CHESS Toolset User Guide

Toolset Release 1.0.0

06 May 2020

1 Table of Contents

1.1 List of Figures

2 Document history

Date	Changes
Current version	Update with respect to the delivery of tool version 1.0.0. Restructuring and reduction of scope as introduction guide, links are provided to separate guides for details on specific modelling and analysis methods.

3 Introduction

This guide introduces to the CHESS Toolset features and their status, as well as basic hints for the usage of the toolset for modelling system and software models and performing analysis.

The CHESS toolset is an Eclipse based open source methodology and toolset, aiming to improve MDE practices and technologies to better address safety, security, reliability, performance, robustness and other non-functional concerns, while guaranteeing correctness of component development and composition for critical embedded systems.

CHESS provides editing capabilities to model all phases of development, from the definition of requirements, to the modelling of the system functional, logical and physical architecture, down to the software design and its deployment to hardware components. The CHESS toolset offers schedulability and dependability analysis functionalities across the life cycle. According to the results of the analysis, that are back-propagated to the model itself, the engineer can perform some tuning on the model in order to satisfy real time and dependability requirements.

To find out more about the CHESS project (e.g. the methodology and language) see http://chess-project.org.

4 The role of MDT Papyrus

CHESS is built as set of extensions of the MDT Papyrus Eclipse UML editor.

So a basic knowledge of MDT Papyrus (not covered by this user manual) is required to be able to work with a CHESS model. General documentation about MDT Papyrus is available at:

http://wiki.eclipse.org/Papyrus_User_Guide

Also the *PAPYRUS USER GUIDE SERIES About UML profiling*[1] provides useful information about working with stereotypes and stereotypes values.

[1] http://www.eclipse.org/modeling/mdt/papyrus/usersTutorials/resources/PapyrusUserGuideSeries_AboutUMLProfile_v1.0.0_d20120606.pdf

5 Creating a new CHESS project

To create a CHESS project in the local Eclipse workspace:

- Open the Project Explorer view (if not already visible on the left side of the Eclipse area) by using the menu Window->Show View->Other…->General->Project Explorer
- Right click on the Project Explorer and select New->Other…->CHESS->CHESS Project, provide a name for the current project and click Finish.

The tool will ask to switch to the Papyrus perspective; this is the right perspective to be used while working with a CHESS project.

6 Creating a new CHESS model

To create a CHESS model:

- right click on a CHESS project in Project Explorer view
- select New->Other…->CHESS->CHESS Model, then provide a name for the current project.
- Right click on the Project Explorer
- select New->Other…->CHESS->CHESS Project, then provide a name for the current model and click Finish.

A CHESS model is created with the CHESS profile automatically applied on it; moreover the default packages structure supporting the CHESS views is created in the model.

The detailed CHESS profile is available at < link profile >

6.1 Defining the domain of the CHESS model

In order to define the Domain of the CHESS model:

- Click on the CHESS model in the Model Explorer
- Select the Properties View
- Go to the Applied Stereotypes field
- Select the CHESS (from CHESS::Core) Stereotype
- Select the “domain” field
- Assign the appropriate domain of your model from the scroll list:
  - Cross_domain
  - Avionics
  - Automotive
  - Telecom
  - Space
  - Medical
  - Petroleum

6.2 Naming CHESS model elements


Please notice that CHESS model elements **should NOT contain spaces** in their names.

7 Creating diagrams

Model diagrams can be easily created:

- From the Model Explorer view, by right clicking on the model element which has to own the new diagram
- From the main Papyrus->Diagrams menu
- From the main toolbar by clicking on the desired diagram icon.

Note that in the last two cases the new diagram is owned by the currently selected model entity.

7.1 The CHESS diagram palettes

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

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

8 Working with the CHESS Views

The CHESS toolset support several design views, which are the means to enforce separation of concerns, as defined by the CHESS methodology.

Figure 1 summarizes the CHESS views. In the same figure we also depict the main design activities allocated to the different views, their related concerns, and some precedence constraints, depicted as arrows.

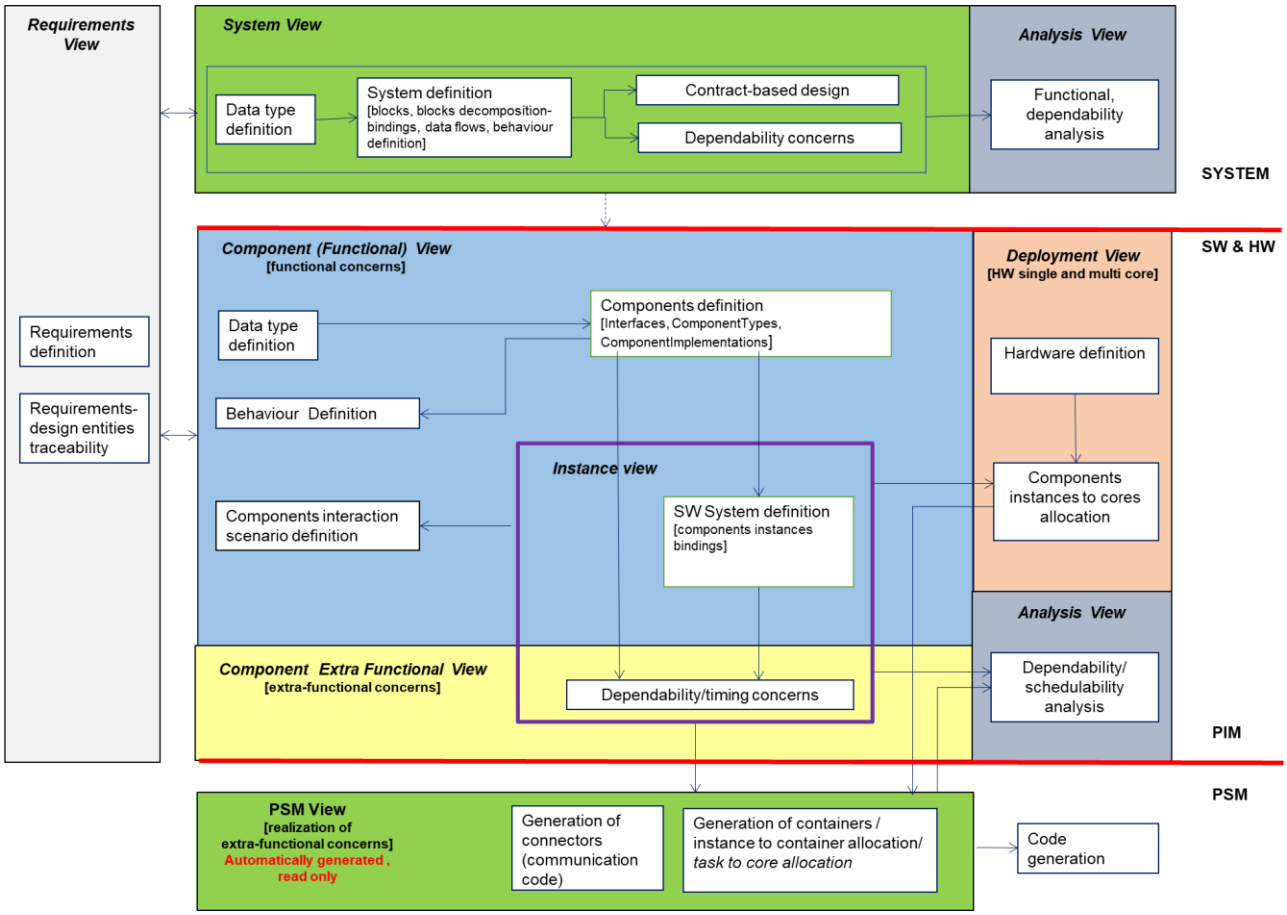

Figure 1: Design Flow

In the following sections we summarize the kind of diagrams and entities that are created in each view.

8.1 Requirement View

Requirements are modeled in the Requirement View package through the SysML Requirement Diagram.

The Requirement View palette allows to create the model entities, as depicted in Figure 2

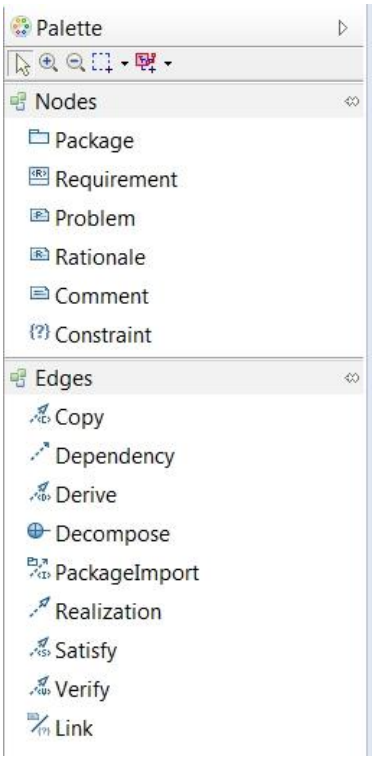

Figure 2 Requirement View - Requirement Diagram palette

8.2 System View

The System View provide a suitable frame for the system level design activities. In the System view system entities are initially designed at a high level of abstraction and then hierarchically decomposed.

The System View is used to work with contract-based design, dependability; several functional and dependability analysis are supported at this level (see the "CHESS guide for contract-based analysis, model checking, and safety analysis").

The diagrams allowed in the System View to model the System Architecture are:

- Block Definition Diagram (BDD)
- Internal Block Diagram (IBD)
- State Machines (SM): e.g. used to model dependability ErrorModel
- Sequence Diagrams: e.g. used to model dependability information (e.g. attack scenarios)

8.3 Component View

This view is conceived to support the design of software components. For full details related to the software development and schedulbility analysis see the "CHESS Software Development Guide".

The Component View is actually composed by two sub-views, the Functional View (enabled by default) and the Extra-Functional View.

It is possible to switch between Functional and Extra-functional views using the button with the Blue/Yellow squares, as illustrated in Figure 3. The blue square indicates that the current view is the Functional view, while the yellow square indicates the Extra-functional view.

Figure 3 Switching between Functional and Extra-functional View

8.3.1 Functional View

Through this view the functional specification of the software can be provided. The next paragraphs illustrate the diagrams that can be modeled in the Functional View.

8.3.1.1 Class Diagram

Class diagrams are used to model Packages, Data Types, Interfaces, Component Types, Component Implementations, Operations and Properties.

UML entities allowed in Class Diagrams are enforced by the customized palette, as illustrated in Figure 4.

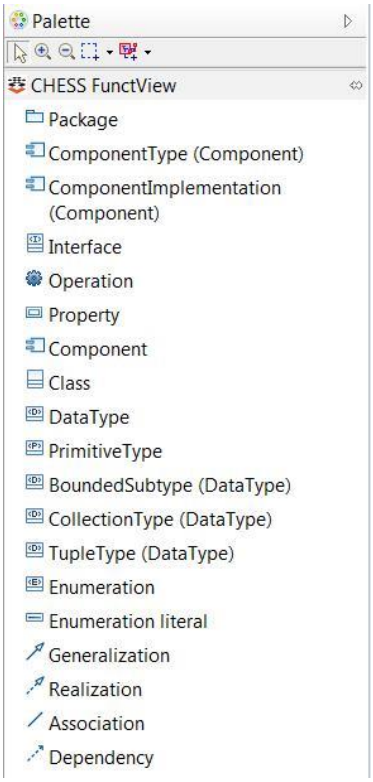

Figure 4 CHESS Funct View - Class Diagram Palette

Figure 5 illustrates the modeling of Interfaces in a Class Diagram in the CHESS Functional View.

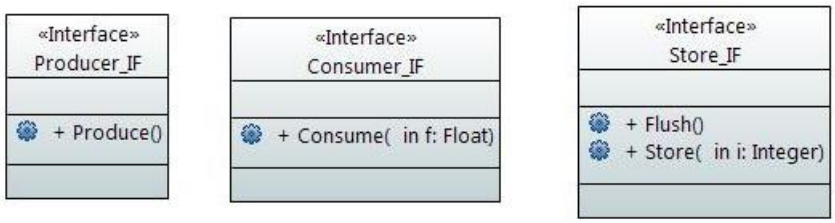

Figure 5: CHESS Funct View - Modeling Interfaces

Figure 6 illustrates the modeling of Component Types in a Class Diagram in the CHESS Functional View.

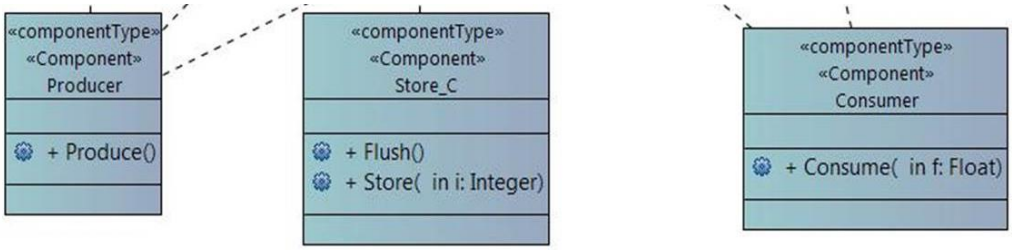

Figure 6: CHESS Funct View - Modeling Component Types

Figure 7 illustrates the modeling of Component Implementations in a Class Diagram in the CHESS Functional View.

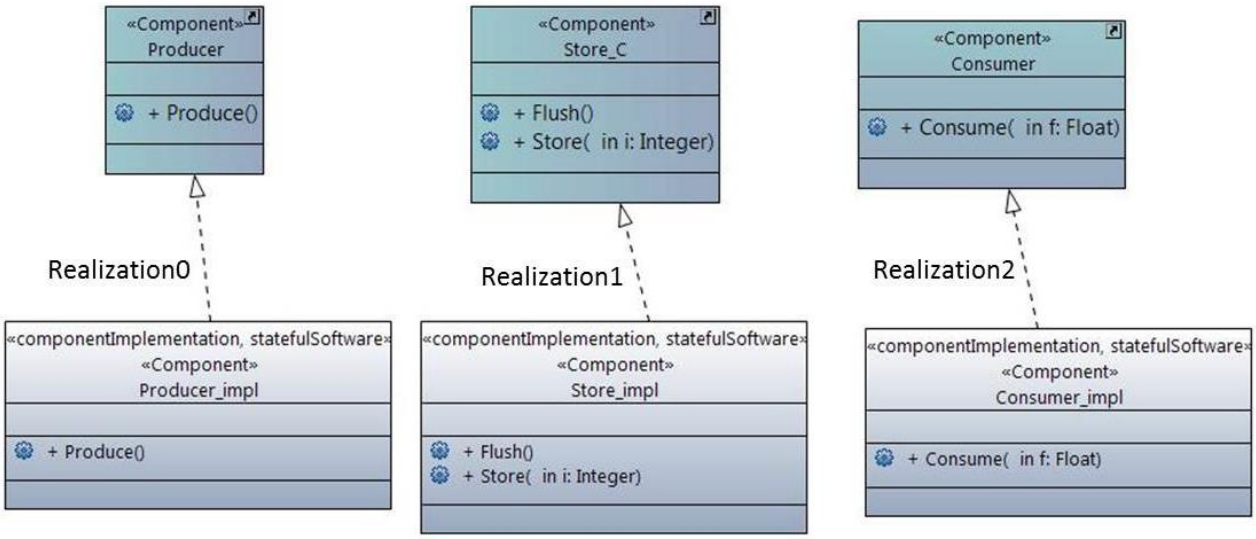

Figure 7: CHESS Funct View - Modeling Component Implementations

8.3.1.2 Composite Structure Diagram

A Composite Structure Diagram is created for a given component, to model:

- provided/required ClientServerPort

- ComponentImplementation instances and connectors; this is not allowed for the <<ComponentType>> Components.

The information provided through the Composite Structure diagrams in this view are also used by the tool to automatically build the software Instance Model (using the **Build Instances command** available through the Context Menu as depicted in Figure 8).

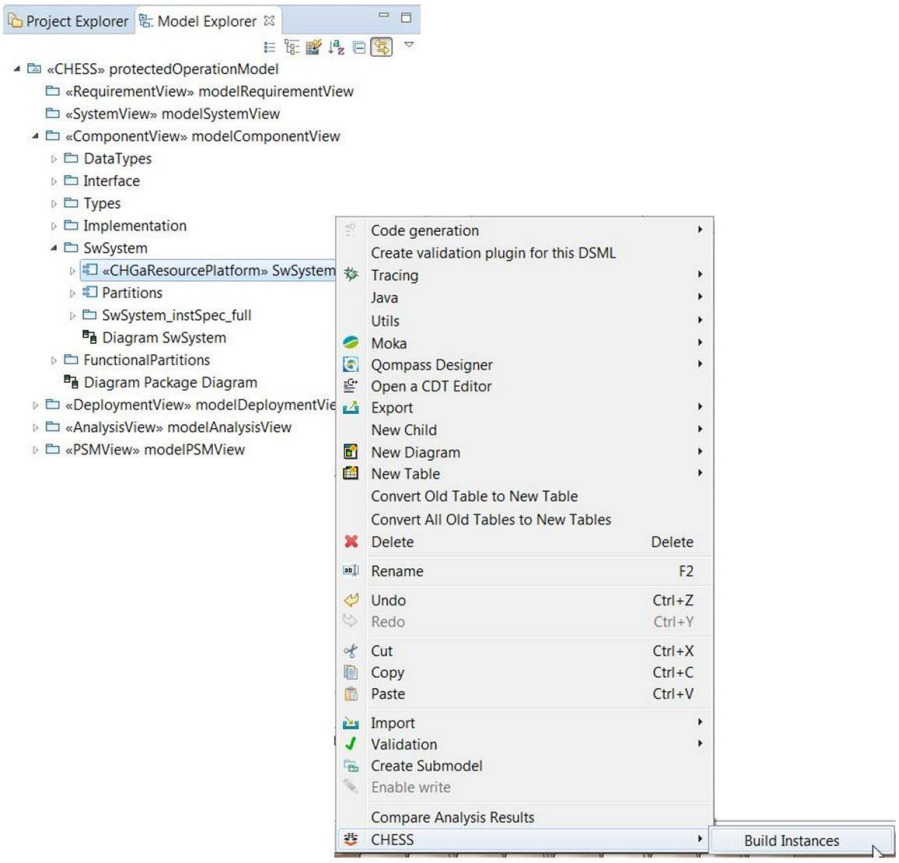

Figure 8 The Build Instances command

See 8.6 for more information about modeling instances.

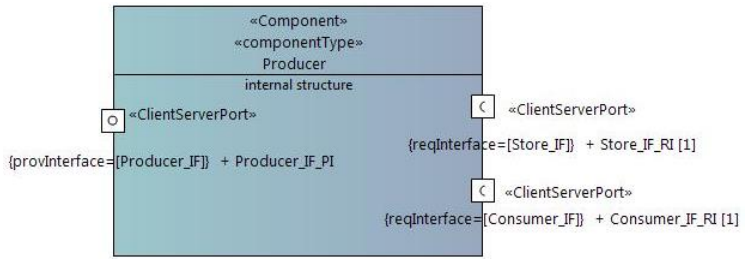

Figure 9: CHESS Funct View: Modeling provided and required Ports in Composite Structure Diagram

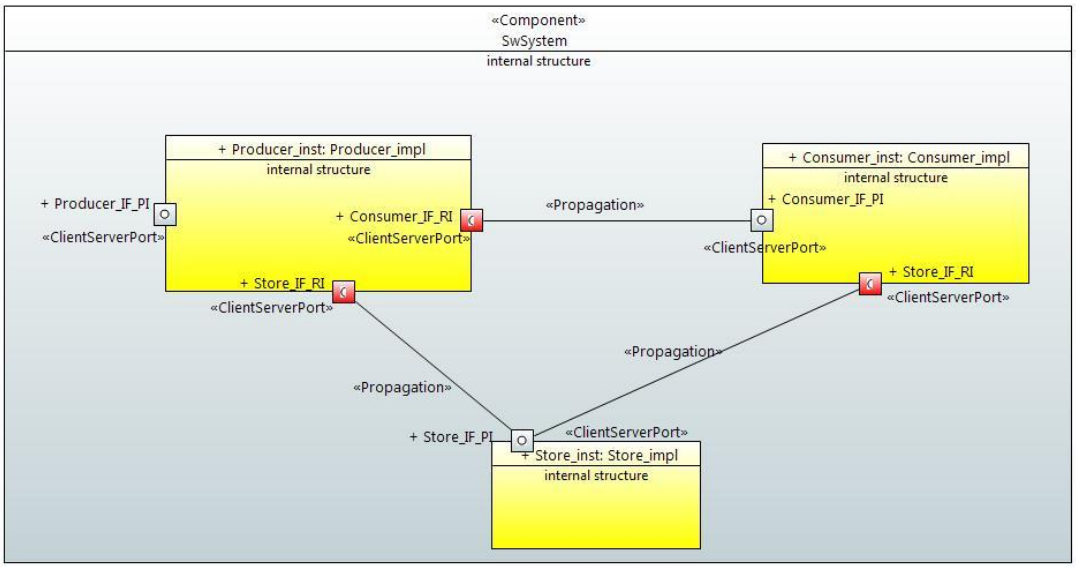

Figure 10**: Modeling Instances in a Composite Structure Diagram**

8.3.1.3 State Machine Diagram

A State Machine Diagram is created for a given ComponentImplementation, to model functional behavior for ComponentImplementation.

8.3.1.4 Activity Diagram

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).

 Currently Decision nodes are not supported by the Schedulability analysis.

8.3.1.5 Sequence Diagram

Sequence diagrams are used to model collaboration scenario. The scenario can be given in input to the schedulability analysis.

8.3.2 Extra Functional View

Through this view the extra-functional specification of the software can be provided, such as the real time and dependability attributes. The next paragraphs illustrate the diagrams that can be modeled in the Extra-functional View.

 The Extra-functional View is indicated in the editor with a yellow square.

Figure 11 Extra-functional View

8.3.2.1 Class diagram

Class Diagrams are used in the Extra-functional View to work with dependability stereotypes (e.g. related to StateBased and FailurePropagation, see the guide related to dependability support).

8.3.2.2 Composite Structure Diagram

Composite Structure Diagrams are used in the Extra-functional View to model real time, state based analysis and failure propagation information for the available ports and parts/instances **in the context of the classifier subject of the composite diagram**. For an alternative and more powerful way to model extra functional properties at instance level the Instance View can be used in place of the composite diagram.

A subset of the modeling elements is available from the palette for Composite Structure Diagrams in the Extra-Functional View, as illustrated in Figure 12, while other stereotypes concerning extra functional properties for state based analysis are available in the Profile tab of the Papyrus Properties view.

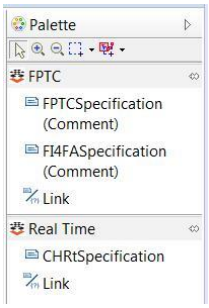

Figure 12 Extra-functional View Composite Structure Diagram Palette

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.

For info on modeling Real Time properties using the <<CHRtSpecification>> Stereotype see Section 10.2.

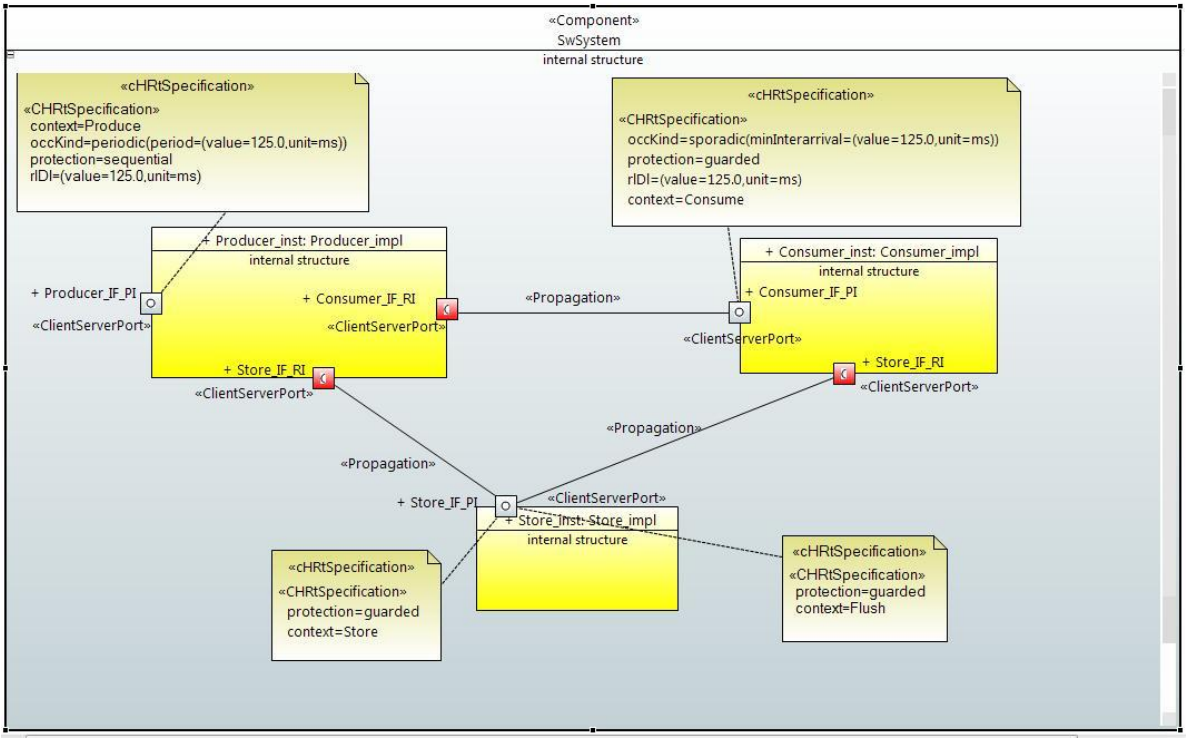

Figure 13 Composite Structure Diagram in Extra-functional View modelling Real Time information

8.3.2.3 State Machine Diagram

State Machine Diagrams are used in the Extra-functional View to model dependability ErrorModel for a given ComponentImpl.

8.3.2.4 Activity diagram

Activity diagrams are not used in the Extra-functional View.

8.4 Deployment View

8.4.1 Class and Composite Structure diagram

Class diagrams are used to model the single or multicore CH_HWProcessors (see Figure 14).

Composite Structure diagrams are used to model the platform system model (see Figure 15), to then enable the allocation of software components instances.

See guide about schedulability analysis for further details.

8.4.2 Modeling Memory Partitions

MemoryPartition (from MARTE) represents a virtual address space which insures that each concurrent resource associated to a specific memory partition can only access and change its own memory space.

MemoryPartitions are created in the deployment view, first at type level in a class diagram, together with the HW types, in particular processors and physical memories, and then at instance level using the composite diagram; see **Error! Reference source not found.** and **Error! Reference source not found.** as example.

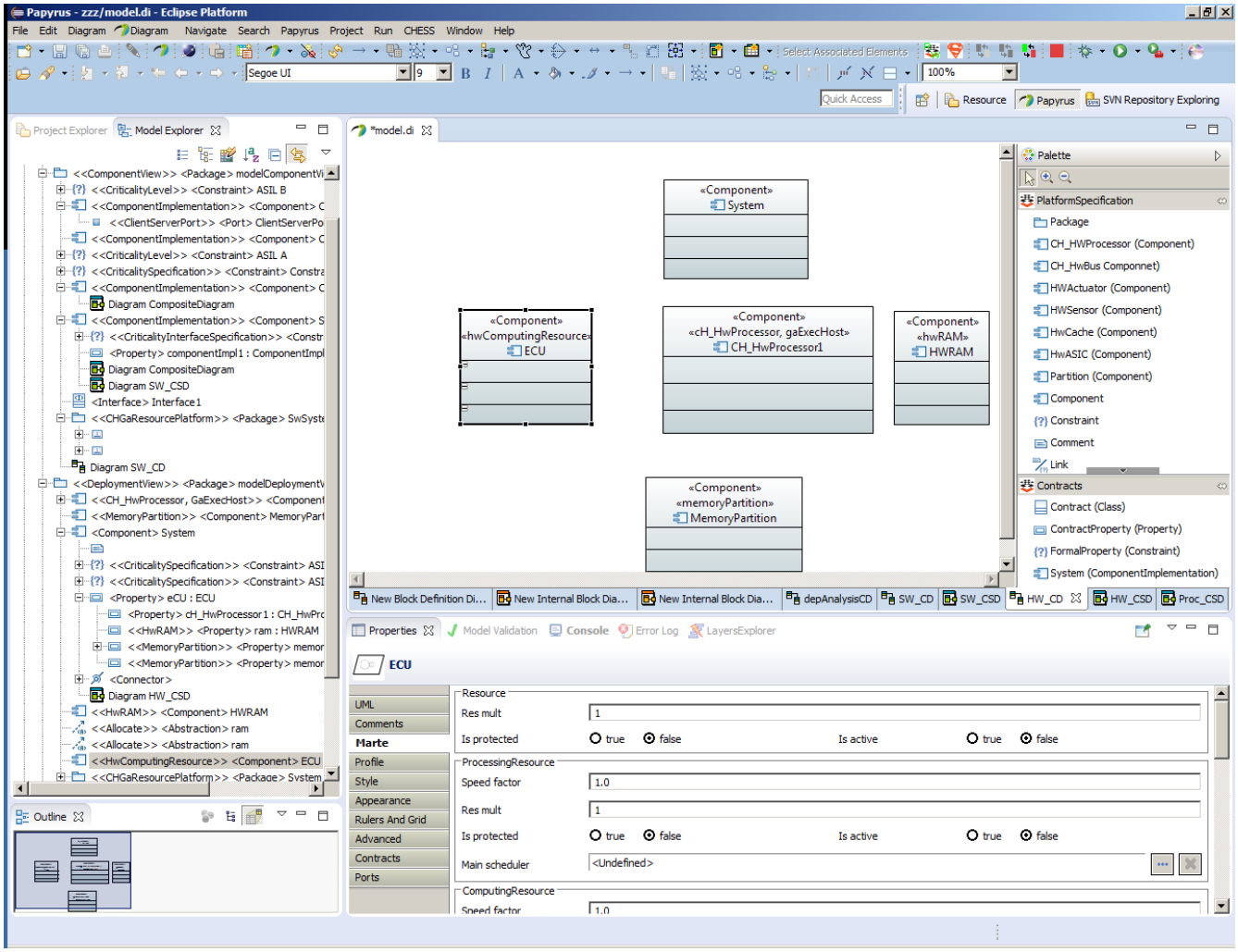

Figure 14: modelling HW types

As an example, in **Error! Reference source not found.** the MARTE HwComputingResource stereotype (as extension of Component) is used to model a HW entity (ECU) composed by one CPU (CH_HWProcessor) and RAM (HWRAM.

The MemoryPartition is instantiated through the composite structure diagram (see **Error! Reference source not found.**), together with the HW components.

⚠ *Memory Partitions instances must be modeled inside a container which also contains the associated Processor instance.*

To allow the specification of MemoryPartition properties at instance level, in the composite diagram the MemoryPartition stereotype has to be manually applied to the properties representing the MemoryPartitions.

MemoryPartition instances are bound to the given HW memory through the Allocate MARTE relationships.

The MemoryPartition stereotype comes with the MemorySizeFootprint property which can be used to set the size of the partition, in particular the percentage of the HW memory which is reserved. The MemorySizeFootprint property can be specified by creating an OpaqueExpression (e.g. named *size*) and then using natural language to provide its value (see bottom part of **Error! Reference source not found.**).

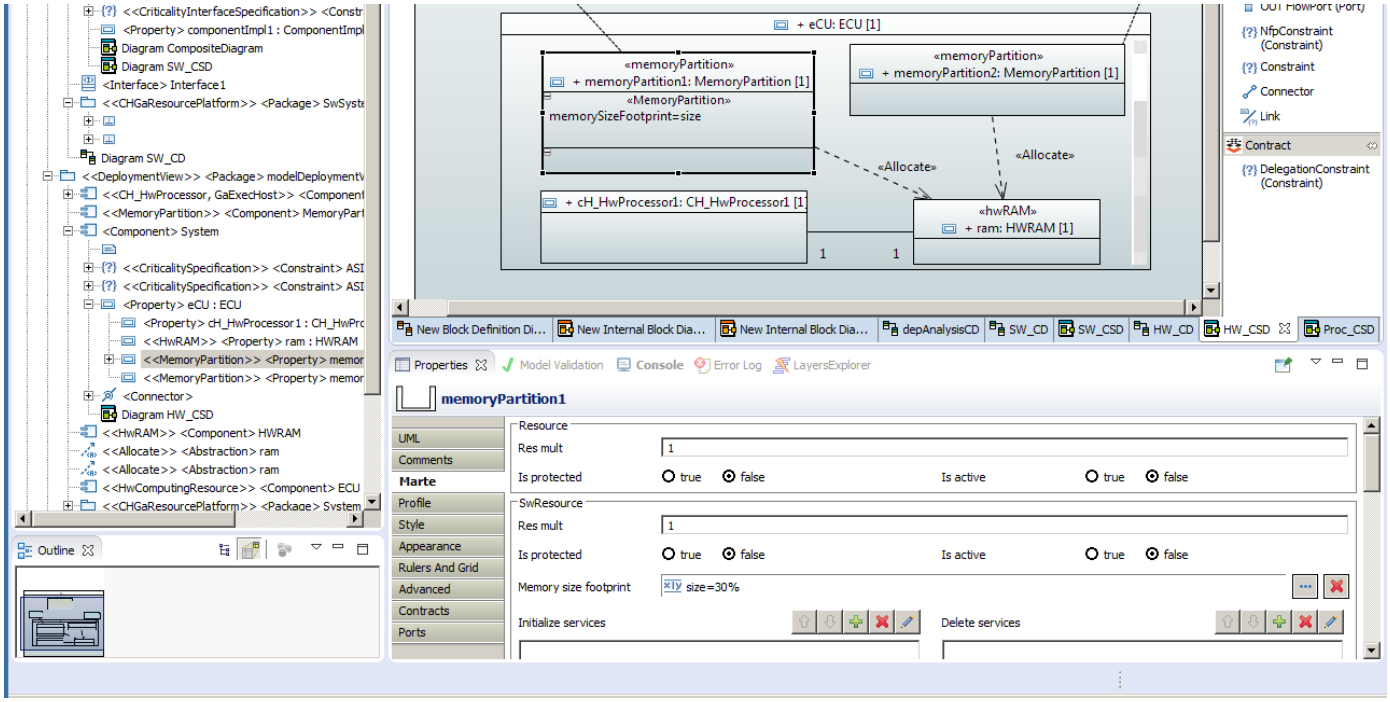

Figure 15: HW instances, MemoryPartition instances and allocations

MemoryPartition component (not instances but classifier) can be decorated with criticality level (e.g. ASIL) by using the CriticalitySpecification entity (see guide about support for modelling criticalities).

Regarding the SW to HW allocation: SW components are allocated to cores. In case of MemoryPartitions associated to the processor the proper allocation of the SW component instances to the MemoryPartition instances (of the same HW computing resource) is not explicitly modelled; it is automatically derived according to the criticalities of the SW components (typing the instances) and the criticality of the MemoryPartitions.

8.5 Analysis View

8.5.1 Dependability View

8.5.1.1 Class diagram

Class diagrams in the Dependability View are used to model dependability analysis contexts, as depicted in Figure 16. See section 10.3 for details on the dependability analysis.

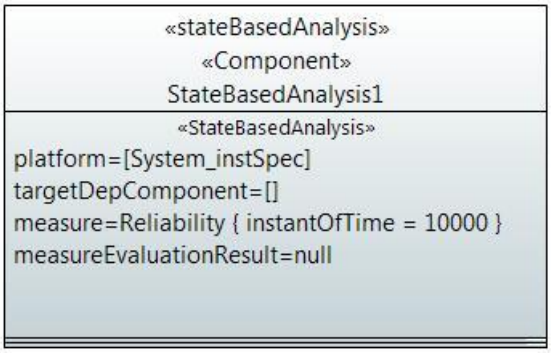

Figure 16 Class Diagram to model dependability concerns in the Analysis View

8.5.2 RT Analysis View

Class diagrams in the RTAnalysisView are used to model timing analysis contexts. See the "CHESS Software Development Guide" for further details related to schedulability analysis.

8.6 Instance View

Hardware and Software Instances owned by a given component in CHESS can be modeled in the Deployment and ComponentView respectively through the Composite Structure Diagram, i.e. through Properties. Each Property comes with its ports, i.e. the ports defined for the Component which types the Property itself. Properties can be connected together through the ports by using the Connector.

UML provides a dedicated set of constructs which can be used to model instances and their properties, basically the InstanceSpecification and Slot; this support allows to overcome some limitation that can be encountered while using Composite Structure Diagrams for the modeling of instances, in particular hierarchical instances with extra functional annotation. However there is not a dedicated UML diagram that can be used to properly work with them to easily model instances of components, ports and connections.

The CHESS toolset allows to automatically derive the InstanceSpecifications and Slot entities by starting from a root component. In particular each Property and Connector is mapped onto a dedicated InstanceSpecification, while Ports are mapped onto Slot. Extra functional information (if available, e.g. modelled in the composite diagram) is attached to InstanceSpecifications and Slots.

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 17) and for the HW system (Figure 18). 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.

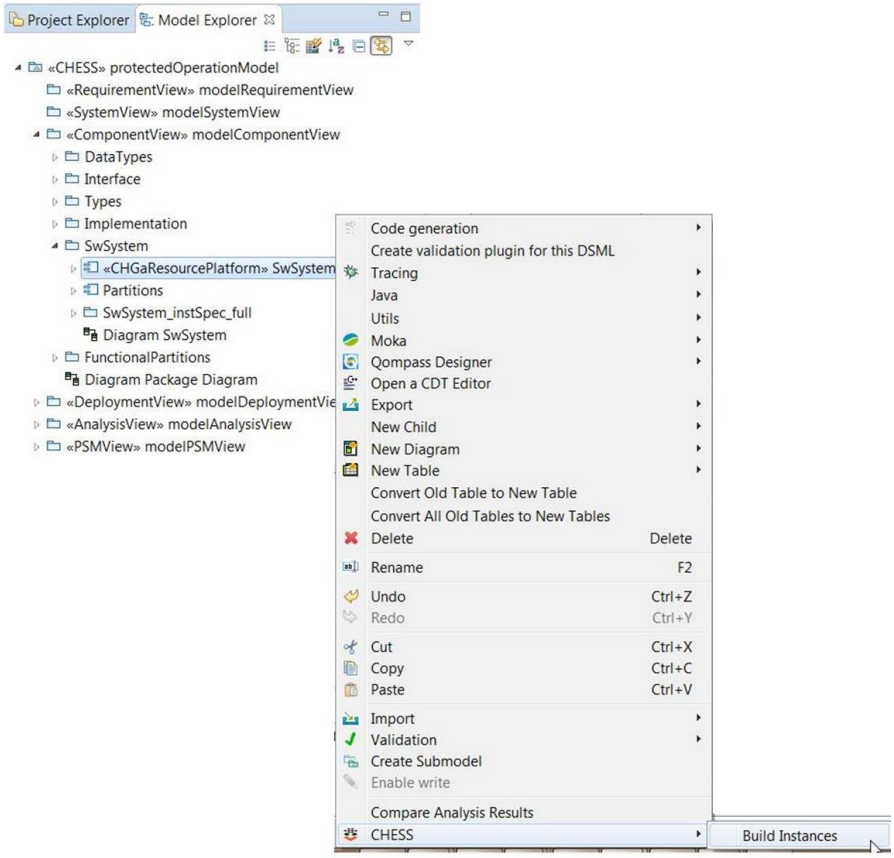

Figure 17 CHESS Build Instances command for Sw System

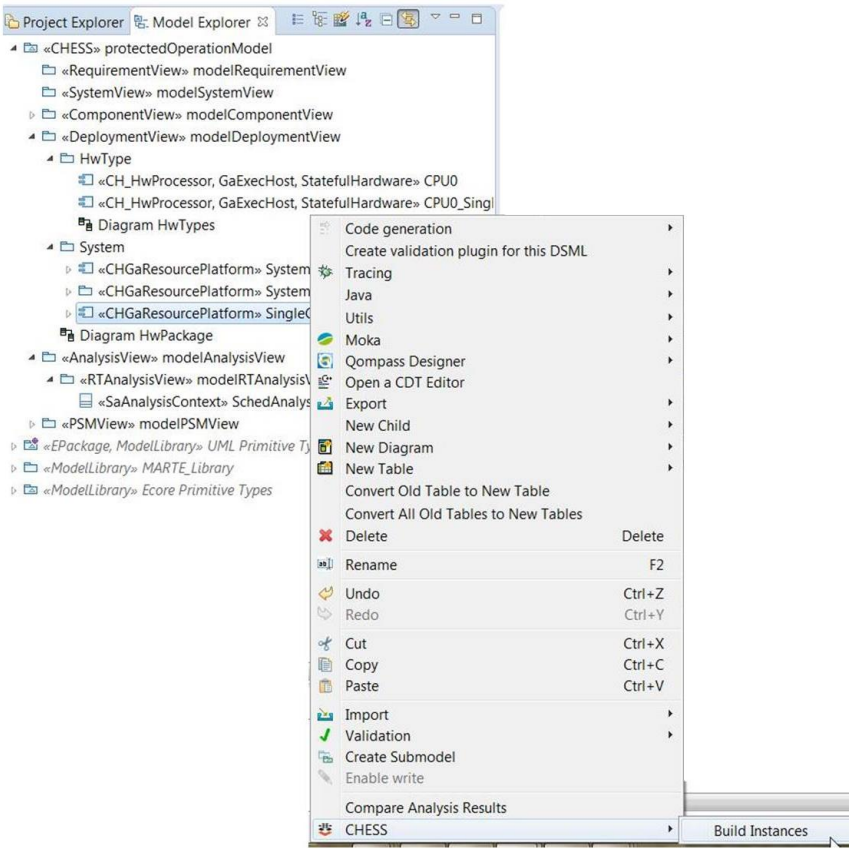

Figure 18 CHESS Build Instances command for HW System

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.

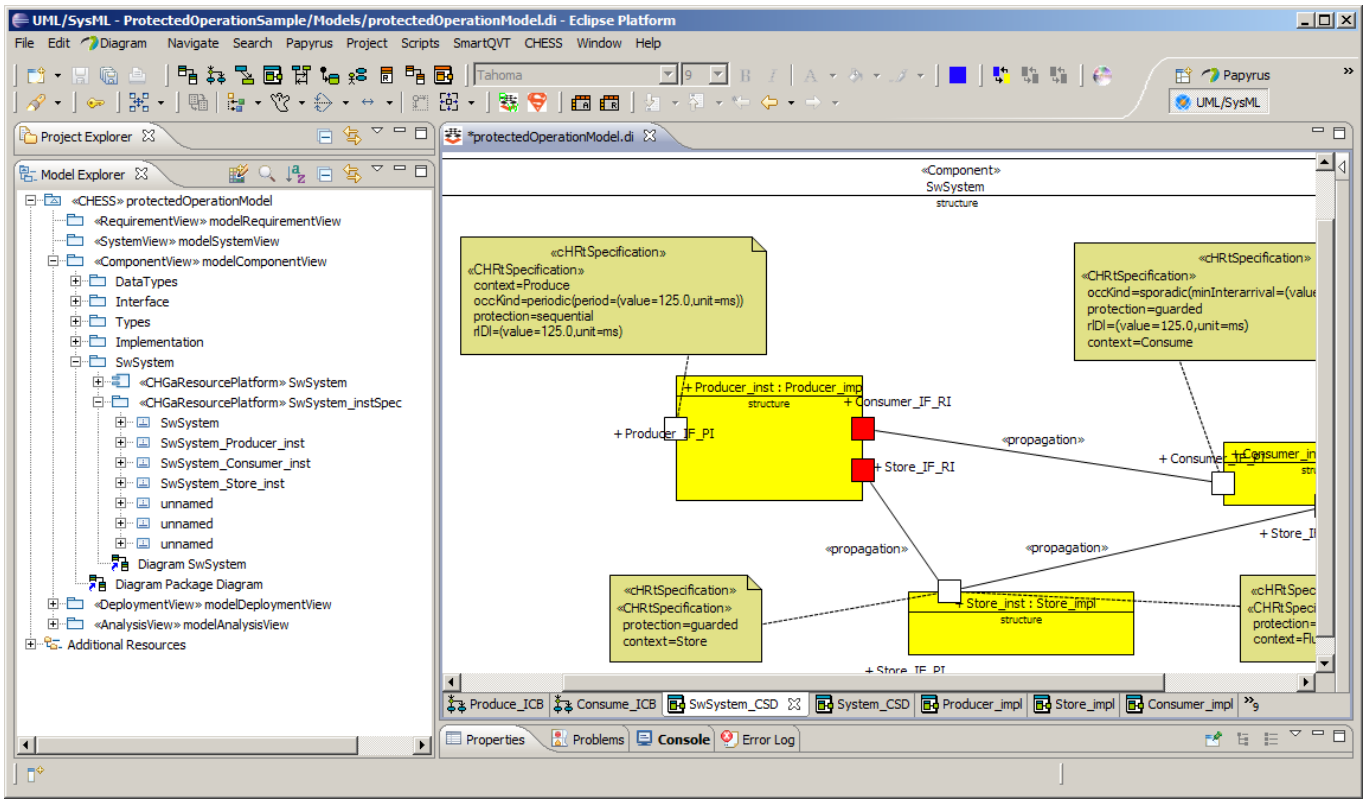

Figure 19: Instances specifications

9 Model validation

Model validation is supported by the CHESS toolset to enforce the methodology constraints.

9.1 Validate Core Constraints

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

9.2 Configuring the CHESS model validation features

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

10 Model-based Analysis

10.1 The Analysis Context

CHESS uses and extend the support given by MARTE for the specification of the analysis context; the analysis context is a modelling construct which allows to specify all the input needed to run a given analysis and in particular it stores the reference to the entities of the design to be considered for the given analysis.

In CHESS the entities of the design which can be subject of the analysis are SW and/or HW components **instances**, together with their declared behaviour (if provided), relationships and allocations.

Basically CHESS uses the MARTE SaAnalysisContext and introduces specializations of the MARTE GaAnalysisContext (e.g. StateBasedAnalysis, FailurePropagationAnalysis) construct to allow the specification of the input needed to run the analysis. In CHESS the set of instances subject of the given analysis must be grouped under packages stereotyped as CHGaResourcePlatform (automatically built using the BuildInstance command). CHGaResourcePlatform extends the MARTE GarResourcePlatform stereotype, in fact the latter cannot be used to group resource instances.

Currently, the following analysis context can be created in the AnalysisView (or in its sub-views):

- SaAnalysisContext : needed to run schedulability and end-to-end response time analysis,
- StateBasedAnalysis (under DendabilityAnalysisView): needed to run state based analysis
- FailurePropagationAnalysis (under DendabilityAnalysisView): needed to run failure propagation analysis (FPTC and FI4FA).

More information about the aforementioned analysis is provided in the following sections.

For each analysis context a link to the relevant CHGaResourcePlatforms to be considered in the analysis must be set through the *platform* attribute: the SW platform instance specification and the HW Deployment platform instance specification must both be specified.

Different analysis contexts can be defined and saved in the CHESS model, allowing to perform real time analysis on different deployment configurations and then to compare the analysis results, as described in the following.

10.2 Timing Analysis

The following timing analysis are supported via integration with the MAST analysis tool at software component development level:

- Schedulability Analysis
- End2End Response Time Analysis

See the “CHESS software development guide” for a detailed timing analysis guide.

10.3 Dependability Analysis

The following dependability analyses are supported:

- State-based quantitative analysis (see the “CHESS dependability guide”)

- Failure Logic Analysis (FLA) based upon FPTC - Fault Propagation and Transformation Calculus, to perform qualitative failure propagation analysis (see the “CHESS dependability guide”).
- FI4FA - Formalism for Incompletion, Inconsistency, Interference and Impermanence Failures (see the “CHESS dependability guide”).
- Model-based safety analysis, including fault injection, fault trees and FMEA tables generation (see the “CHESS guide for contract-based analysis, model checking, and safety analysis”)

10.4 Contract-based analysis

The following contract-based analyses are supported (see the “CHESS guide for contract-based analysis, model checking, and safety analysis”):

- Contract refinement verification
- Contract-based compositional verification of state machines
- Contracts validation
- Contract-based safety analysis

10.5 Analysis of Reliability using SAN Models

CHESS supports modelling of safety and security concerns and automated transformations to SAN models for reliability analysis with the MOBIUS tool. See the “CHESS-MOBIUS Integration Guide” for details.

11 Architectural Patterns

The CHESS tool includes a library of design patterns to be instantiated in a model. It is also possible to create new patterns and so different libraries.

See the “CHESS Architectural Patterns Guide” for details.

12 Ada infrastructural code generation

CHESS allows Ada code generation starting from the ComponentView.

For details see http://chess-project.org/page/training.

13 Runtime Monitoring

The CHESS tool includes trace analysis and back propagation support.

For details see the “CHESS Runtime Monitoring Guide”.

14 References

[1] B. Gallina and Z. Haider, A. Carlsson, S. Mazzini S. Puri, “Multi‐concern Dependability‐centered Assurance for Space Systems via ConcertoFLA”, International Conference on Reliable Software Technologies- Ada-Europe 2018, Lisbon, June 2018

[2] Mazzini S., Puri S., A. Russino, “Fitting the CHESS approach to the AUTOSAR development flow”, Ada User Journal, Vol. 37, Number 3, September 2016.

[3] Mazzini S., Favaro J., L. Baracchi, “A model-based approach for the development of critical space systems”, 2015 IEEE Intelligent Transportation Systems Conference (ITSC 2015), 3rd IEEE International Workshop on Metrology for Aerospace Conference, Florence, June 2016.

[4] Mazzini S., J. Favaro, S. Puri, L. Baracchi., “CHESS: an open source methodology and toolset for the development of critical systems”, 2nd International Workshop on Open Source Software for Model Driven Engineering (OSS4MDE), Saint-Malo, October 2016

[5] L.Baracchi, S.Mazzini, S.Puri, T.Vardanega: “Lessons Learned in a Journey Toward Correct-by-Construction Model-Based Development”, Reliable Software Technologies – Ada-Europe 2016 Volume 9695 of the series Lecture Notes in Computer Science pp 113-128, 31 May 2016

[6] Sljivo I., Gallina B., Carlson J., Hansson H., Puri S., 2016. A Method to Generate Reusable Safety Case Argument-Fragments from Compositional Safety Analysis. Journal of Systems and Software: Special Issue on Software Reuse, 131, pp. 570-590

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

[8] Andrea Baldovin · Alessandro Zovi · Geoffrey Nelissen · Stefano Puri: “ The CONCERTO Methodology for Model-Based Development of Avionics Software”, 20th International Conference on Reliable Software Technologies - Ada-Europe 2015; 06/2015

[9] Sljivo I., Gallina B., Carlson J., Hansson H., Puri S.: “A Method to Generate Reusable Safety Case Fragments from Compositional Safety Analysis”, 14th International Conference on Software Reuse, 4-6 January 2015.

[10] Baracchi L., Cimatti A., Garcia G., Mazzini S., Puri S., Tonetta S., “Requirements refinement and component reuse: the FoReVer contract-based approach”, in Bagnato A., Quadri I. R., Rossi M., Indrusiak l. S., Editors “Industry and Research Perspectives on Embedded System Design”, IGI Global, 2014.

[11] Cicchetti A., Ciccozzi F., Mazzini S., Puri S., Panunzio M., Zovi A., “CHESS: a Model-Driven Engineering Tool Environment for Aiding the Development of Complex Industrial Systems”, ASE2012, September 3-7, 2012, Essen, Germany

[12] Mazzini S., Puri S., Veran G., Vardanega T., Panunzio M., Santamaria C., Zovi A., “Model-Driven and Component-Based Engineering with the CHESS Methodology”, DASIA 2011 - Malta, 19 May 2011

[13] Cicchetti A., Ciccozzi F., Mazzini S., Puri S., Panunzio M.,Zovi A., Vardanega T.,“ New Ideas and Emerging Results Track: Tackling Industrial Challenges in Model-Driven Engineering of Complex Systems”, Proc. of International Conference on Software Engineering (ICSE), Honolulu, May 2011.