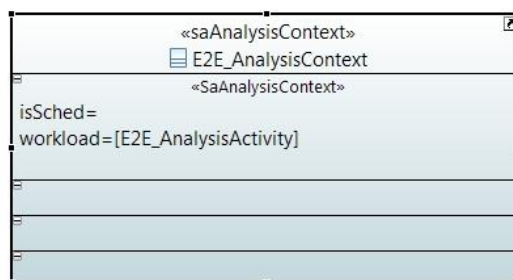


Figure 49 End2End Analysis Context

10. Invoke the End2End Analysis from the CHES command menu (Figure 50)

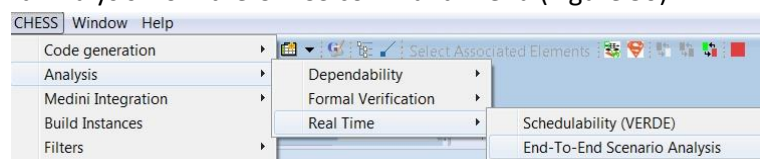


Figure 50 End2End Analysis Command

11. Select the context for the End2End Analysis. The dialog offers a choice between all the End2End Analysis Contexts found in the CHES model (Figure 51)

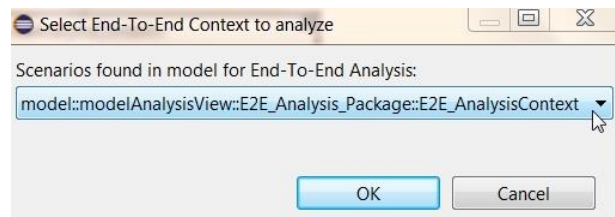


Figure 51 End2End Analysis Context selection

12. Wait while the analysis is executed (may take some time)
13. Result appears in a pop-up window (Figure 52) where the first section (HW Resources) is described in 7). The second one details the result of the End-To-End Scenario analyzed. The last reports the response time of those operations in the model not involved in the End-To-End Scenario under analysis (as in 7)

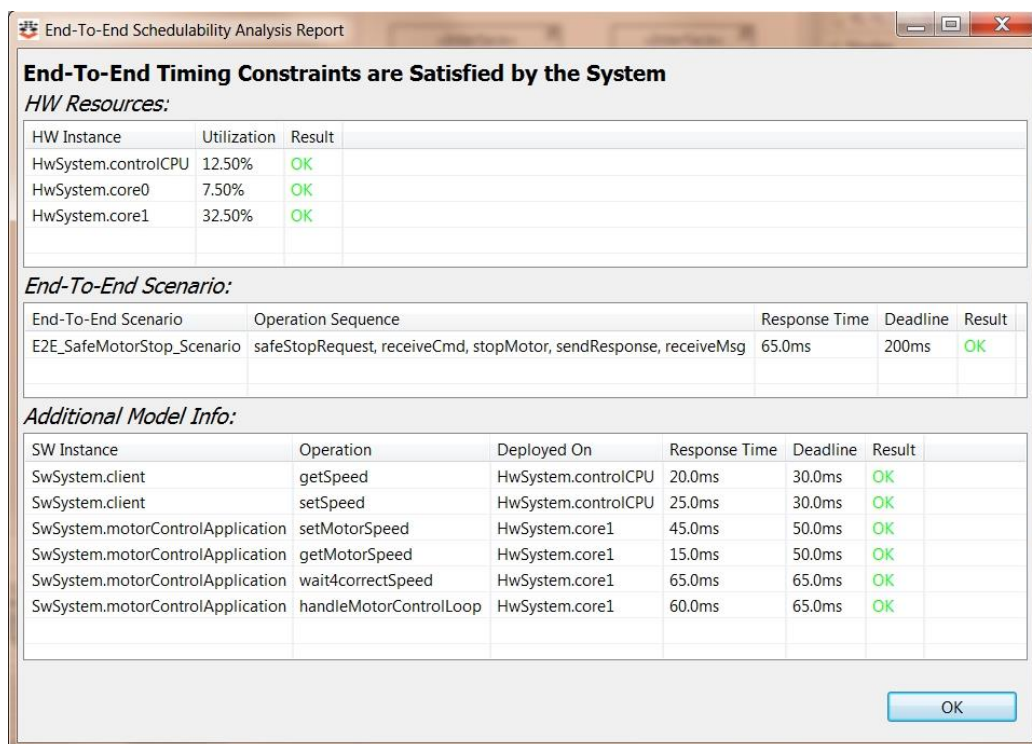


Figure 52 End2End Analysis Result pop-up window

14. The end2end Response Time property in the Activity <<SaEndtoEndFlow>> is updated with the analysis results (Figure 53)

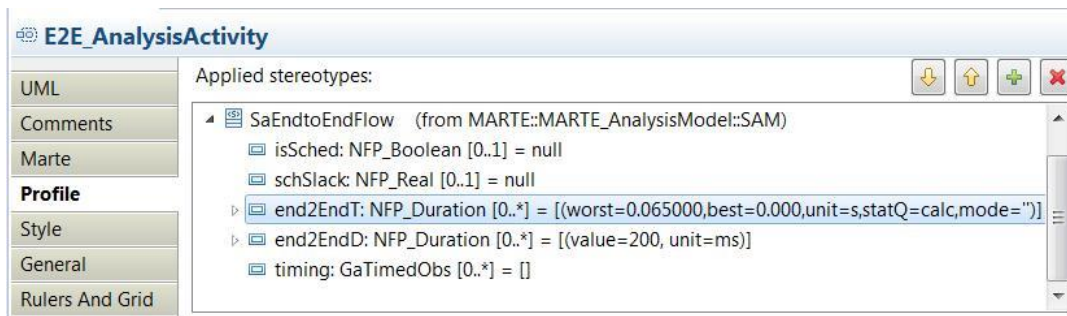


Figure 53 End2End Analysis results in Activity diagram SaEnd2EndFlow properties

15. Also the Context is updated with the analysis results (Figure 54)

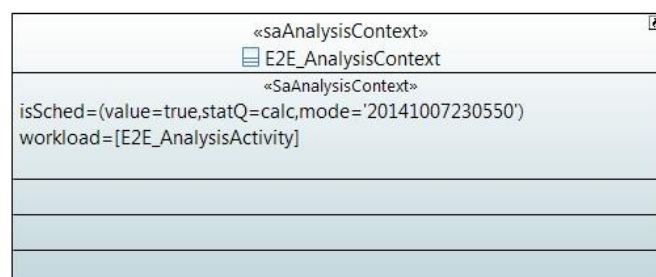


Figure 54 End2End Context updated with results

End2End response time analysis results are automatically saved as a textual file in the CHES project (in the End-To-End_analysis_results folder).

Moreover as for the schedulability analysis, analysis results can be retrieved at any time by selecting the AnalysisContext in the model explorer or diagram and then selecting the AnalysisContext tab in the Property View (see Figure 38).

10 Domain specific modeling

This section introduces some domain specific modelling features provided by CHES. ²

10.1 Avionics³

10.1.1 Creating functional partitions

IMA Functional partitions can be modeled in the Component View. In order to model functional partitions, first of all it is necessary to create a Package named “FunctionalPartitions” inside the Component View. To create a functional partition, simply create a Component in the FunctionalPartitions Package and then

² In addition to the support for Avionics, CHES provides support for Automotive, in particular by allowing integrations with AUTOSAR models. For more information please contact the CHES development team (e.g. via chess-dev@eclipse.org).

³ In addition to the modelling support for Avionics, timing analysis of IMA architectures is also available in CHES. For more information please contact the CHES development team (e.g. via chess-dev@eclipse.org).

apply the Stereotype <<CHESS::ComponentModel::FunctionalPartition>>. The functional partitions must then be instantiated as part properties in the <<CHGaResourcePlatform>> Component that represents the SWSYSTEM in the Component View.

These operations are completed when the Build Instance command is invoked and the actual SwSystem Instance Specification is created with its defined functional partitions.

Notice: if the user deletes a functional partition that is part of the <<CHGaResourcePlatform>> Component that represents the SWSYSTEM in the Component View, active verification is carried out to request user's confirmation to delete all assignments that involve the functional partition that has been deleted either in the To field (Component to Partition Assignment), or in the From field (Partition to ProcessorAssignment).

10.1.2 IMA Partitions Support Wizard

10.1.2.1 Assign Components to Partitions

In order to assign components to functional partitions: under the "CHESS" command button, select "IMA Partitions Support" and then "Assign Components to Partitions". This will open a form that allows to assign the Component Instances (listed on the left side) to the functional partitions (on the right side), as illustrated in Figure 55.

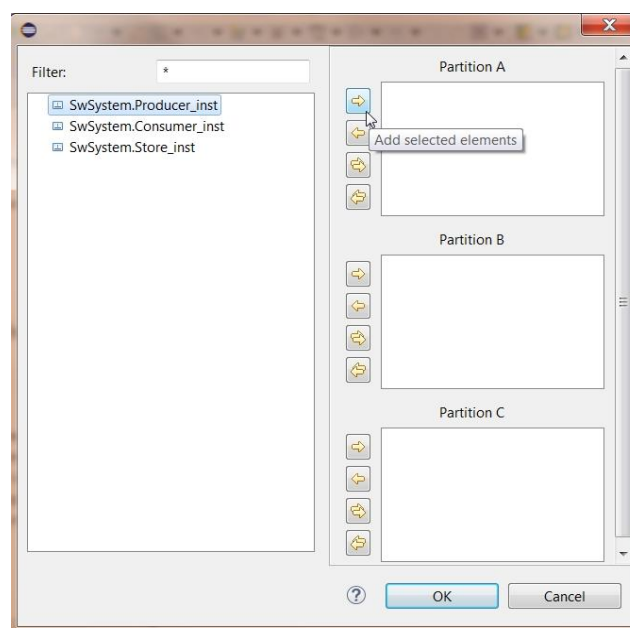


Figure 55 Assign Components to Partitions Wizard

⚠ The current implementation does not support resource sharing between partitions.

10.1.2.2 Assign Partitions to Cores

In order to assign functional partitions to cores, under the "CHESS" command button, select "IMA Partitions Support" and then "Assign Partitions to Cores". This will open a form that allows to assign the functional partitions (listed on the left side) to the available instances of processor's cores (on the right side), as illustrated in Figure 56Figure 55.

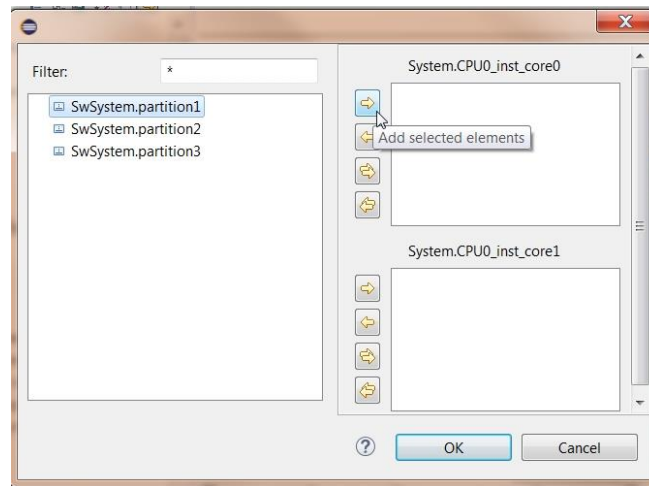


Figure 56 Assign Partitions to Cores Wizard

10.1.3 ARINC Processes

This section provides an example of application of the ARINC related stereotypes.

Figure 57 shows the usage of the ARINCComponentImpl stereotype applied to a <<ComponentImplementation>> Component owning one <<ARINCProcess>> and three <<ARINCFunction>> public operations.

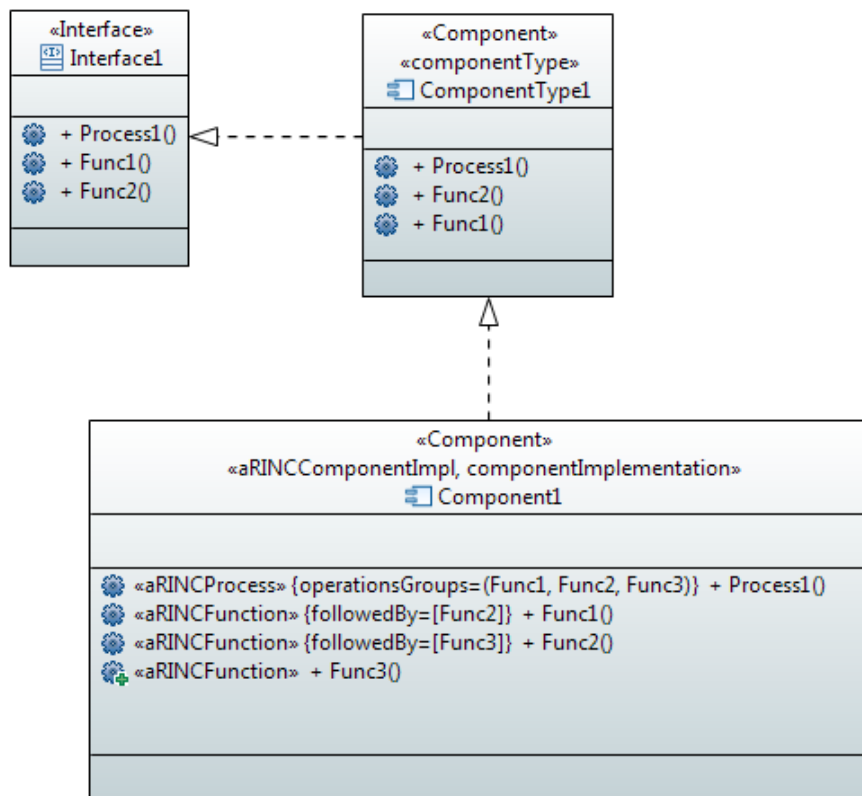


Figure 57: ARINCComponentImpl

Figure 58 below shows how additional real time properties can be set for the ARINCProcess and the ARINCFunction operations for a given ARINCComponentImpl component instance, according to the CHES profile for ARINC; in particular real time properties are provided through the CHRTSpecification stereotype.

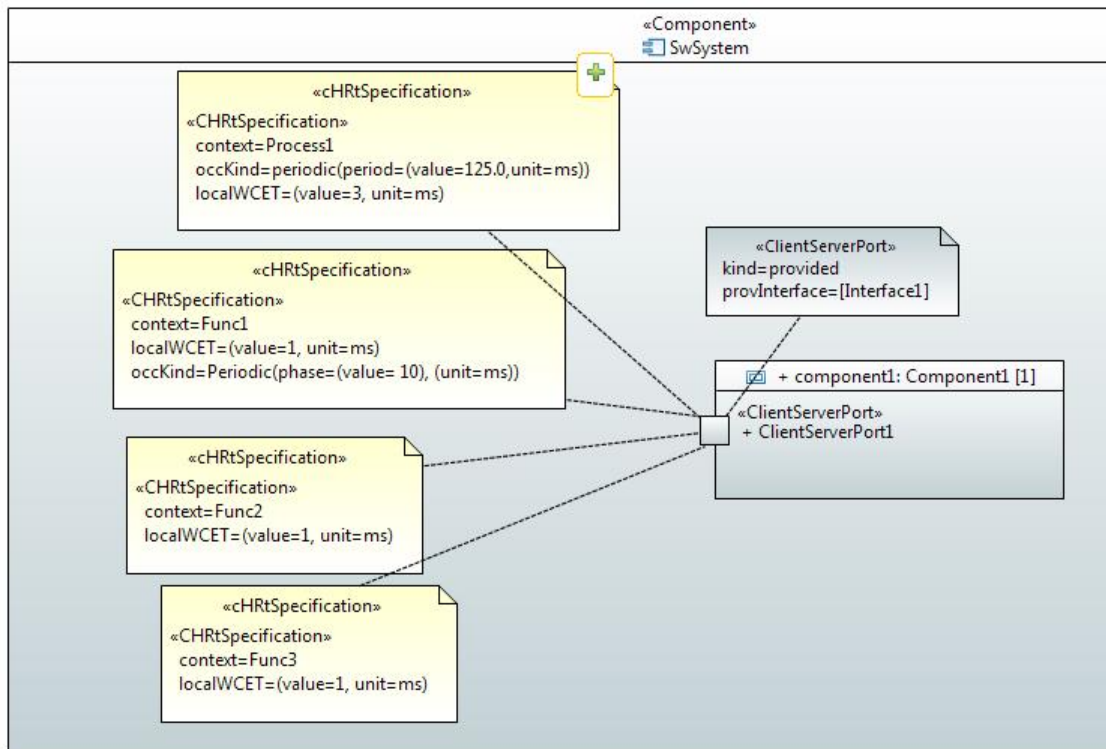


Figure 58: providing real time properties for ARINCProcess and ARINCFunction operations

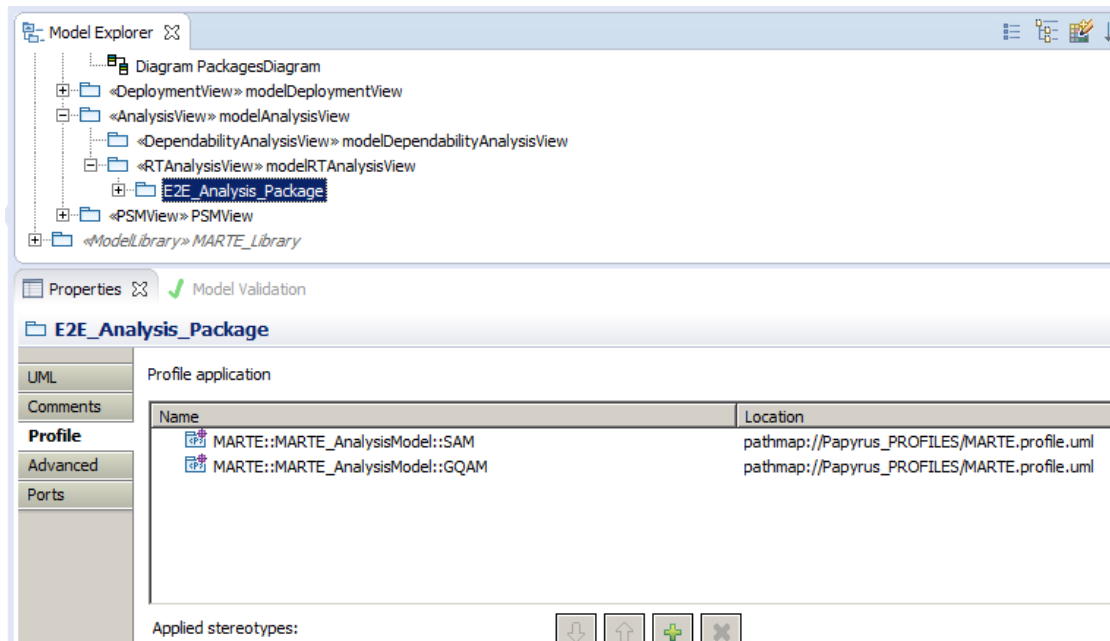


Figure 59 End2End Analysis Package

11 Appendix

11.1 Tutorial: the CHES process - protected operation example

11.1.1 Create data types

Go to the Component View, then right click on Component View and select New Child->Package to create a package called “DataTypes”.

Right click on the newly created package and select New Diagram->Class Diagram to create a new Class Diagram, then use the palette to create the needed types.

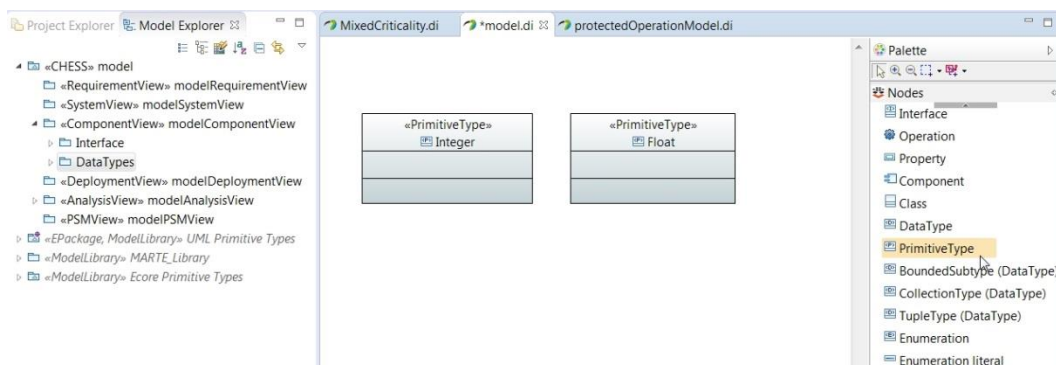


Figure 60 Creating data types

11.1.2 Create interfaces

Go to the Component View, then right click on Component View and select New Child->Package to create a package called “Interface”.

Create the Interfaces: right click on the Interfaces Package and select New Diagram->Class Diagram to create a new Class Diagram, then use the palette to create the Interfaces (e.g. Producer_IF, Consumer_IF and Store_IF), as depicted in Figure 61.

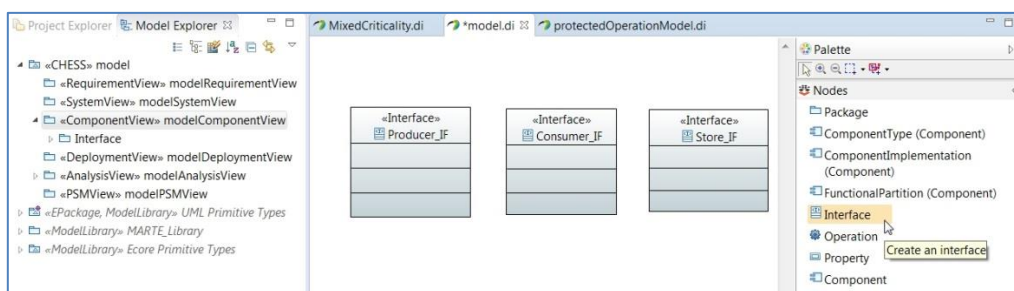


Figure 61 Creating interfaces

Create the operations for the Interfaces that you have just created using “ Operation” from the palette.

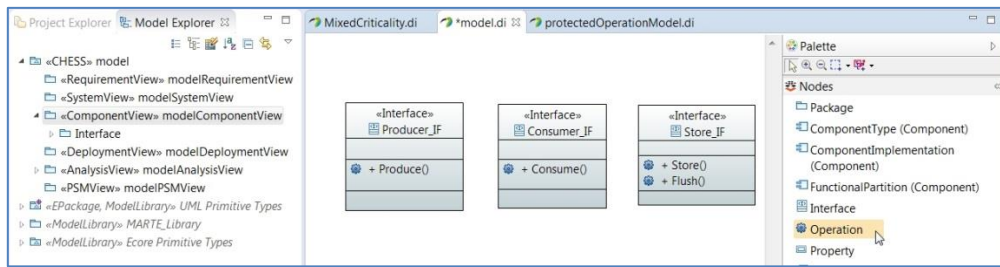


Figure 62 Creating operations

Use the UML tab in the Properties below to define the (in/out) parameters for the operations you have just created.

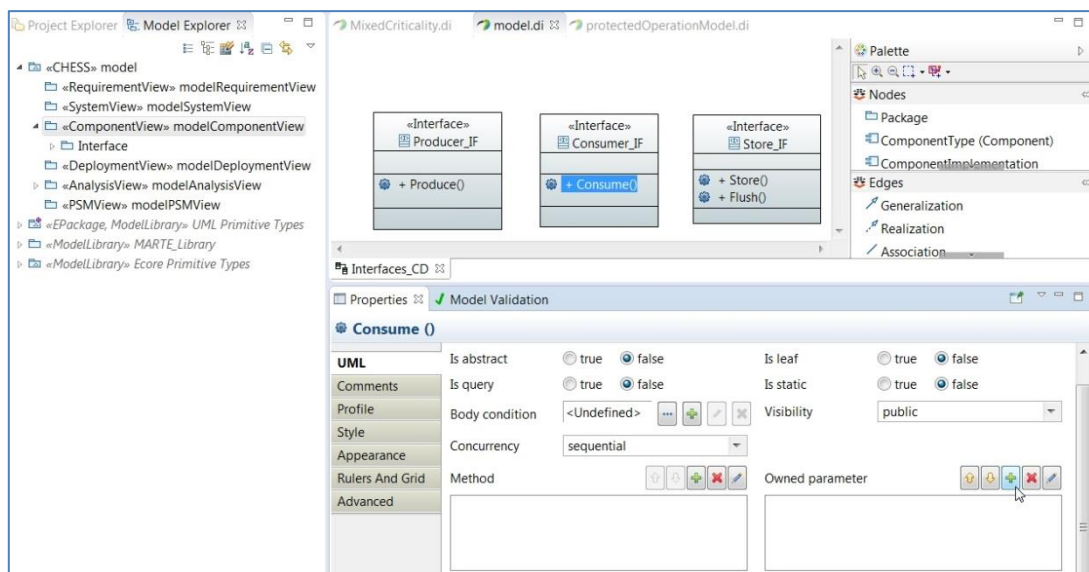


Figure 63 Defining operation's parameters

Assign the type to the parameters from the Type field clicking on “...” and selecting the needed type, as depicted below.

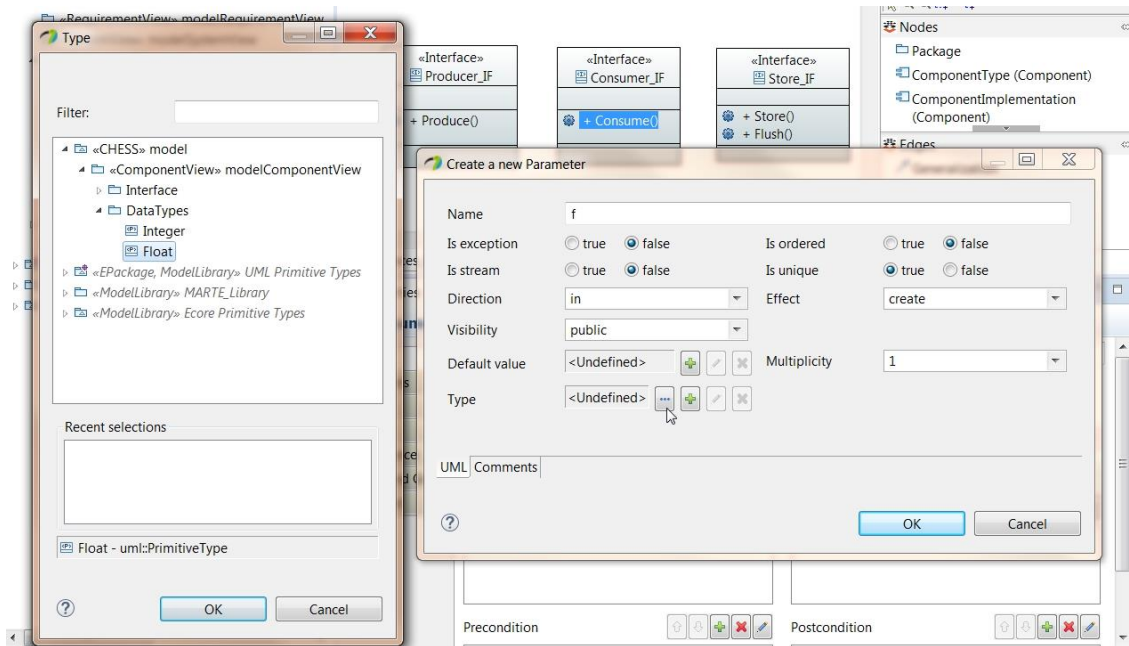


Figure 64 Assigning a type to operation parameters

So now you have created your interfaces and operations, as illustrated below.

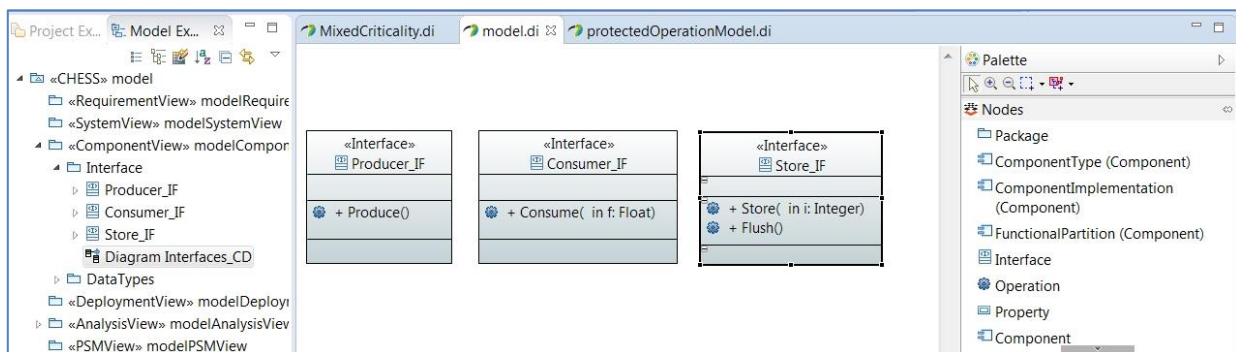


Figure 65 Modeling interfaces and operations

11.1.3 Create component types

Go to the Component View (Functional). Right click on the package and select New Diagram-> Class Diagram.

Create component types using the palette, as depicted in Figure 66.

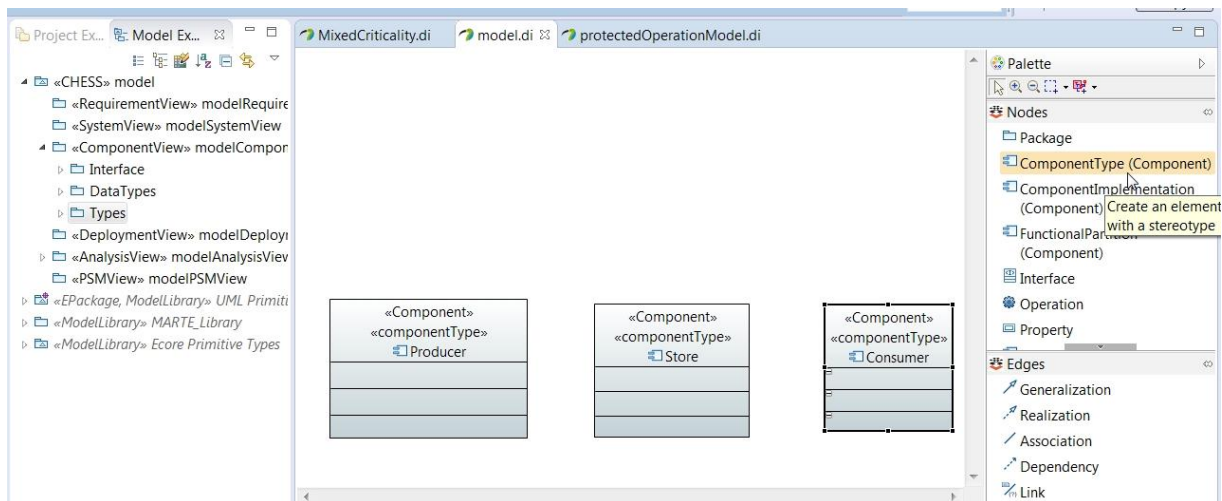


Figure 66 Creating component types

Create the needed Provided and Required Ports for each Component Type:

- Select the ComponentType from the Model Explorer
- Create its Composite Structure Diagram (right-click->Create new Diagram->Composite Structure Diagram)
- Select Provided Port or Required Port icon from the CHESS FunctView palette
- Click on the border of the ComponentType

11.2 Constraints

In the following a brief description of the constraints grouped by category is provided.

11.2.1 Functional View (included in “Core Constraints”)

11.2.1.1 Provided/Required ports of Interfaces (Connector_01)

Check if Interfaces provided and/or required in Ports are not compatible, or components are not at the same level.

This constraint has warning severity and executes in live (automatic) mode.

11.2.1.2 Connector: FlowPorts compatibility of direction (Connector_02)

Check that flow ports on the ends of a connection have compatible directions (i.e. the directions are opposite, or the connection is a delegation and the directions are the same).

This constraint has error severity and executes in live (automatic) mode.

11.2.1.3 Connector: FlowPorts compatibility of type (Connector_03)

Check that flow ports on the ends of a connection are compatible in type.

This constraint has warning severity and executes in live (automatic) mode.

11.2.1.4 FlowPorts not mapped on req/prov operations (FlowPorts_01)

Check that the FlowPort is not mapped to a parameter of a required or provided operation

This constraint has warning severity and executes in batch (on-demand) mode.

11.2.1.5 FlowPorts parameters type compatibility (FlowPorts_02)

Check that the FlowPort should be mapped to parameters of the same type.

This constraint has warning severity and executes in batch (on-demand) mode.

11.2.1.6 FlowPorts parameters direction compatibility (FlowPorts_03)

Check that the FlowPort should be mapped to parameters according to their directions, i.e.:

- port direction IN -> param direction IN or INOUT (param of PROVIDED operation)
- port direction OUT -> param direction OUT or INOUT (param of PROVIDED operation)
- port direction IN -> param direction OUT or INOUT (param of REQUIRED operation)
- port direction OUT -> param direction IN or INOUT (param of REQUIRED operation)
- port direction INOUT -> param direction IN or OUT or INOUT (param of REQUIRED or PROVIDED operation)

This constraint has warning severity and executes in batch (on-demand) mode.

11.2.1.7 ComponentType Interface (FV_02)

Check if the ComponentType does not provide an Interface.

This constraint has warning severity and executes in batch (on-demand) mode.

11.2.1.8 ComponentImplementation realization (FV_03)

Check that:

- the ComponentImplementation must realize exactly one ComponentType
- the ComponentImplementation must define the same operation as the ComponentType it realizes

This constraint has error severity and executes in batch (on-demand) mode.

11.2.1.9 ClientServerPort (FV_05)

Check that the Port must be stereotyped by ClientServerPort FlowPort

This constraint has error severity and executes in batch (on-demand) mode.

11.2.1.10 ClientServerPort SPEC_KIND (FV_06)

Check that the value of ClientServerPort stereotype's attribute SPEC_KIND must be set to INTERFACEBASED

This constraint has error severity and executes in batch (on-demand) mode.

11.2.1.11 ClientServerPort with kind PROREQ (FV_07)

Check that the ClientServerPorts with kind PROREQ are not allowed in ComponentTypes and ComponentImplementations

This constraint has error severity and executes in batch (on-demand) mode.

11.2.1.12 Interfaces (FV_08)

Check that Interfaces should be created only in packages that are descendants of the root package with the stereotype CHESS::ComponentView

This constraint has warning severity and executes in live (automatic) mode.

11.2.1.13 Operations (FV_09)

Check that the Operations should be created only in packages that are descendants of the root package with the stereotype CHESS::ComponentView

This constraint has warning severity and executes in live (automatic) mode.

11.2.2 Extra-Functional View (included in “Core Constraints”, for schedulability analysis)

11.2.2.1 CHRTSpecification Context (EFVRT_01)

Check that the attribute 'context' of CHRTSpecification must be set to an operation of the component (e.g. not of the interface).

This constraint has error severity and executes in batch (on-demand) mode.

11.2.2.2 CHRTSpecification occKind(EFVRT_02)

Check that the attribute 'occKind' of CHRTSpecification must be set or protection must be 'guarded' or 'concurrent'.

This constraint has error severity and executes in batch (on-demand) mode.

11.2.2.3 CHRTSpecification PeriodicPattern (EFVRT_03)

Check that the in CHRTSpecification, if occKind is Priodic, then the attributes 'period' and 'WCET' and 'reIDI' must all be ≥ 0

This constraint has error severity and executes in batch (on-demand) mode.

11.2.2.4 CHRTSpecification PeriodicPattern (EFVRT_04)

Check that in CH_RtSpecification if occKind= Periodic, then phase (if present in occKind) and relativePriority attributes must both be ≥ 0

This constraint has error severity and executes in batch (on-demand) mode.

11.2.2.5 CHRTSpecification partWithPort (EFVRT_05)

Check that CHRTSpecification partWithPort attribute must not be null.

This constraint has error severity and executes in batch (on-demand) mode.

11.2.2.6 CHRTSpecification SporadicPattern (EFVRT_20)

Check that In CH_RtSpecification if occKind=SporadicPattern, then minInterarrival, WCET and reIDI attributes must all be ≥ 0

This constraint has error severity and executes in batch (on-demand) mode.

11.2.2.7 CHRTSpecification priority (EFVRT_30)

Check that CH_RtSpecification if occKind=SporadicPattern, then if relativePriority is set, it must be ≥ 0

This constraint has error severity and executes in batch (on-demand) mode.

11.2.2.8 *CHRTSpecification localWCET (EFVRT_40)*

Check that CH_RtSpecification attribute localWCET must be ≥ 0

This constraint has error severity and executes in batch (on-demand) mode.

11.2.3 Deployment View (included in “Core Constraints”)

11.2.3.1 *Component Implementation deployment (DV_01)*

Check that the ComponentImplementation instance must be allocated on exactly one Processing Unit (for each specified deployment) or one Functional Partition.

This constraint has error severity and executes in batch (on-demand) mode.

11.2.3.2 *Assign must have exactly one “from” and one “to” (DV_02)*

Check that the attribute “from” and “to” of stereotype Assign must be valued with exactly one Element. The value of the attribute 'from' in stereotype Assign must be an instance (InstanceSpecification) of a ComponentImplementation or of a Functional Partition or a <<CHRTPortSlot>> Slot. The value of the attribute 'to' in stereotype Assign must be an instance (InstanceSpecification) of a CH_HwProcessor or of a Functional Partition.

This constraint has error severity and executes in batch (on-demand) mode.

11.2.3.3 *HwBus packett (DV_03)*

Check that the attribute HW_BUS_PACKETT of stereotype HwBus must be set.

This constraint has warning severity and executes in batch (on-demand) mode.

11.2.3.4 *HwBus attributes (DV_04)*

Check that the attribute 'speedFactor' of stereotype HwBus is null, default (value=1.0) will be used.

The attribute 'blockT' of stereotype HwBus is null, default (worst=0.0,unit=ms) will be used.

The attribute 'speedFactor' of stereotype CH_HwProcessor is null, default (value=1.0) will be used.

This constraint has warning severity and executes in batch (on-demand) mode.

11.2.3.5 *Used Functional Partition allocated on exactly one Core (DV_05)*

Check that a Functional Partition that has at least one ComponentImplementation instance allocated on it is allocated on exactly one Core (for each specified deployment).

This constraint has error severity and executes in batch (on-demand) mode.

11.2.3.6 *Un-used Functional Partition allocated on exactly one Core (DV_06)*

Check that a Functional Partition that has NO ComponentImplementation instances allocated on it is allocated on exactly one Core (for each specified deployment).

This constraint has warning severity and executes in batch (on-demand) mode.

11.2.4 Analysis View

11.2.4.1 *AnalysisView unique package AV_01*

Check that there is only one root package of the model on which the stereotype CHESS::AnalysisView is applied.

This constraint has error severity and executes in live mode.

11.2.4.2 *SaAnalysisContext Platform (AV_03)*

Check that the SaAnalysisContext classifier in the Analysis View has its "Platform" property correctly valued with a CHGaResourcePlatform that is a Package in the Component View.

This constraint has error severity and executes in batch (on-demand) mode.

12 References

- [1] CONCERTO, "D2.8 – Multi-concern Component Methodology and Toolset – Final Version," www.concerto-project.org/, 2015.