CHESS

Composition with Guarantees for High-integrity

Embedded Software Components Assembly

CHESS Toolset User Guide

Table of Contents

Introduction

This guide summarizes the status of the CHESS tools (supported and missing features) and gives some hints about its usage.

Tool Status

Version 3.0

Added Ada infrastructural code generation (UPD)

Added C++ functional code generation (MDH)

Added extended data type support.

See Missing Features section.

Missing Features

[MF1] Support to collaborative work

> The Control-Uncontrol feature coming with Eclipse is not currently working due to some bugs/limitations. Moreover its usage in the context of the CHESS toolset has to be deeper investigated.

[MF2] Stereotypes for comments not automatically added

> Stereotypes CHRtSpecification, Assign, FPTC, FPTCSpecification are not automatically applied on Comments when created with the corresponding tool in the palette (Papyrus bug)

[MF3] PSM readable

> The PSM generated by the schedulability analysis should be made (easily) readable to the modelers.

Table 1. TAS Identified issues within WP6 iteration #2 – Summary and EDT responses

Issue	Status	Comment	EDT responses
all Views			
It is impossible to delete a diagram in the model explorer view of the CHESS toolset	Open?		Not able to reproduce this behavior.
Data types: there is the need to model the following data types: • Constants • Arrays • Structures • Ranges	closed	In order to keep the Data types at PIM level aligned with PSM level, the approach is to re-uses PSM data type solution based on MARTE when it exists, and to make new propositions for other types	Supported in v3.0

• …		(constants, arrays …).	
Component View			
How could we model the "event" based communication between components (flow ports?)	open	Solutions to express this at PIM level will be assessed before CHESS 3.0 toolset.	TBD
Operations directly attached to components (without using ports)	??	This is not the approach of CHESS. CHESS development process separate the modeling and the decoration activities. The periodic/sporadic nature of operation is chosen late in the process. This means all the operations defined during modeling phase have to be put in interfaces. However a compromise could be assessed consisting in back propagating decoration activities on instances to component types…	TBD
Extra-functional View			
Decoration contents is not exported during the export of diagrams as images.	open		Papyrus issue
Deployment View			
State-based Dependability Analysis			
Schedulability Analysis			
Users should be aware of the possible range for task priorities (for instance, [1..256]).	open	This data could be configured, possibly using a configuration file ?	TBD
Safety Analysis			

Table 2. Telecom Identified issues within WP6 iteration #2 – Summary and EDT response

Issue	Status	Comment	EDT responses
Component View			
Call operation exception	Solved	Identified in v2.0. Issue not present in v2.1.	Probably Papyrus (solved) exception
Updating operations in interfaces	Partially solved	Occurs in v2.0 (on models created with v1.2). Not present in new models created with v2.0.	
'Build instance' command "no text" status	Partially solved	Occurs in v2.0 (on models created with v1.2). Not present in new models created with v2.0.	
Extra-functional View			
Missing <<Propagation>> stereotype	Partially solved	Occurs in v2.0 (on models created with v1.2). Not present in new models created with v2.0.	Issue with backward compatibility: can be fixed with little modification on the XMI model file.
Attribute value change is not shown in the diagram.	Not solved	Present in v2.0 and v2.1.	Papyrus issue.
Deployment View			
Flow port type structure	Not solved	Ongoing discussion	TBD
Modeling memory	Not solved	Present in v2.0 and v2.1	TBD
Saving the model	Unknown	Not tested in v2.1 Present in v2.0 Workaround for v2.0	Not able to reproduce this behavior.
Assignment of SW instanced on HW	Not solved	Ongoing discussion	TBD

State-based Dependability Analysis			
Missing <<ChGaResourcePlatform>> stereotype	Solved	Not present in v2.1	
'Build instance' command resets attribute values	Not solved	Present in v2.0 and v2.1	TBD
Error model definition	Unknown	Not tested in v2.1. Present in v2.0 Ongoing discussion	TBD
Schedulability Analysis			
Exporting model from CHESS 2.0 to CHESS 2.1	Solved	-	
Strange analysis results	Not solved	Ongoing discussion	
'relativePriority' attribute is not optional	Not solved	Present in v2.0 and v2.1	TBD
'Check model for schedulability analysis tool' option doesn't cover all error cases	Not solved	Present in v2.0 and v2.1	TBD
'Build instance' command sometimes fails	Unknown	Not tested in v2.1. Present in v2.0	It should be related to model created with old profile versions
Safety Analysis			
Safety analysis modeling in CHESS	Not solved	Present in v2.0	TBD. Not easy to introduce needed permissions without preserving the proper separation of concern (for example editing of SA dependency in the extra functional view).
Problem with profile application URIs	Partially solved	Workaround for v2.0 Should be solved in upcoming releases	

Table 3. Railway Identified issues within WP6 iteration #2 – Summary

Issue	Status	Comment	EDT responses
Component View			

Extra-functional View			
Deployment View			
State-based Dependability Analysis			
Dependability analysis does not support the ability to specify multiple "targetDepComponent".		Present in v2.0 and v2.1	Solved in 3.0
Created an error model for BoardSystemHardware Component but it can not be used because the analysis has produced abnormal results (see bug **:[#13610]**)	Not solved	Present in v2.0 and v2.1	
Schedulability Analysis			
With CHESS 2.0 the schedulability analysis does not work if there are hardware components defined in the model but not used in the analysis performed (see bug :[#13609])	Solved in 2.1	Present in v2.0	
Safety Analysis			

Working with Views and Diagrams

Component View

Functional view

Use Case Diagram

To model system use cases (used by FTA/FMECA)

Class Diagram

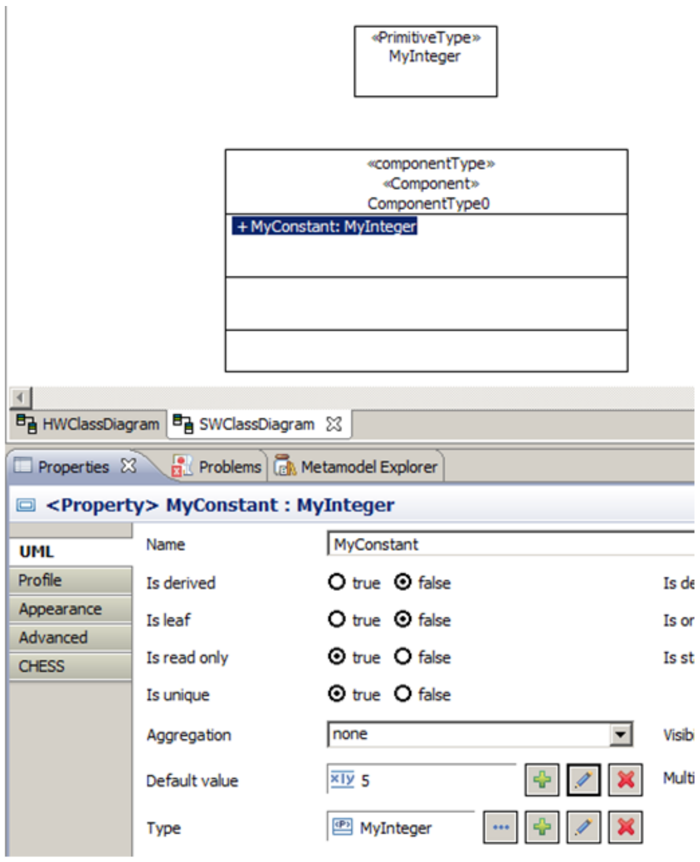
To model Packages, data types, Interfaces, ComponentTypes, ComponentImplementations, Operations, Properties.

About Modeling Data Types

Primitives Types are supported, they can be created through the CHESS FunctView palette.

In a given Component **Constants** can be modeled by selected the 'read-only' attribute of the given Property.

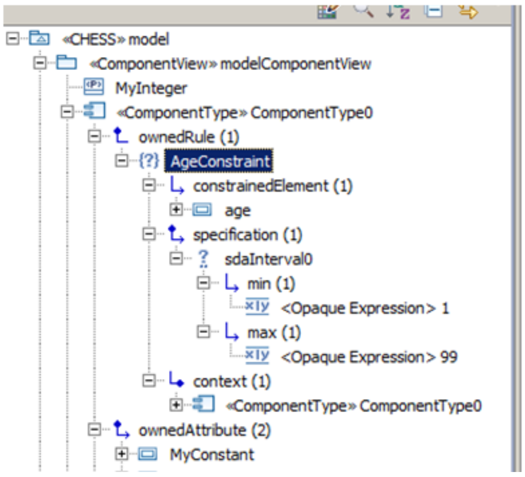
Default value can be specified by setting the 'default value' attribute of the property, for instance by using an OpaqueExpression (see figure below).



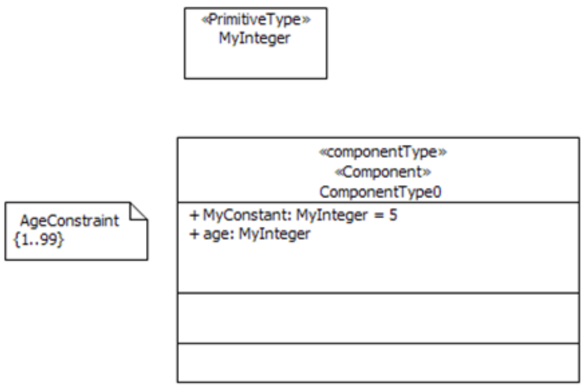
<Property> MyConstant : MyInteger

Name	MyConstant
Is derived	true [✓ false]
Is leaf	true [✓ false]
Is read only	[✓ true] false
Is unique	[✓ true] false
Aggregation	none
Default value	5
Type	MyInteger

Data ranges can be modeled by using standard Constraint having Interval element as 'specification' attribute. The 'context' of the Constraint as to reference the property to bound (see figure below for an example).

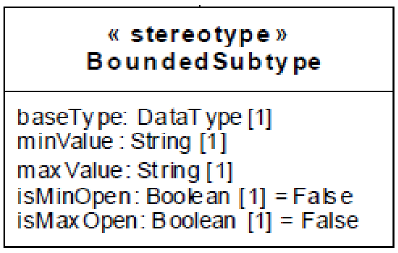


```
«CHESS» model
  «ComponentView» modelComponentView
    MyInteger
    «ComponentType» ComponentType0
      ownedRule (1)
        AgeConstraint
          constrainedElement (1)
            age
          specification (1)
            sdaInterval0
              min (1)
                <Opaque Expression> 1
              max (1)
                <Opaque Expression> 99
          context (1)
            «ComponentType» ComponentType0
      ownedAttribute (2)
        MyConstant
```

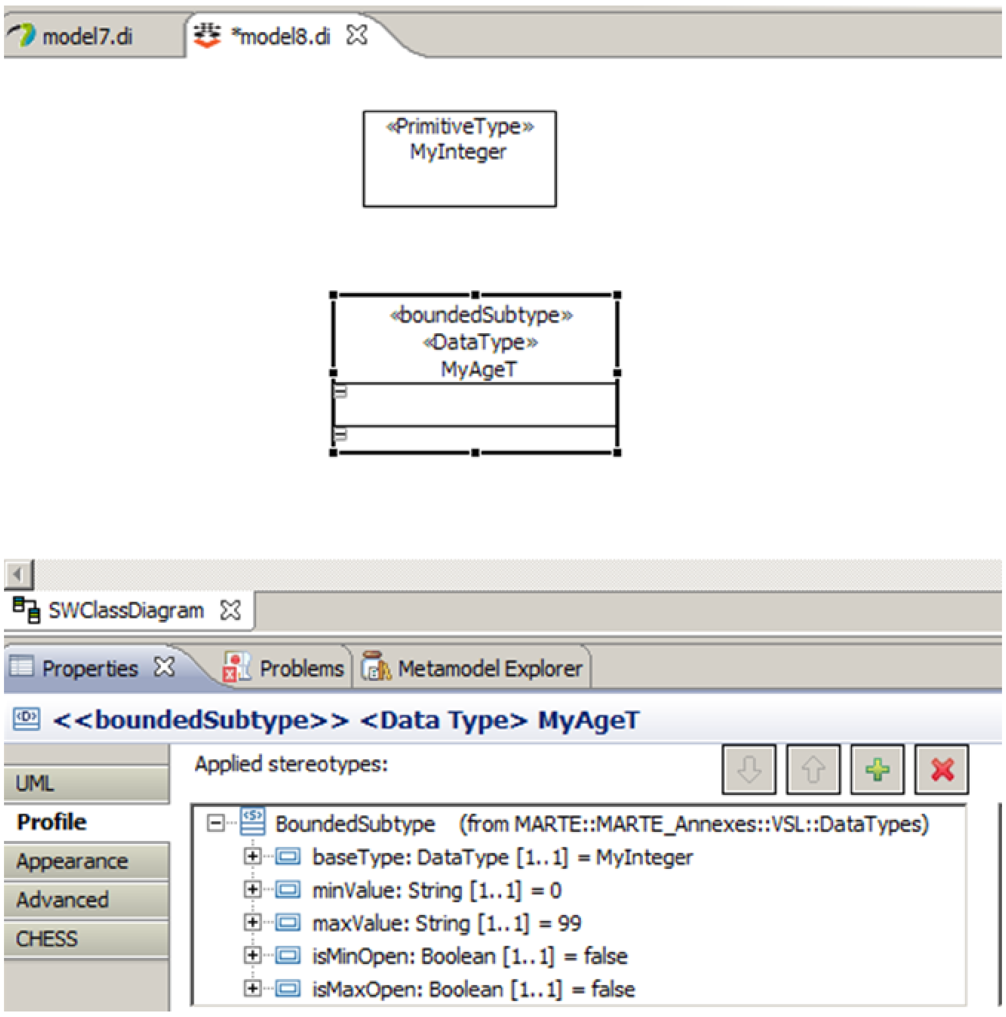


Another possibility for data ranges would be using IntervalType from MARTE VSL but currently it is not properly implemented in Papyrus.

In alternative MARTE::VSL::DataTypes::BoundedSubtype stereotype can be used



baseType: DataType [1]
minValue : String [1]
maxValue: String [1]
isMinOpen: Boolean [1] = False
isMaxOpen: Boolean [1] = False

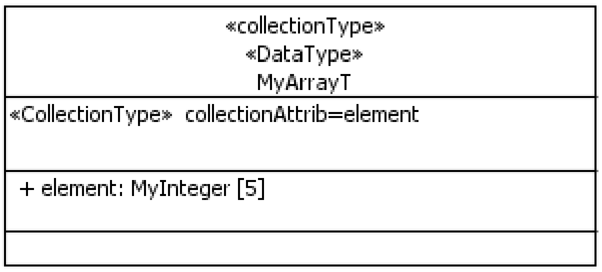


<<boundedSubtype>> <Data Type> MyAgeT

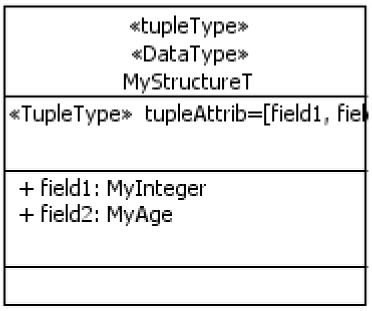
Applied stereotypes:

```
BoundedSubtype   (from MARTE::MARTE_Annexes::VSL::DataTypes)
    baseType: DataType [1..1] = MyInteger
    minValue: String [1..1] = 0
    maxValue: String [1..1] = 99
    isMinOpen: Boolean [1..1] = false
    isMaxOpen: Boolean [1..1] = false
```

Arrays can be modeled by using CollectionType from MARTE VSL.



Structures can be modeled by using TupleType from MARTE VSL.



Composite Structure Diagram

For a given ComponentImpl, to model:

- provided/required ClientServerPort
- ComponentImplementation instances and connectors (also Component can be used in place of ComponentImplementation as context of the Composite Structure Diagram)

Composite Structure diagrams in this view are also used by the tool to automatically build the software InstanceSpecifications, with the proper extra-functional information attached (**Build Instance command**).

State Machine Diagram

For a given ComponentImplementation, to model functional behavior for ComponentImplementation.

Supports ALF.

They can be used to generate functional code (currently C++ generation).

Activity Diagram

For a given Operation, to model:

- intra ComponentImplementation bindings, i.e. the called Operations (information used by Schedulability Analysis from UPD)
- operation behavior (full definition of available constructs coming soon), used by deployment analysis

Extra Functional View

Use Case Diagram

To model stereotypes for FTA/FMECA.

Class diagram

To model stereotypes for StateBased, FailurePropagation, FMECA/FTA … (other analysis coming soon)

Composite Structure Diagram

To model real time , state based analysis, failure propagation, FTA/FMECA information… (other analysis coming soon)

From the main CHESS menu in the toolbar: CHESS->Filters->CHRtSPecification->Show/Hide to manage CHRtSpecification visibility for the current diagram.

Right click on a ComponentImplementation instance, select Filters->CHRtSPecification->Show/Hide to manage CHRtSpecification visibility for the current instance.

State Machine Diagram
For a given ComponentImpl, to model dependability ErrorModel.

Activity diagram
Not used

Deployment View

Class diagram
To model hardware components (only the ones needed by the predictability analysis).

Composite Structure diagram
For a given hardware component, to model:

- packages
- data flow ports
- hardware components
- hardware components instances and connectors.
- Allocation of software to hardware (InstanceSpecifications generated through the Build Instance command have to be referred for the allocation).

CompositeStructure diagram 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 Instance command** invocation.

DependabilityView

Class diagram
To model dependability concerns for hardware components.

Composite Structure diagram
To model dependability concerns for hardware components instances.

StateMachine
To model dependability ErrorModel for hardware components.

Requirement View

Requirement Diagram
To model requirements and traceability between architectural elements (e.g. ComponentImplementation) and requirements.

Working with the instances
Hardware and Software (i.e. ComponentImplementation) Instances in CHESS can be modeled through the Composite Structure Diagram, i.e. through Properties.

Due to UML limitation about the entities-instances available in the Composite Structure Diagram, all the information appearing in the Composite Structure Diagram need to be properly represented in the model by using UML InstanceSpecifications in order to be properly used by model transformations.

The CHESS toolset allow to automatically derive the InstanceSpecifications set by starting from a Composite Structure Diagram. In particular each Property and Connector are mapped into a dedicate InstanceSpecification, while Ports are mapped into Slot. Extra functional information is attached to InstanceSpecifications and Slots.

To invoke the Build Instance command:

-open the Composite Structure Diagram where the instance (as Properties) have been modeled

-from the CHESS Toolbar menu select Build Instances entry

-the InstanceSpecifications/Slots are generated in a dedicated package

Tips Multiple Composite Structure Diagrams are supported in the ComponentView, in particular one diagram for each ComponentImplementation. In this way the hierarchy of ComponentImplementation can be modeled by using a hierarchy of Composite Structure Diagram. In this way just invoke the Build Instance command from the root Composiste diagram: the tool automatically builds the instances hierarchy while navigating the diagrams.

⚠ Only one Composite Structure Diagram for a given HW/SW component is supported.

Working with the CHESS palettes

CHESS toolset implements the proper palettes to work with the current view and diagram.

At any time the tool shows the proper palette by considering the current view and diagram.

Working with the model validation

From the Papyrus Model Browser select the Model or the entity from which perform the validation and choose:

- Validation->Validate CHESS model: performs checks starting from the selected entity and all the owned ones.
- Validation->Check model for Thales code generation tool: checks specific preconditions required by Thales code generation tool
- Validation->Check model for (UPD) schedulability analysis tool: checks specific preconditions as required by the (UPD) schedulability analysis.
- Validation of preconditions for the other analysis coming soon.

Working with the 42Analyzer tool

(A advanced tutorial is available)

Select a composite structure diagram where ComponentImplementation instances have been designed and decorated with real time information.

Right click on the main Component which is the context of the current Composite Structure diagram and select “CHESS : 42 analyzer” command: a report about tasks and protected resources is generated in a dedicated panel.

Working the VSL editor to edit stereotype attributes

Stereotype attributes can be set by using the Profile tab in the Properties view. Moreover stereotype attributes can be set by using a VSL editor: this is particularly useful to edit complex stereotype attributes (e.g. typed as records), like the ones inherited by MARTE (e.g. NFP_xxx, OccKind).

Select the stereotyped entity in the diagram for which the attributes need to be filled.

Right click and select “Open Textual Editor for Stereotype Applications”: the textual editor is opened in a dedicated panel.

The list of applied stereotypes is shown in the editor, for instance:

```
<<StereoX>>

Attr1 = <value here> ,

… ,

AttrN = <value here>

<<StereoY>>

…..
```

For each stereotype a list of attributes can appear by default. If the desired attribute is not visible:

> put the cursor just under the stereotype, or after the comma which follows an existing stereotype’s attribute, then use CTRL+space to make appear the list of available attributes for the current stereotype. Select the desidered attribute.

To edit the X attribute:

> If the X attribute is already visible simple type the value directly after ‘=’. Put the cursor after ‘=’ and use the CRTL+space to get editor assistance.

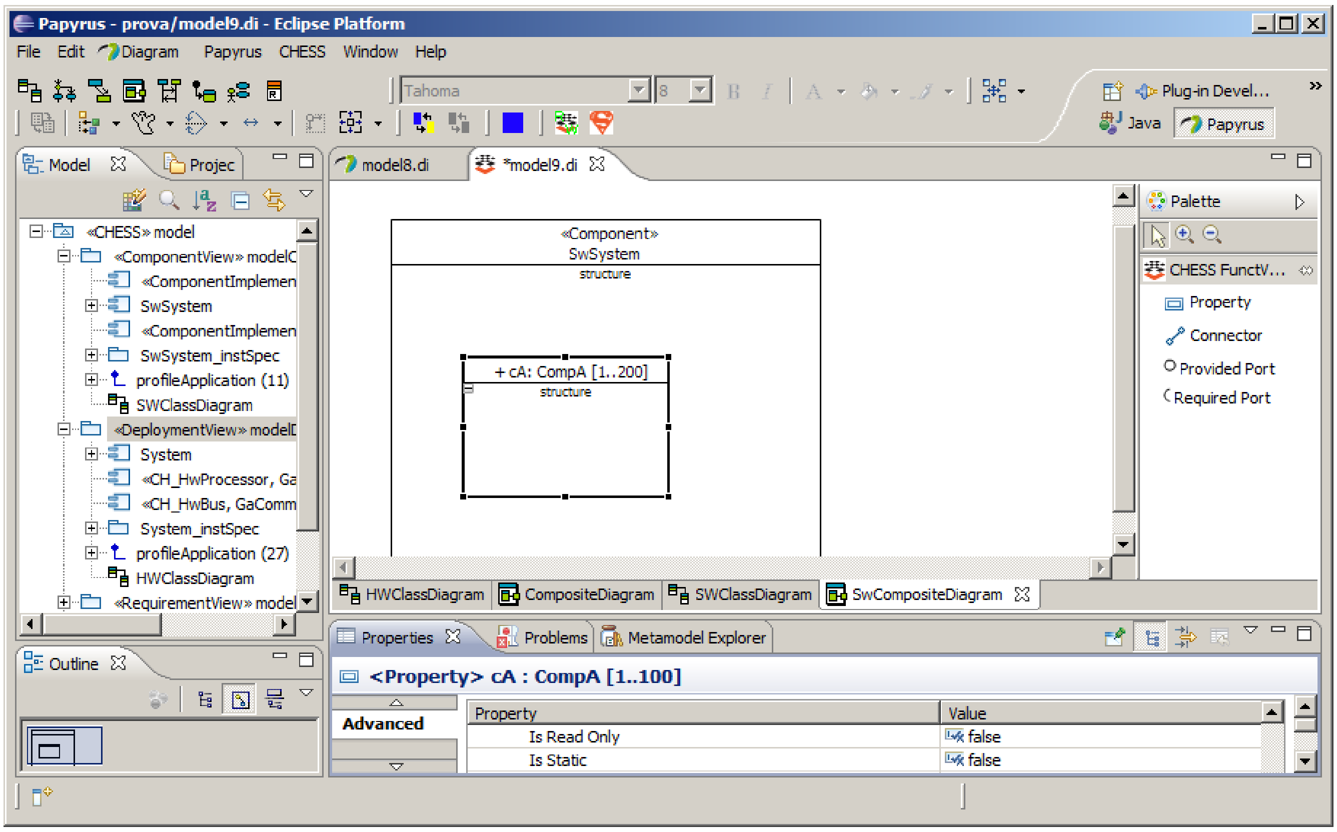
Allocating multiple SW instances to HW (Proposal for CHESS v3.1)

CHESS allows to specify SW to HW deployment by using the Assign MARTE feature. Moreover CHESS supports modeling of SW multiple instances, i.e. instances having the same Component has classifier/type.

While working with the composite structure diagram to model SW component instances, it is possible to directly instantiate multiple instances of the same Component just creating different Properties, where all the Properties share the same Component as type. Then the Build Instance command can be used to create the corresponding instanceSpecifications which in turns can be referred by the Assign.from field for deployment on the hardware.

This is one way to address the deployment of multiple instances and it can be useful and usable only if the number of multiple instances is reasonably low. If this is not the case, e.g. having hundreds of instances of the same component to represent, then it is possible to use the multiplicity attribute of the Property in the compositste structure diagram. In this case the Build Instance command does not generate all the corresponding multiple entities but just a single InstanceSpecification representing all the set. So, concerning the deployment, the aforementioned InstanceSpecification can be referred by the Assign.from field, then it is possible to use NfpConstraint to specify the proper (sub-)set of instances to be allocated on the same hardware.

Figure below shows how this can be modeled with an example: suppose to have a Property of type CompA with multiplicities 200

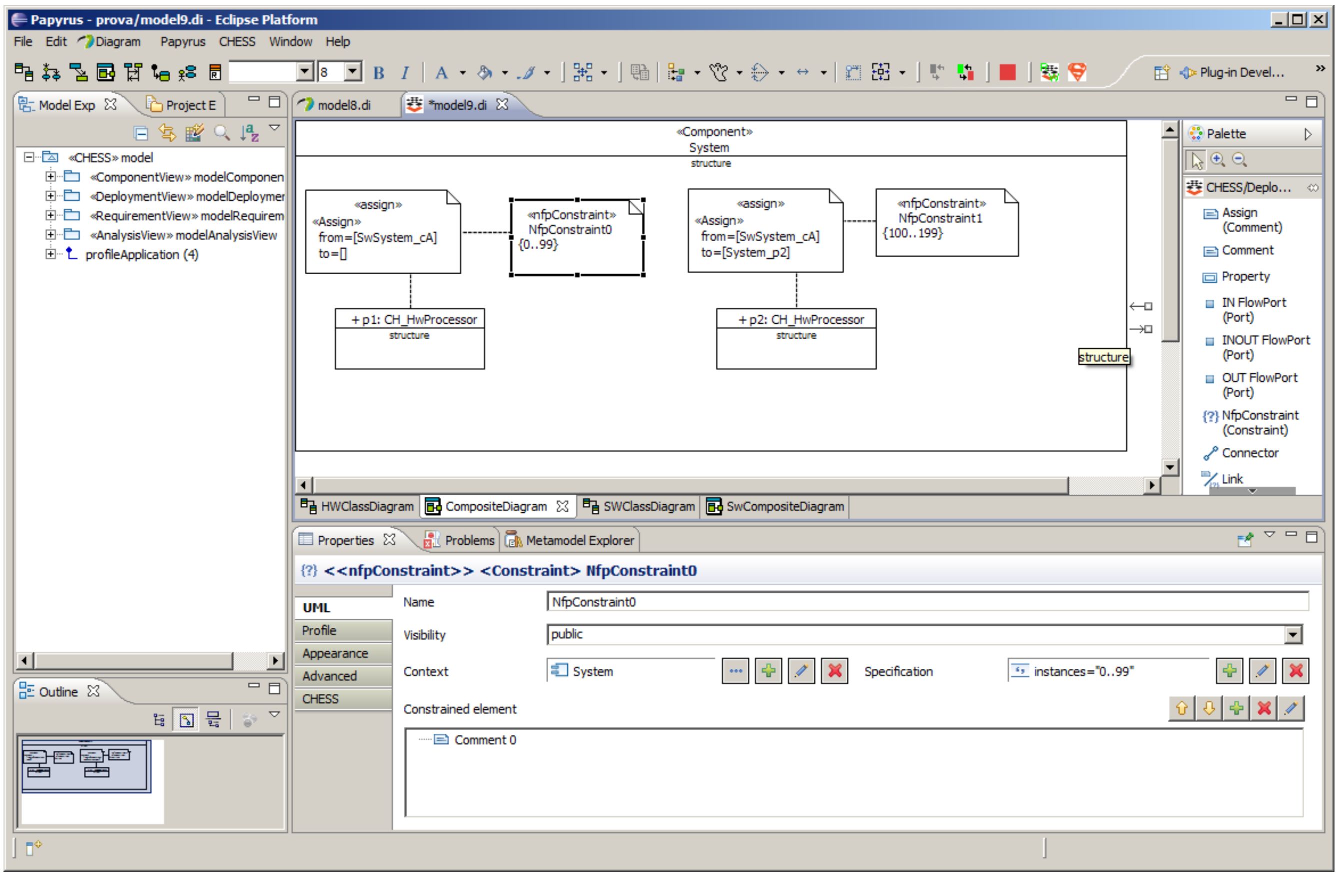


<Property> cA : CompA [1..100]

Advanced

Property	Value
Is Read Only	false
Is Static	false

Then if we want to model the scenario where instances from 0 to 99 are deployment on a processor and instances from 100 to 199 are deployed on a different processor, then we can use the NfpConstraint like in the following figure.



<<nfpConstraint>> <Constraint> NfpConstraint0

Name: NfpConstraint0

Visibility: public

Context: System

Specification: instances="0..99"

Constrained element: Comment 0

Note that for CHESS model created with CHESS tool v<3.0 it is required to manually apply the MARTE::MARTE::Foundations::NFPs profile to the DeploymentView Package in order to be able to create the NFPConstraint element.

Currently the Assign.impliedConstraint attribute cannot be set in Papyrus 0.8.1, so Link can be used to associate the NFPConstraint to the given Assign.

Supported Automations

[Auto1] Action:

a port providing an interface is added to a ComponentType.

Effect:

the operations listed in the Interface are automatically added in the ComponentType. These new operations are not automatically showed in the diagram (see [tip 1]).

a InterfaceRealization link from the ComponentType to the Interface is added in the model (if not already defined).

[Auto2] Action:

a port requiring an interface is added to a ComponentType.

Effect:

A Dependency link from the ComponentType to the Interface is added in the model (if not already defined)

[Auto3] Action:

a Realization is traced from a ComponentImplementation to a ComponentType.

Effect:

The operations and ports owned by the ComponentType are replicated in the ComponentImplemenation. These new entities are not automatically showed in the diagram (see [tip1]).

[Auto4] Action:

a Parameter is added/changed/removed to/in/from an Operation owned by an Interface.

Effect:

The same kind of Parameter is added/changed/removed to/in/from the corresponding Operation owned in the ComponentType and ComponentImplementation which realize the given Interface.

[Auto5] Action:

a port is removed from a ComponentType.

Effect:

The corresponding port is removed from all the realizing ComponentImplementation.

[Auto6] Action:

an Operation is added on an Interface.

Effect:

A corresponding Operation is added on each ComponentType and ComponentImplementation realizing the Interface.

Tips

[Tip 1] Show information stored in the model in a given diagram (if allowed by the CHESS profile):

-drag and drop the information from the model browser into the diagram (e.g. to show existing Components in a Class diagram) or diagram entity (e.g. to show a Component's Operation in a Class diagram)

-right click on the diagram entity, then select Show/Hide Contents: the properties that can be showed/hided for the current diagram entity are listed for selection.

[Tip 2] Show default Papyrus palette.

⚠ Warning: this action should be performed to overcome possible limits/bugs of the implemented CHESS palettes support coming with the current CHESS editor release. Please contact the EDT team to notify the existing problem.

-right click on the palette, deselect the Papyrus standard palette.

-right click on the palette, select the Papyrus standard palette: the palette is showed.

[Tip 3] The CHESS SuperUser.

⚠ Warning: this action should be performed to overcome possible limits/bugs of the implemented CHESS view support. Please contact the EDT team to notify the existing problem.

Click the CHESS SuperUser button located in the CHESS toolbar to skip all the checks concerning working with views.