Composition with Guarantees for High-integrity Embedded Software Components Assembly

Project Number 216682

CHESS Profile Specification V 1.2

11 January 2012

Public Distribution

Project Partners: **Aicas, AONIX, Atos Origin, CNRI-ISTI, Enea, Ericsson, Fraunhofer, FZI, GMV Aerospace & Defence, INRIA, Intecs, Italcertifier, Maelardalens University, Thales Alenia Space, Thales Communications, The Open Group, University of Padova , University Polytechnic of Madrid**

**** Including University of Florence (sub-contractor)***

DOCUMENT CONTROL

Version	Status	Date
0.1	Original input from D2.2	
0.2	Started D2.3 and D.3.2 alignment.	11 February 2011
0.9.2	CHESS 1.2 release	8 May 2011
0.9.3	CHESS 1.3 release (PIM support)	4 July 2011
1.0	Implemented by the CHESS 2.0 toolset release	3 October 2011
1.0.1	Text editing	21 October 2011
1.1	Added FI4FA support, stereotypes for Simulation Based Analysis, comments about hardware resource usage small (released with CHESS 2.1 toolset)	25 November 2011
1.2	Added FI4FAAnalysis stereotype, DataTypeAssign, ErrorModelAssign Added extended data types. Updated Simulation-Based Timing Analysis section. Aligned with CHESS toolset v3.0	11 January 2012

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

This document addresses the specification of the CHESS ML which objective is to allow the definition of platform independent models (PIM), platform specific models (PSM) and analysis models according to the CHESS methodology.

CHESS ML is defined as a *collection-extension of subsets of standard OMG languages* (UML, MARTE, SysML); for its definition modelling features already available from other methodologies and languages (e.g. HRT-UML/RCM, LwCCM) have taken into account.

Figure 1-1 gives a conceptual view about the CHESS ML dependencies.

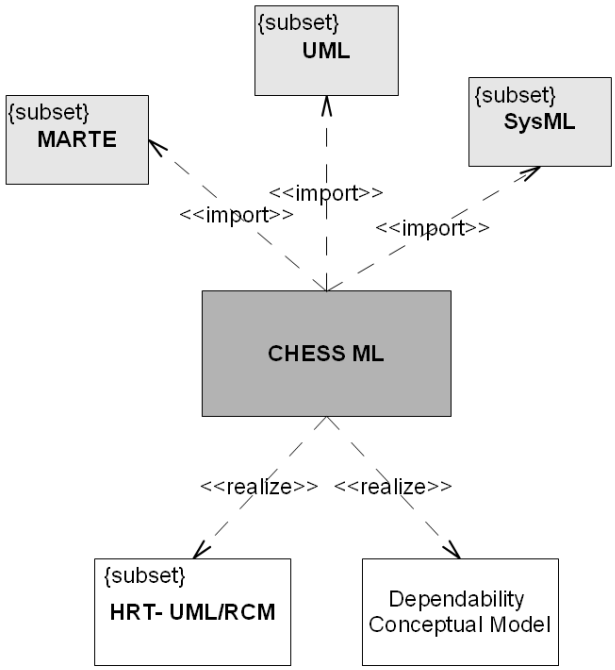

Figure 1-1: CHESS ML dependencies

This specification replaces the one appearing in deliverable D2-3.

The current specification is under finalization considering the WP3 and WP4 analysis on-going implementation and the results coming from the CHESS use cases implementation activities performed in WP6.

CHESS profile is structured in four packages:

- Requirements,

- CHESS PIM,
- CHESS Hardware Platform Specification,
- CHESS PSM,

where each package comes with specific entities directly imported from UML, SysML, MARTE or created as brand new stereotype of the CHESS profile.

In the following sections details about these packages are provided.

2 REQUIREMENT

This package is defined by re-using the entities available in SysML::Requirement package.

2.1 ENTITIES

2.1.1 Requirement

From SysML::Requirements

2.1.2 Derived Reqt

From SysML::Requirements

2.1.3 Satisfy

From SysML::Requirements

3 CHESS PIM

3.1 CORE

3.1.1 Stereotypes

3.1.1.1 CHESS

CHESS represent a model compliant with the CHESS methodology.

Extension:
UML ::Model

Attributes
None

Associations:
None

Constraints:
[1] May have RequirementView, ComponentView, DeploymentView and AnalysisView as owned members.

3.1.1.2 CHGaResourcePlatform

Extends the MARTE::GQAM::GaResourcePlatform and applies to InstanceSpecification. Allows to specify an instance specification as resource platform to be considered in GaAnalysisContext.

Extension:
MARTE ::GQAM::GaResourcePlatform, UML::InstanceSpecification

Attributes
None

Associations:
None

Constraints:
None

3.2 VIEWS

This package implements the CHESS views as defined in CHESS deliverable D2.3.

3.2.1 Stereotypes

3.2.1.1 Requirement View

Requirement view is used to model system requirements in a CHESS model.

Extension:
Package

Attributes
None

Associations:
None

Constraints:
None.

3.2.1.2 Component View

Component view is used to model data types and software components according to the CHESS component model definition. ComponentView is the result of the application of two sub.views: *FunctionalView* and *ExtraFunctionalView*.

Extension:
Package

Attributes
None

Associations:

None

Constraints:
[1] Entities allowed to be edited in this view through the FunctionalView are: Package, ComponentType, ComponentImplementation, Realization, ClientServerPort, FlowPort, Property, Operation, Interface, Connector, DataType, InterfaceRealization, Dependency, Enumeration, EnumerationLiteral, InstanceSpecification, Slot, StateMachine, ModeBehavior, Mode, ModeTransition, Configuration, Activity, Interaction. (see Section 3.3 for further restriction upon these entities).

[2] Entities allowed to be edited in this view through the ExtraFunctionalView are:

- For predictability: CHRtSpecification (see section 3.4.1.1)
- For dependability: the entities described in section 3.6.

3.2.1.3 Deployment View

Deployment View allows to model the hardware platform and software to hardware allocation. It owns the DependabilityView as sub view.

Extension:
Package

Attributes
assignList : Assign[0..*]
References all the Assign stereotypes used to model software to hardware component/port instances allocation. CHESS profile does not constraints the kind of entity(s) which can own these Assign.

Associations:
None

Constraints:
[1] Entities allowed to be edited in this view are:

- the ones defined in the Hardware Platform Specification package (section 4).
- MARTE::Allocate::Assign, to model allocations.

[2] Entities allowed to be edited through the Dependability View are described in section 3.6

3.2.1.4 Real Time Analysis View

Real Time Analysis View allows to model real time analysis contexts.

Usage to be defined.

Extension:
Package

Attributes
None

Associations:
None

Constraints:
TBD.

3.2.1.5 Dependability Analysis View

Allows to model dependability analysis, e.g. StateBasedAnalysis (see 3.6.5.14).

Extension:
Package

Attributes
None

Associations:
None

Constraints:
TBD.

3.3 COMPONENT MODEL

The component model package defines a set of concepts that can be used to model CHESS software component model as described in D2.3.

3.3.1 Entities

3.3.1.1 Package

From UML.

Additional Constraints:

- *PackageMerge*. Not addressed in CHESS ML. Note: merge of packages requires definition of sets of transformations. From SysML specification: "*Combining packages that have the same named elements, resulting in merged definitions of the same names, could cause confusion in user models and adds no inherent modelling capability".*

3.3.1.2 Realization

From UML.

Additional Constraints:
From ComponentImplementation to ComponentType.

3.3.1.3 ClientServerPort

From MARTE::GCM.

Additional Constraints:

[1] A request arriving at a delegating port can have only one delegated port able to handle the request.

[2] A ClientServerPort can provide (trough *provInterface* attribute) or require interfaces (through *reqInterfaces* attribute); it cannot provide and require interfaces at the same time.

[3] Multiplicity has to be 1.

[4] At instance level required port can have only one connector attached.

3.3.1.4 FlowPort

From MARTE::GCM.

3.3.1.5 Property

From UML.

Used to model component attributes and internal parts for composite component.

3.3.1.6 Operation

From UML.

Operation's method can be modeled through activity diagram or by using UAL.

Additional Constraints:

[1]*Operation::raisedException*: the modelling of exception in CHESS ML is under investigation.

[2] *Operation::preCondition.* Excluded from the CHESS ML.

[3]*Operation::postCondition.* Excluded from the CHESS ML.

[4]*Operation::bodyCondition.* Excluded from the CHESS ML.

3.3.1.7 Interface

From UML.

3.3.1.8 Connector

From UML.

Additional Constraints:

Connector maps the connector entity defined in the CHESS component model. Semantic variation point regarding what makes connectable elements compatible needs to be fixed.

[1] It can connect ports only.

3.3.1.9 PrimitiveType

From UML.

3.3.1.10 DataType

From UML.

See section 3.3.3 for additional information about modeling data types in CHESS.

3.3.1.11 InterfaceRealization

From UML.

Not mandatory in CHESS.

3.3.1.12 Dependency

From UML.

It can be used to model the interface required by a ComponentType or ComponentImpl: not mandatory in CHESS.

3.3.1.13 Enumeration

From UML.

3.3.1.14 EnumerationLiteral

From UML.

3.3.1.15 InstanceSpecification

From UML.

3.3.1.16 Slot

FromUML

3.3.1.17 StateMachine

From UML.

Additional Constraints:

StateMachine in CHESS ML can only be used to model functional behaviour of ComponentImplementation; usage of state chart for hardware component has to be investigated.

The following *CHESS state machine profile* is needed to allow (functional) code generation starting from state machine:

- *Pseudostate*: the following kind of pseudostate are included in CHESS ML:
  - *Initial*

- *Final*

- *Region*: only one region can exist in the owning context.

- *State*: a State can own at least one region. Since orthogonal states are not supported, the *isOrthogonal* attribute will always be false. The *doActivity* behavioural specification is not supported and neither is the *stateInvariant* constraint.

  The body of the entry and exit behaviour must be written by using the UAL action language syntax.

  Concerning the semantic variation point in composite states where no initial pseudostate exists, the state machine will stay in the composite state without entering its region or any of the regions substates. To enter the region of the composite state a transition originating from the composite state border has to be taken by the triggering of that transition.

- *StateMachine*: orthogonal state machines are not supported and the number of regions owned directly by a state machine is therefore limited to one.

- *Transition*: all semantics related to orthogonal state execution is not supported and therefore invalid in CHESS ML.

  The effect of the transition, i.e. the (optional) behaviour to be performed when the transition fires, must be written by using the UAL action language syntax.

Note: constraints about state machine redefinitions (i.e. how it is possible to apply generalization between state machines) are not provided in the current version of this specification.

DeferredEvent are not supported.

3.3.1.18 ModeBehavior

From MARTE::CoreElements: representes a dedicated state machine to model operational modes for a component implementation.

3.3.1.19 Mode

From MARTE::CoreElements: represents a state in a state machine representing an operational mode for a component implementation.

3.3.1.20 ModeTransition

From MARTE::CoreElements.

3.3.1.21 Configuration

From MARTE::CoreElements; allows to represent a scenario made of a set of component implementation instances which are available in a given mode.

3.3.1.22 Activity

From UML.

It is used in CHESS to model operation implementation (i.e. the operation's method in the UML meta-model), in particular intra-component bindings, i.e. if a given operation invokes a required operation, how many times etc.

The use case is the following:

1) the modeler wants to specify intra-component bindings for an operation Op of a given ComponentImpl C1
2) he/she creates the Activity diagram in the ComponentView as owned behaviour of C1
3) the modeler sets the activity as the method of the operation Op
4) the modeler use CallOperationAction to set
    a. the operation called
    b. the required port through which the operation is called
5) The modeler uses initial and final activity to complete the activity diagram

Additional Constraints:

[1] activity has to be owned by a ComponentImplementation.

[2] only action CallOperationAction can be used

3.3.1.23 Interaction

From UML.

Allows to model collaboration scenarios between component implementation instances through sequence diagrams.

Under evaluation in CHESS.

3.3.1.24 Assign

From MARTE::Allocate.

It is used in CHESS to model the allocation of component implementation instance to hardware instance.

Additional Constraints:

[1] "from" has to be an InstanceSpecification typed as ComponentImplementation.

[2] “to” has to be an InstanceSpecification typed as hardware component (see section 4).

3.3.2 Stereotypes

3.3.2.1 ComponentType

Maps the component type notion of the CHESS component model.

Extension:
Component

Attributes

None

Associations:
None

Constraints:
[1] It cannot own behaviours.

3.3.2.2 ComponentImplementation

Maps the component implementation notion of the CHESS component model. It can have requirements associated representing technical budgets.

Extension:
Component

Attributes

language : String [0..1]

OS : String [0..1]

sourceCodeLocation : String [0..*]

Associations:
None

Constraints:
None

3.3.3 Working with extended data types

In addition to DataType, several construct are made available in CHESS to model extended data types.

Primitives types can be modelled through PrimitiveType UML construct.

In a given component **Constants** data can be modeled by selecting the 'read-only' attribute of the Property representing the data.

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

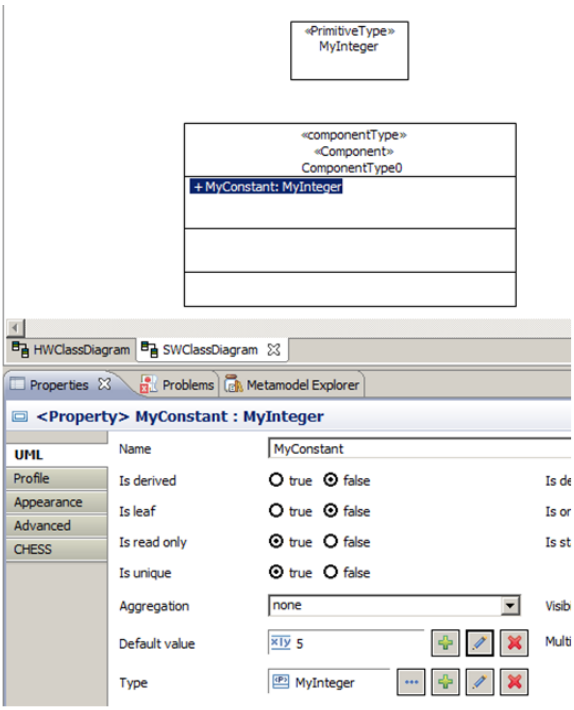

Figure 3-1: Constant property with default value

Data ranges can be modeled by using MARTE::VSL::DataTypes::BoundedSubtype stereotype:

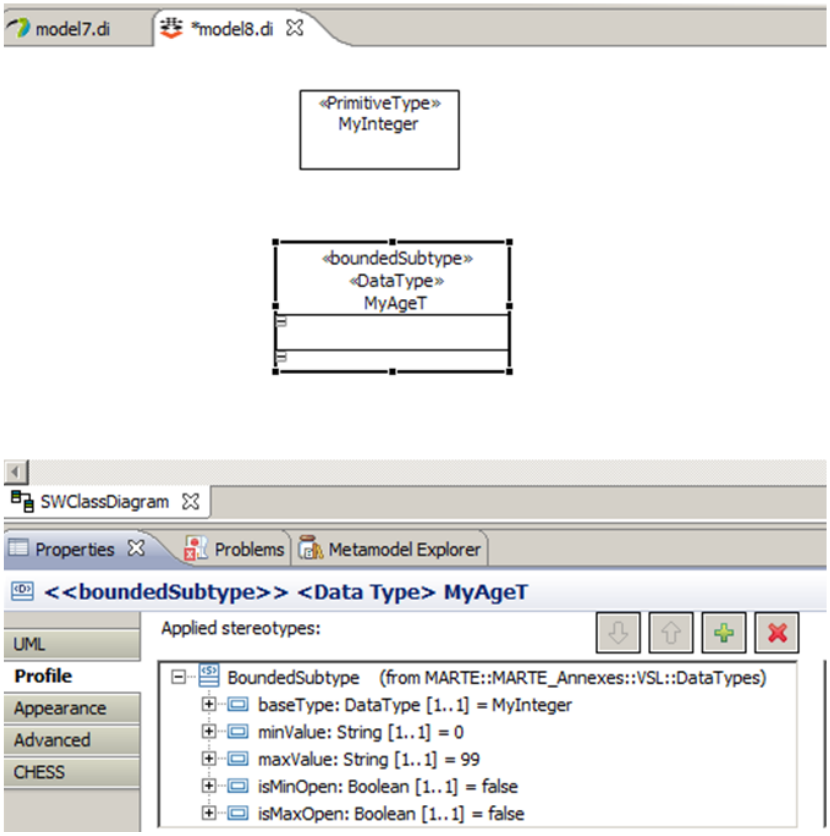

Figure 3-2: Range data type

Arrays can be modeled by using CollectionType from MARTE VSL.

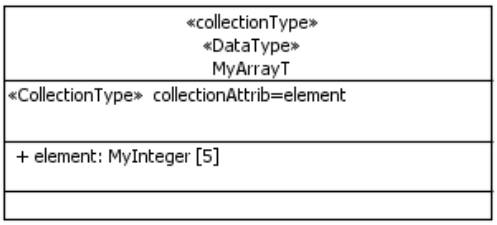

Figure 3-3: Array data type

Structures can be modeled by using TupleType from MARTE VSL.

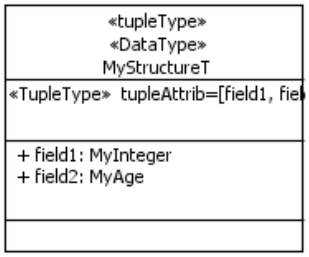

Figure 3-4: Structure data type

3.4 CONCURRENCY

CHESS address real time properties specification at PIM level by annotating operations which are provided/required through *component-instances ports*. This is a practice derived from HRT-UML/RCM methodology and partially supported also by MARTE.

Real-time information is provided at functional instance level only.

So CHESS profile extends MARTE support in order to allow modeling of PIM real time information at instance level through port&operation annotation. According to CHESS methodology these PIM real time information can be automatically transformed into PSM real time specification.

3.4.1 Stereotypes

3.4.1.1 CHRtSpecification

This is the core construct in CHESS to provide concurrent information at PIM level. Used in composite structure diagram allows to model qualitative and quantitative real time attributes for component implementation instances.

Extension:
UML::Comment

Attributes

- partWithPort: Property [0..1]
- occKind : ArrivalPattern [0..1]
- protection : CallConcurrencyKind [0..1]
- relativePriority : NFP_Integer [0..1]
- ceiling : NFP_Integer [0..1]

  (in MARTE available from GRM::MutualExclusionRes)
- WCET : NFP_Real[0..1]

(in MARTE available from GQAM::ResourceUsage)

- localWCET : NFP_Real[0..1]
- rlDl : NFP_Duration
- respT : NFP_Duration [0..1]

  (in MARTE available from GQAM::GaScenario)
- blockT : NFP_Duration [0..1]

  (in MARTE available from by SaSchedObs)
- memorySizeFootprint: NFP_DataSize [0..1]

  (in MARTE available from SwResource)
- stackSize : NFP_DataSize [0..1]

  (in MARTE available from SwConcurrentResource)
- heapSize : NFP_DataSize [0..1]

  (in MARTE available from SwConcurrentResource)
- context : BehavioralFeature [1..1]

Associations:
None

Constraints:
[1] It has to be applied at instance level. The referred instance has to be typed as ComponentImplementation.

[2] occKind can be Periodic, Sporadic or Bursty

[3] protection can be ‘guarded’ or ‘concurrent’

3.4.1.2 CHRtPortSlot

Allows to model a slot representing an instance of a port having CHRtSpecification information attached. This is useful in case of multiple structured instances where the part-with-port relationships in the model are not enough.

Extension:
UML::Slot

Attributes

- rtSpec : CHRtSpecificaiton[0..*]

Associations:
None

Constraints:
None

3.4.1.3 Ga_Step (TBC)

From MARTE::GQAM.

Used to model message sizes in sequence diagrams.

Additional Constraints:
Used to model message sizes only.

3.5 PREDICTABILITY

This package addresses the CHESS predictability analysis support. It is structured in sub-packages.

3.5.1 Schedulability analysis

3.5.1.1 Stereotypes

SchedulabilityAnalysis (TBC)

SchedulabilityAnalysis collects relevant qualitative and quantitative information for performing schedulability analysis. Schedulability analysis tool can use SchedulabilityAnalysis to extract all the information that it needs.

Extensions:
MARTE::SAM:SaAnalysisContext

Attributes
None

Associations:
None

Constraints:
[1] The platform attribute (from GaAnalysisContext) has to refer the system to be analyzed, i.e. typically a root component owning hardware instance, component implementation instances and allocations.

3.5.2 Deployment configuration

The hardware baseline is represented through the deployment view. SysML hardware description is imported in the CHESS-ML model, and then dependencies with the CHESS deployment view entities are traced.

Below stereotypes for this package are listed.

3.5.2.1 Stereotypes

CH_HwBus

Extensions:
MARTE::DRM::HRM::HwBus

Attributes

utilization : NFP_Real [0..*] ; derived from schedulability analysis

Commento [s1]: We should use MARTE::GQAM::GaExecHost, agreed for release CHESS v3.1?

Associations:
None

Semantics:

Constraints:
None

CH_HwComputingResource

Extensions:
MARTE::DRM::HRM::HwComputingResource

Attributes

utilization : NFP_Real [0..*] ; derived from schedulability analysis

Commento [s2]: We should use MARTE::GQAM::GaExecHost, agreed for release CHESS v3.1?

Associations:
None

Semantics:

Constraints:
None

SchedulabilityAnalysis

Extensions:
MARTE::SAM:SaAnalysisContext

Attributes

mapping : Assign[*]

resultingPriorities : CHRtSpecification [*]

schedulingStrategy: String [*]

Associations:
None

Constraints:
TBD.

BusConfigurationAnalysis

Extensions:
MARTE::SAM:SaAnalysisContext

Attributes

hwBusPlatform : HWBus[*]

inputFibex: FIBEX [*]

resultingBusConfig: FIBEX [*]

resultingCommLatencies: SaStep[*]

Associations:
None

Constraints:
TBD.

MappingConfigurationAnalysis

A MappingConfigurationAnalysis collects relevant qualitative and quantitative information for performing mapping configuration analysis scenario. An analysis tool can use MappingConfigurationAnalysis to extract all the information that it needs.

Extensions:
MARTE::GQAM:GaAnalysisContext

Attributes

prohibitedAllocation : Allocate[*] (or Assign[*])

Models prohibition of mapping.

fixedAllocation : Allocate[*](or Assign[*])

Represents a fixed mapping decision that is unchangeable.

bottomUpAnalysis : boolean

resultingMapping : Allocate[*]

Associations:
None

Semantics:
The inherited attribute *platform* represents a concrete architecture of hardware and software processing resources.
The inherited attribute *workload* owns the sequence diagram modeling the control flow and communication dependencies between software processes in case of top-down approach. This workload is considered in the top-down approach only.
Only periodic execution, jitter of periodic execution and bursts activation patterns are considered for this analysis; WCET and buffer size are not mandatory.

Constraints:
[1] property *platform* must refer an hardware and software model.
[2] Hardware busses have to stereotyped as FIBEX (see hardware platform specification, section 4)
[3] In case of bottom-up approach, software components which are referred by *platform* must have a SystemC implementation.
[4] For the determination of scheduling priorities and bus configurations end-to-end latencies are mandatory.
[5] Hardware processor has to be stereotyped as CH_HwProcessor.

CH_HwProcessor

Extensions:
MARTE::DRM::HRM::HwProcessor

Attributes

dataType : HWDataType [0..*]

Associations:
None

Semantics:

Constraints:
None

CH_ControlFlow
This is a simplification of the MARTE GaStep stereotype.

Extensions:
UML::ControlFlow

Attributes

rep : Real [0..1], the actual or average number of repetitions of a loop.

prob : Real [0..1], the probability of the flow to be executed (for a conditional execution).

order: Real [0...1], to determine the hierarchy of loops.

compComplex : ComputeComplexity [0..*], the computational complexity

Associations:
None

Semantics:

Constraints:
None

ComputeComplexity

Extensions:
UML::Class

Attributes

swDataType : SWDataType[1]

opCount : OperationCount[1..*]

Associations:
None

Semantics:

Constraints:
None

OperationCount

Extensions:
UML::Class

Attributes

operation : BasicOperation[1]

count : Integer[1]

Associations:
None

Semantics:

Constraints:
None

DataTypeExecution

Extensions:
UML::Class

Attributes

operation: BasicOperation[0..1]

executionCycles: Integer[0..1]

Associations:
None

Semantics:

Constraints:
None

HWDataType

Extensions:
UML::DataType

Attributes

bitLength: Integer[0..1]

signed: Boolean [1]

fixedPoint: Boolean [1]

float: Boolean [1]

execution : DataTypeExecution [0..*]

Associations:
None

Semantics:

Constraints:
None

SWDataType

Extensions:
UML::DataType

Attributes

bitSize: Integer[1]

Associations:
None

Semantics:

Constraints:
This is an abstract class.

FloatSWDataType

Extensions:
SWDataType

Attributes

signed: Boolean[1]

Associations:
None

Semantics:

Constraints:
None.

FixedSWDataType

Extensions:
SWDataType

Attributes

signed: Boolean[1]

Associations:
None

Semantics:

Constraints:
None.

IntegerSWDataType

Extensions:
SWDataType

Attributes

signed: Boolean[1]

Associations:
None

Semantics:

Constraints:
None.

CharSWDataType

Extensions:
SWDataType

Attributes

signed: Boolean[1]

Associations:
None

Semantics:

Constraints:
None.

BooleanSWDataType

Extensions:
SWDataType

Attributes:

None

Associations:
None

Semantics:

Constraints:
None.

DataTypeAssign

Extensions:
MARTE::Alloc::Assign;

Attributes

None

Associations:
None

Semantics:
Allocates software data types to hardware data types.

Constraints:
[1] from property can hold SwDataType entities only.
[2] to property can hold HwDataType entities only.

Scheduling&BusConfigurationAnalysis

A Scheduling&BusConfigurationAnalysis collects relevant qualitative and quantitative information for performing for scheduling and bus configuration parameters determination analysis.

Extensions:
MARTE::GQAM:GaAnalysisContext

Attributes

mapping : Allocate[*] (or Assing[*])

resultingPriorities : ….

resultingBusConfig : FIBEX

resultingCommLatencies : {GaStep.blockT, Connector} [*]

Associations:
None

Semantics:

The inherited attribute *platform* represents a concrete architecture of hardware and software processing resources, where software components are allocated on hardware; as alternative this information can be provided local to the analysis using the mapping attribute, so to allow execution of several analysis based on different allocations strategies.
In case of top-down approach the inherited attribute *workload* owns the sequence diagram modeling the control flow and communication dependencies between software processes

Constraints:
None

BasicOperation

Extensions:
None

Semantics:
BasicOperation is an enumeration of the following literal values:
{add, mul, div, mod, shift}

BasicDataType

Extensions:
None

Semantics:
BasicDataType is an enumeration of the following literal values:
{Signed, Integer, Fixed_Point, Float}

3.5.3 Simulation-Based Timing Analysis

Simulation analysis requires to model functional entities which communicate through data-exchange. Data exchange between structural entities can be modeled in CHESS-ML by using the MARTE flow ports. If these entities are hardware, then they can be modeled in the deployment view.

For what concerns timing and timing constraints to be attached to end-to-end flows, MARTE::GQAM sub-profile and its derivations (SAM and PAM) are used.

CH_RtSpecification is used to retrieve Periodic, Sporadic activation patterns.

Additional stereotypes are introduced in the following section.

3.5.3.1 Entities

SimulationBasedTimingAnalysis

A SimulationBasedTimingAnalysis collects relevant qualitative and quantitative information for performing simulation based timing analysis. An analysis tool can use SimulationBasedTimingAnalysis to extract all the information that it needs.

Extensions:
MARTE::GQAM:GaAnalysisContext

Attributes

None

Associations:
None

Semantics:
The inherited attribute *platform* represents a concrete architecture of hardware and software processing resources.

Constraints:
None

AgeTimingConstraint

Extensions:
MARTE::GQAM:GaLatencyObs

Attributes

None

Associations:
None

Constraints:
None

ReactionConstraint

Extensions:
MARTE::GQAM:GaLatencyObs

Attributes

None

Associations:
None

Constraints:

None

OutputSynchronizationConstraint

Extensions:
MARTE::GQAM:GaLatencyObs

Attributes

- width : NFP_Duration [1]

Associations:
None

Constraints:
None

InputSynchronizationConstraint

Extensions:
MARTE::GQAM:GaLatencyObs

Attributes

- width : NFP_Duration [1]

Associations:
None

Constraints:
None

3.6 DEPENDABILITY

The dependability package defines a set of concepts that can be used to model dependability related information.

3.6.1 Dependable Components

3.6.1.1 Stereotypes

DependableComponent

This is an abstract class.

Extensions:
UML::Component

Attributes

- errorModel : ErrorModel[0..1]

Associations:
None

Constraints:
Applies to ComponentImplementation, Components in the DeploymentView

Propagation

Extensions:
UML::Connector, Comment, InstanceSpecification

Attributes

- prob : NFP_Real[1]
- propDelay : NFP_Duration[1]

Associations:
None

Constraints:
[1] Comment has to be stereotyped as MARTE::Alloc::Assign

3.6.2 Threats And Propagation

Threats and propagations for a given component (software and hardware) are modeled in CHESS through state machines. Following section lists the set of stereotype entities owned by this package.

3.6.2.1 ErrorModel

Extension:
StateMachine (from UML::StateMachine)

Attributes

Associations:
None

Constraints:
[1] ErrorModel state machine can have only states and transitions stereotyped with entities belonging to ThreatsAndPropagation package.

[2] It has to be owned by a ComponentImplementation

3.6.2.2 InternalFault

Extension:
Transition (from UML::StateMachine)

Attributes

- Occurrence : Distribution [0..1]
- permanentProb : NFP_real [1]
- transientDuration : Distribution [1]
- transfFunct : String [0..1] - Represents the transfer function (FMEA).
- property : Property [0..1] – The child component property (instance) from which this internal fault originates.
- childFailure : State [0..1] – Represent the failure mode of a child component which propagates up to the parent originating this internal fault.

Associations:
None

Constraints:
[1] childFailure must be stereotyped as [TBD] and must to be owned by a child comp

3.6.2.3 ExternalFault

Extension:
Transition (from UML::StateMachine)

Attributes

- fromPort : Port[1..*]

  The port of the component owning the transition from which the external fault can origin.

- propagationCondition : String [0..1]

  Specifies the combination of failures (incoming from ports) that are needed to generate the external fault (e.g. “and”, “or”, “2-out-of-3”).

Associations:
None

Constraints:
[1] components instances appearing in the logic expression have to be child of the component owning the ExternalFault

[2] fromPort and propagationCondition are mutually exclusive.

3.6.2.4 ThreatState

Extensions:
State (from UML::StateMachines)

Attributes

- unit : String[0..1] - Unit is related to the probability and describes its physical unit.
- probability : NFP_Real[1]

Associations:
None

Constraints:
[1] The owning state machine must be an ErrorModel

3.6.2.5 Error

Extensions:
ThreatState

Attributes

- vanishingTime : Distribution [0..1]
- type: {transient, permanent} – Type describes whether the fault results in a permanent or a transient error.

Associations:
None

Constraints:

3.6.2.6 ErrorFree (FMEA only)

Extensions:
ThreatState

Attributes

None

Associations:
None

Constraints:

3.6.2.7 UnclassifiedError (FMEA only)

Extension:
ThreatState

Attributes

None
Associations:
None

Constraints:
None

3.6.2.8 FailureMode

Extension:
ThreatState

Attributes

- affectedPorts : Port[0..*]
- type: {transient, permanent} – Describes whether the fault results in a permanent or a transient error.

Associations:
None

Constraints:
[1] *affectedPorts* has to refer ports owned by the classifier owning the FailuireMode.

3.6.2.9 FailureFree (FMEA only)

Extensions:
ThreatState

Attributes

None

Associations:
None

Constraints:

3.6.2.10 UnclassifiedFailure (FMEA only)

Extension:

ThreatState

Attributes

None

Associations:
None

Constraints:
None

3.6.2.11 Propagation

Extension:
Transition (from UML::StateMachine)

Propagation (from CHESS::DependableComponents, merge increment)

Attributes

- prob : NFP_Real [1..1]
- propDelay : NFP_Duration [1..1]
- weight : NFP_Real[0..1]
- transfFunct : String [0..*] - Represents the transfer function (FMEA).

Associations:
None

Constraints:
[1] from Error, ErrorFree and Unclassified to FailureMode only.

3.6.3 SA profile

The CHESS-ML profile imports the SA UML profile (from UPM) to support FMECA and FTA analysis.

Please refer to the SA profile specification for further details about its usage.

3.6.4 FMEA profile

The following stereotypes are introduced in CHESS-ML FMEA profile.

3.6.4.1 ExternalFault

Extension:
ExternalFault (from CHESS::ThreatsPropagation, merge increment)

Attributes

- probability : NFP_Real[1]
- unit : String[0..1] - Unit is related to the probability and describes its physical unit.
- type: {transient, permanent} – Describes whether the fault results in a permanent or a transient error.
- transfFunct : String [0..*] - Represents the transfer function.

Associations:
None

Constraints:
None.

3.6.4.2 ErrorModelAssign

Extensions:
MARTE::Alloc::Assign;

Attributes

None

Associations:
None

Semantics:
Allocates error model to flow port instances.

Constraints:
[1] from property can hold ErrorModel state machines only.
[2] to property can hold FlowPort ports only.

3.6.4.3 FMEAAnalysis

A FMEAAnalysis collects relevant qualitative and quantitative information for performing FMEA analysis. An analysis tool can use FMEAAnalysis to extract all the information that it needs.

Extensions:
MARTE::GQAM:GaAnalysisContext

Attributes

- errorType : Error[0..1]

  An error state within the dependability view. This configures the type of error to inject to the simulation.

- simulationRuns : int[1]

  This value describes the number of simulation runs for the statistically based analysis.

- analysisType : {frame, system}

Associations:
None

Semantics:
The inherited attribute *platform* represents a concrete hardware architecture level and software processing resources. The hardware resources are tagged with MARTE::HRM stereotypes using FIBEX for bus system configuration. Moreover *platform* owns the dependability error models (i.e. fault, error, failure, error free and unclassified objects) with allocation relationships to the hardware.
The inherited attribute *workload* has no mean here.

Constraints:
TBD

3.6.5 State-based

The following stereotypes are introduced in CHESS-ML.

3.6.5.1 StatefulHardware

Extensions:
CHESS_ML::DepComponent

Attributes

- probPermFault : NFP_Real[1]

- errorLatency : NFP_Duration[1]

- repairDelay : NFP_Duration[1]

Associations:
None

Constraints:
None

3.6.5.2 StatelessHardware

Extensions:
CHESS_ML::DepComponent

Attributes

- probPermFault : NFP_Real[1]
- repairDelay : NFP_Duration[1]

Associations:
None

Constraints:
[1] May not have owned properties. (TBC)

3.6.5.3 StatefulSoftware

Extensions:
CHESS_ML::DepComponent

Attributes

- errorLatency : NFP_Duration[1]
- repairDelay : NFP_Duration[1]

Associations:
None

Constraints:
None

3.6.5.4 StatelessSoftware

Extensions:
CHESS_ML::DepComponent

Attributes:
None

Associations:
None

Constraints:

[1] May not have owned properties. (TBC)

3.6.5.5 FaultTolerant

Extensions:
UML::Component

Attributes

- redundancyScheme : RedundancyKind [1]
- SchemeAttributes : VSL::Expression (TBD) [1]

Associations:
None

Constraints:
None

3.6.5.6 RedundancyManager

Extensions:
UML::Component

Attributes

- redundancyScheme : RedundancyKind [1]

Associations:
None

Constraints:
None

3.6.5.7 Variant

Extensions:
UML::Component

Attributes

- ...

Associations:
None

Constraints:
[1] Component has to be stereotyped as StatefulHardware, StatelessHardware, StatefulSoftware, StatelessSoftware

3.6.5.8 Adjudicator

Extensions:
UML::Component

Attributes

- coverage : NFP_Percentage [1]

Associations:
None

Constraints:
[1] Component has to be stereotyped as StatefulHardware, StatelessHardware, StatefulSoftware, StatelessSoftware

3.6.5.9 MMActivity

Extensions:
UML::Activity, UML::Action

Attributes

- when : String [0..1]
- duration : NFP_Duration [1]
- probSuccess : NFP_Real [1]
- onCompletion : Activity[*]
- onSuccessfulCompletion : Activity[*]
- onFailedCompletion : Activity[*]

Associations:
None

Constraints:
None

3.6.5.10 Repair

Extensions:
CHESS_ML ::M&MActivity

Attributes
targets : Property[*]

Associations:
None

Constraints:
None

3.6.5.11 Replace

Extensions:
CHESS_ML ::M&MActivity

Attributes

- targets : Property[*]
- replacement : UML ::Activity

Associations:
None

Constraints:
None

3.6.5.12 ErrorDetection

Extensions:
CHESS_ML::MMActivity

Attributes

- target : Property[1]
- correctionProbability : NFP_Real [1]
- controlledFailure : FailureMode [1]

Associations:
None

Constraints:
[1] target Property must be an instance of DepComponnet
[2] controlledFailure must be a FailureMode specified by the target

3.6.5.13 FailureDetection

Extensions:
CHESS_ML ::M&MActivity

Attributes

- onDetection : Actuivity [*]

Associations:
None

Constraints:
None

3.6.5.14 StateBasedAnalysis

Extensions:
MARTE::GQAM:GaAnalysisContext

Attributes

- measure : String [1]
- measureEvaluationResult : String [0..1]
- type : EvaluationType[1]
- evalMethod : String[1]
- targetFailureMode : FailureMode[*]
- targetDepComponent : InstanceSpecification[*]

Associations:
None

Constraints:
[1] The platform attribute (from GaAnalysisContext) has to refer the system to be analyzed, i.e. a root InstanceSpecification owning hardware instances and the deployment information (MARTE Assign).

[2] targetFailureMode and targetDepComponent are mutually exclusive.

3.6.6 Data-flow call-graph

The following stereotypes map the data-flow and call-graph analysis.

DataFlowCallGraphAnalysis

A DataFlowCallGraphAnalysis collects relevant qualitative and quantitative information for performing data-flow and call-graph analysis. An analysis tool can use DataFlowCallGraphAnalysis to extract all the information that it needs.

Extensions:
MARTE::GQAM:GaAnalysisContext

Attributes

- result : Property [*]

  List of properties with *isComposite* relation violated.

Associations:
None

Constraints:
None

3.6.7 Failure propagation

3.6.7.1 FPTC

The FPTC stereotype allows to set FPTC expression for a given component. This is an alternative to the usage of ErrorModel state machine.

Extensions:
UML::Kernel::Comment,
CHESS::Dependability::DependableComponent::DependableComponent

Attributes

- FPTC : String[1]

Associations:
None

Constraints:
[1] If applied to Comment then annotated element has to be a ComponentImplementation

3.6.7.2 FPTCSpecification

The FPTCSpecification allows to set FPTC expression at property (i.e. instance) level.

Extensions:

UML::Kernel::Comment,

Attributes

- failure : FPTCFailureType[0..*]
- partWithPot : Property[1]

Associations:
None

Constraints:
[1] Comment applies on port of ComponentImplementation or HW Component.
[2] partwithPort has to refer a Property which type owns the annotated port.

3.6.7.3 FI4FA

The FI4FA stereotype allows to set FI4FA expression for a given component. This is an alternative to the usage of ErrorModel state machine (TBC).

Extensions:
CHESS::Dependability::DependableComponent::DependableComponent

Attributes

- fi4fa : String[1]

Associations:
None

Constraints:
None

3.6.7.4 FI4FASpecification

The FI4FASpecification allows to set FPTC expression at property (i.e. instance) level.

Extensions:
FPTCSpecification

Attributes

- None

Associations:
None

Constraints:
[1] In case the attribute failure, of this stereotype, equals:
valueCoarse or valueSubtle or omission or commission or early or late
the comment has to be stereotyped also with ACIDAvoidable.

[2] In case the attribute failure, of this stereotype, equals noFailure the comment has to be stereotyped also with ACIDMitigation.

3.6.7.5 ACIDAvoidable

Allows to characterize the FI4FA with the ACID mitigation information.

Extensions:
Comment

Attributes

- a:a_avoidable[0..1]
- c:c_avoidable[0..1]
- i:i_avoidable[0..1]
- d:d_avoidable[0..1]

Associations:
None

Constraints:
[1] The comment has to be stereotyped with FI4FASpecification
[2] FI4FASpecification.failure may be one of the following value: valueCoarse or valueSubtle or omission or commission or early or late

3.6.7.6 ACIDMitigation

Allows to characterize the FI4FA with the ACID avoidable information.

Extensions:
Comment

Attributes

- a:a_mitigation[0..1]
- c:c_ mitigation [0..1]
- i:i_ mitigation [0..1]
- d:d_ mitigation [0..1]

Associations:
None

Constraints:
[1] The comment has to be stereotyped with FI4FASpecification
[2] FI4FASpecification.failure may noFailure.

3.6.7.7 FPTCPortSlot

Allows to model a slot representing an instance of a port having FPTCSpecification or FI4FASpecification information attached. This is useful in case of multiple structured instances where the part-with-port relationships in the model are not enough.

Extension:
UML::Slot

Attributes

- FPTCSpecification : FPTCSpecification[1]

Associations:
None

Constraints:
None

3.6.7.8 ExternalFault

From ThreatsAndPropagation.

AdditionalAttributes

- kind : FPTCFailureType[1]

  The kind of incoming failure.

3.6.7.9 FailureMode

From ThreatsAndPropagation.

AdditionalAttributes

- kind : FPTCFailureType[1]

3.6.7.10 FailurePropagationAnalysis

A FailurePropagationAnalysis collects relevant qualitative and information for performing failure propagation analysis. FPTC analysis tool can use FailurePropagationAnalysis to extract all the information that it needs.

Extensions:
MARTE::GQAM:GaAnalysisContext

Attributes

- result : FPTCSpecification[0..*]

Associations:
None

Constraints:
[1] The platform attribute (from GaAnalysisContext) has to refer the system to be analyzed, i.e. typically a root component owning hardware instance, component implementation instances and allocations.

3.6.7.11 FI4FAAnalysis

A FI4FAAnalysis collects relevant qualitative and information for performing FI4FA analysis. FI4FA analysis tool can use FI4FAAnalysis to extract all the information that it needs.

Extensions:
MARTE::GQAM:GaAnalysisContext

Attributes

None

Associations:
None

Constraints:
[1] The platform attribute (from GaAnalysisContext) has to refer the system to be analyzed, i.e. typically a root component owning hardware instance, component implementation instances and allocations.

3.6.7.12 DataTypes

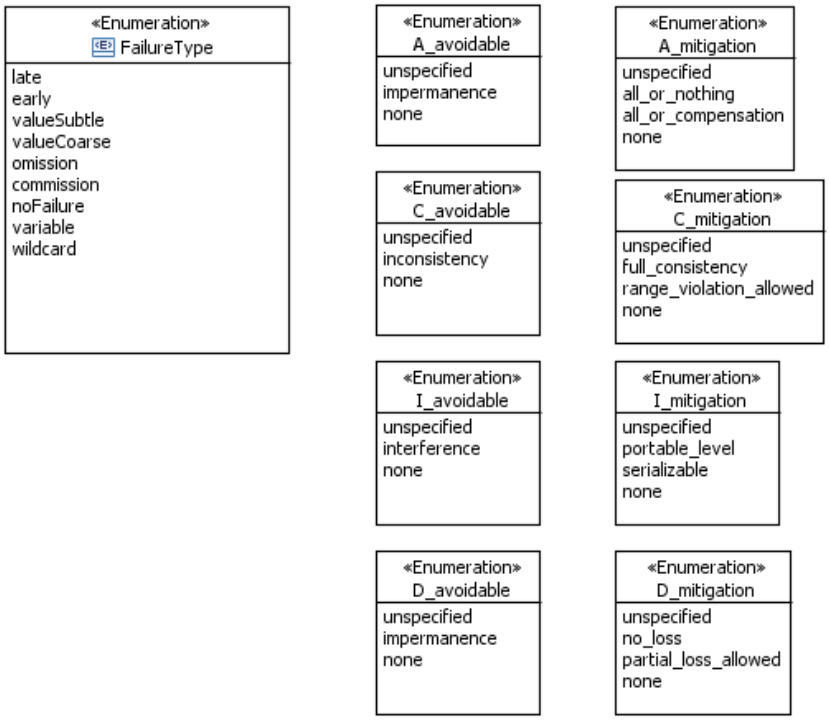

Figure 3-5: FPTC data types

3.6.7.13 Examples

Figure 3-6 shows an investigation-example of software component design (functional view) with attached extra functional information regarding FPTC (extra-functional view).

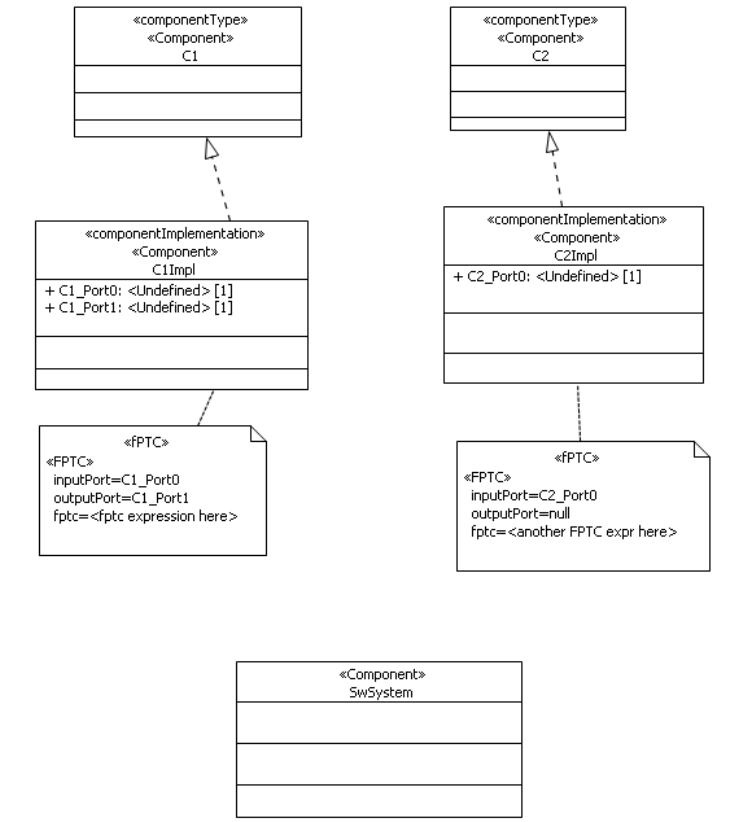

Figure 3-6: Components with annotated FPTC

Alternatively, a state machine error model can be provided for each component implementation instead of FPTC expressions, see Figure 3-7; in this case the FPTC expressions can be automatically derived starting from this error model.

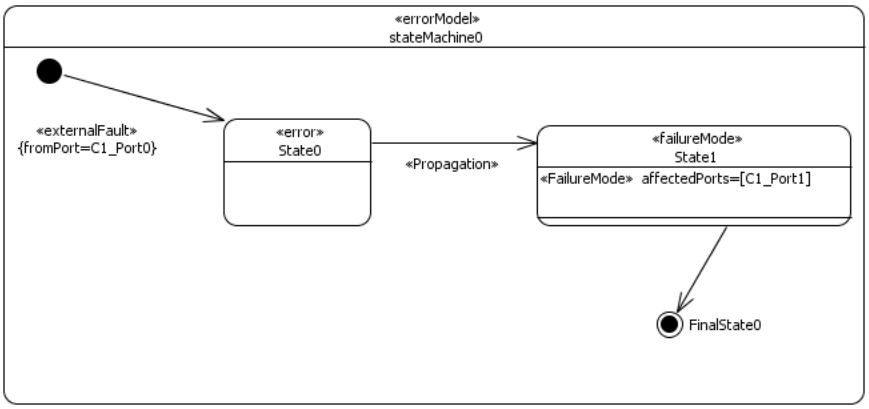

Figure 3-7: Error state machine for C1Impl

Figure 3-8 shows the details regarding how component implementation are instantiated and connected. Here Propagation connector stereotype from DependableComponent package can be used.

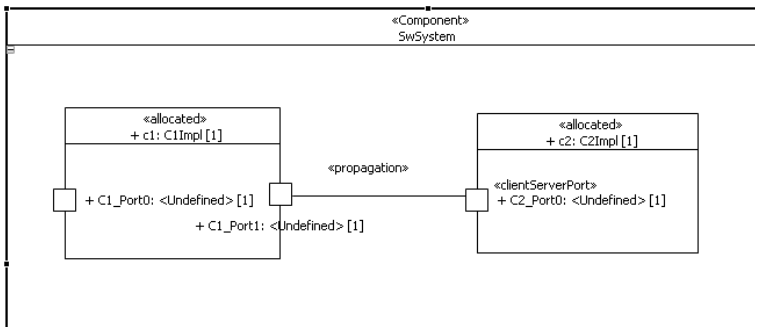

Figure 3-8: Component instances and connector (modeled as internal of SwSystem)

Figure 3-9 shows the hardware platform (deployment view); also allocations of component implementation instances to hardware component instances are modeled though the Assign MARTE construct.

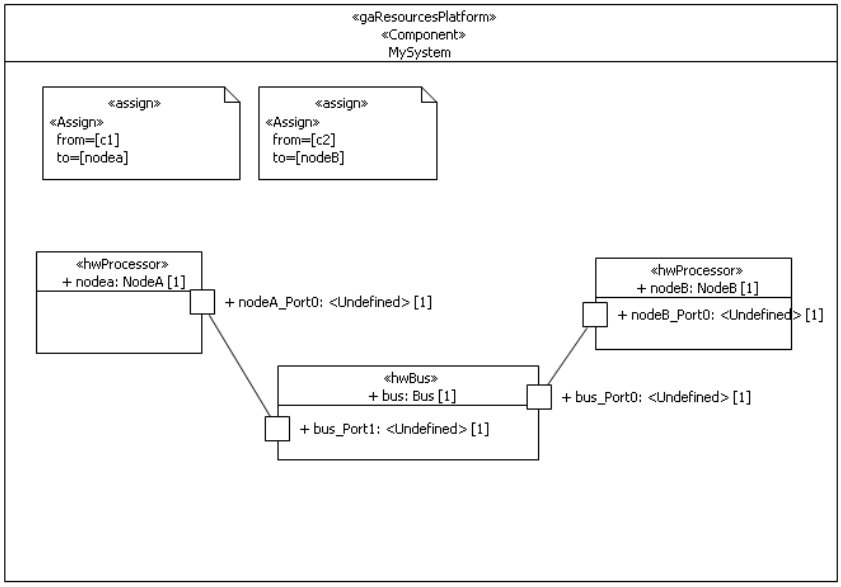

Figure 3-9: The hardware resource platform with the deployment information modeled through 'assign'

Finally Figure 3-10 shows the stereotype which can be used to feed the FPTC analysis; in particular this stereotype allows to specify the platform resources that the analysis (and so the model transformation) has to take into account: in this case the platform is the entity MySystem modeled in Figure 3-9. Starting from MySystem the analysis can derive the input hardware and component implementation instances.

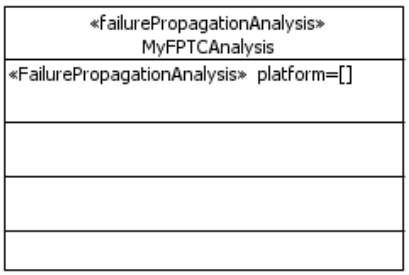

Figure 3-10: FPTCAnalysis: platform here refers System

4 CHESS HARDWARE PLATFORM SPECIFICATION

This package lists the set of entities and stereotypes which can be used to model hardware components in CHESS.

4.1 ENTITIES

4.1.1 Package

From UML.

Additional Constraints:

- *PackageMerge*. Not addressed in CHESS ML. Note: merge of packages requires definition of sets of transformations. From SysML specification: "*Combining packages that have the same named elements, resulting in merged definitions of the same names, could cause confusion in user models and adds no inherent modelling capability".*

4.1.2 FlowPort

From MARTE::GCM.

Allows to model information flow between hardware components such as its direction.

4.1.3 Connector

From UML.

Additional Constraints:

[1] When connecting hardware components it can be used to connect flow ports only.

4.1.4 DataType

From UML

4.1.5 Assign

From MARTE::Alloc. See 3.3.1.24.

4.2 STEREOTYPES

4.2.1 FIBEX

Extension:
UML ::Comment

Attributes:

- config : String[1]

XML description according to FIBEX specification

Associations:
None

Constraints:
Annotated element has to be MARTE ::HRM ::HwBus. To be used in the deployment view.

4.3 CH_HwProcessor

Extension:
MARTE::DRM::HRM::HwProcessor

Attributes:
- dataType : HwDataType[0..*]
- utilization : NFP_Real [0..1] ; derived from schedulability analysis

Commento [s3]: We should use MARTE::GQAM::GaExecHost, agreed for release CHESS v3.0

Associations:
None

Constraints:
None.

4.4 HwActuator

From MARTE::DRM::HRM.

Additional Constraints:
None

4.5 HwSensor

From MARTE::DRM::HRM.

Additional Constraints:
None

4.6 HwCache

From MARTE::DRM::HRM.

Additional Constraints:
None

4.7 HwASIC

From MARTE::DRM::HRM.

Additional Constraints:
None

5 CHESS PSM

5.1 CHESS VIEWS

5.1.1 Platform Specific Concurrent View

Not currently supported in the toolset.

Extension:
Package

Attributes
None

Associations:
None

Constraints:
None

5.2 SOFTWARE PLATFORM SPECIFICATION

See CHESS deliverable D2.3.2 v1.0.

Appendix A UML2 subset for CHESS ML

A.1 Introduction

This section addresses the selection of the UML metamodel (UML2 Superstructure v.2.2) subset; this subset plays the role of the core platform independent language for the CHESS ML.

The adoption of the entire UML meta-model in CHESS ML is avoided due to the fact that several UML constructs can violate the CHESS modelling 'philosophy' based on correct by construction and MDE principles; for instance UML constructs that cannot be managed by PIM to PSM transformation have to be denied, the reason is that *crucial* information modelled at PIM level could not be preserved during model transformations. Also UML constructs are avoided if the adoption of a dedicated profile allows a better management of the same information. Nevertheless, in the context of the CHESS project it is impossible to address and support all the modelling features available in UML2; in particular features that are not relevant for the CHESS scope are excluded.

UML semantic variation points for the imported features are fixed here at PIM level when possible, otherwise they are left to be fixed in next CHESS ML releases when support for PSM levels will be addressed (for instance through CHESS ML extensions). It is worth noting that certain semantic variation points could be left unfixed at PIM level and then specified at PSM level according to a specific target domain.

In the following, only UML2 Superstructure packages and owned features which are excluded, discussed for proposal or further constrained in the CHESS ML are listed; so

UML2 Superstructure packages/features not covered in this section are meant to be imported in CHESS ML. Concerning excluded features some related comment is given. Moreover UML entities mapping the CHESS component model are addressed.

A.2 Classes

A.2.1 Association Classes

Features from this package are excluded from the CHESS ML. Association classes, as a modelling construct add significant semantic complexity and their effect can be equivalently modelled using regular classes and associations. They are not considered fundamental for CHESS ML.

A.2.2 Power Types

Features from this package are excluded from the CHESS ML. Generalization sets add significant complexity to the semantics of generalization. Further, the effect of a generalization set can be equivalently modelled using regular classes and generalizations, albeit at the expense of some modelling convenience. Power types and generalization sets are therefore not considered fundamental for the CHESS subset.

A.2.3 Dependencies

No exclusions.

A.2.4 Interfaces

No exclusions.

A.2.5 Kernel

From Root: no exclusions.

From Multiplicities: no exclusions.

From Namespaces: no exclusions.

From Expressions:

- *OpaqueExpression*. Opaque expressions can be included in CHESS ML to support UAL (TBC).
- *Expression*. Expressions are excluded from the CHESS ML, usage of opaque expression is proposed instead.

From Constraints:

- *Constraint*. Constraint import will be needed in CHESS ML as required by MARTE or other profiles.

From Instances: no exclusions. Usage of instances in CHESS models has to be defined.

From Classifier: no exclusions. Constraints about allowed generalization hierarchies to be defined. Semantic variation point for compatibility between the redefined element and the redefining element to be fixed.

From Features:

- *BeahavioralFeature::raisedException*: the modelling of exception in CHESS ML need to be investigated.

From Operations:

- *Operation::raisedException*: the modelling of exception in CHESS ML need to be investigated.
- *Operation::preCondition.* Excluded from the CHESS ML.
- *Operation::postCondition.* Excluded from the CHESS ML.
- *Operation::bodyCondition.* Excluded from the CHESS ML.

From Classes:

- *Property::subsettedProperty*. Subsetting is excluded from the CHESS ML (not a relevant feature).
- *Property::redefinedProperty*. Property redefinition in CHESS ML needs to be investigated.

From Data Types: no exclusions.

From Packages:

- *PackageMerge*. Not currently addressed in CHESS ML. Note: merge of packages requires definition of sets of transformations. From SysML specification: "*Combining packages that have the same named elements, resulting in merged definitions of the same names, could cause confusion in user models and adds no inherent modelling capability".*

A.3 Components

A.3.1 Basic Components:

- *Component*. Component maps the functional component defined in the CHESS component model.

  Additional constraint: component (instances) can be connected through ports only.

- *Connector::contract.* Not currently addressed in CHESS ML (maybe useful to model communication patterns?).

A.4 Composite Structures

A.4.1 InternalStructures

- *Connector*. Connector maps the connector entity defined in the CHESS component model. Semantic variation point regarding what makes connectable elements compatible is fixed.

A.4.2 Ports

Additional constraint for port to solve semantic variation point: concerning delegation of ports in composite structure, a request arriving at a delegating port can have *only one* delegated port able to handle the request.

A.4.3 StructuredClasses

No exclusions.

A.4.4 Collaborations

Features from these packages are currently excluded from the CHESS ML (low priority feature).

A.4.5 InvocationActions

Currently Excluded from the CHESS ML.

A.4.6 StructuredActivities

Currently excluded from the CHESS ML.

A.5 Deployments

Features from these packages are excluded from the CHESS ML. The Deployments package of UML specifies constructs like *DeploymentTarget*, *Node*, *Device* which can be used to define roughly (for the CHESS scope) hardware architecture.

Deployment is addressed in CHESS ML by using the extensions provided by the MARTE profile and which allow to model hardware by using structural UML diagrams (Class, Component, Composite Structure).

A.6 Actions

Only CallOperationAction is imported in CHESS.

A.7 Activities

Used to model intra-component bindings.

A.8 Common Behaviours

A.8.1 SimpleTime

Features from this package cannot be directly instantiated in CHESS models; timed information in CHESS (PIM) ML is addressed through a customization of the MARTE

HLAM package. Note that this package is used as basis for Time and GQAM MARTE packages definition.

A.8.2 Basic Behavior

From *Common Behavior*:

- *Behavior::redefinedBehavior*. Currently excluded from the CHESS ML. Note: redefinition of behaviour can be at feature (e.g. replacing of operation behaviour) or classifier level (e.g. extension of state machines); the latter in particular can be hard to support if restrictions are not provided.

  The following semantic variation point has to be fixed: "How the parameters of a behavior match the parameters of a behavioural feature is a semantic variation point."

- *OpaqueBehavior*. Opaque behavior can be used for UAL integration.

- *FunctionBehavior*. Function behavior can be used for UAL integration.

From *Expression*:

- *OpaqueExpression*. Opaque expressions can be useful for UAL integration, for instance to set default value for an attribute.

From *Precondition and postcondition constraints for behavior*:

- *Behavior::precondition* and *Behavior::postcondition*. Excluded from the CHESS ML (hard to support).

A.8.3 Communications

From *Reception*:

- *Class.isActive.* In CHESS ML this information is derived according to the MARTE features application (TBD). Default value is false.

- *Signal.* Excluded from the CHESS ML; signal means asynchronous communication and the latter, when required, has to be specified through MARTE features (according to separation of concerns).

From *Extensions to behavioral features*:

- *BehavioralFeature.concurrency:CallConcurrencyKind.* Call concurrency kind for behavioral feature are excluded from the CHESS ML. This kind of information is addressed with the MARTE profile.

From *Triggers*: no exclusion.

From *Events*:

- *TimeEvent.* Excluded from CHESS ML (if not required by following import of related MARTE features); timed information in CHESS ML are addressed through MARTE HLAM package.
- *ChangeEvent*. Excluded from the CHESS ML (low level priority feature).
- *SignalEvent*. Signal event represents receipt of an asynchronous signal. It is excluded from CHESS ML.
- *AnyReceiveEvent*. Currently excluded from the CHESS ML.

A.9 Interactions

From UML specification "*an interaction can be displayed in several different types of diagrams: Sequence Diagrams, Interaction Overview Diagrams, and Communication Diagrams. Optional diagram types such as Timing Diagrams and Interaction Tables come in addition. Each type of diagram provides slightly different capabilities that make it more appropriate for certain situations*."

Not currently addressed in CHESS ML. Contribution of interaction diagrams in CHESS ML has to be investigated (for instance in order to support modelling and analysis of specific run-time scenarios, see Appendix E).

A.10 State Machines

A.10.1 BehaviorStateMachine

StateMachine in CHESS ML can only be used to model functional behaviour of class (CHESS component); usage of state chart for hardware component has to be investigated.

The following *CHESS state machine profile* is needed to allow (functional) code generation starting from state machine:

- *Pseudostate*: the following kind of pseudostate are included in CHESS ML:
  - *Initial*
  - *deepHistory*
  - *shallowHistory*
  - *junction*
  - *choice*
  - *entryPoint*
  - *exitPoint*
- *Region*: only one region can exist in the owning context.

- *State*: a State can own at least one region. Since orthogonal states are not supported, the *isOrthogonal* attribute will always be false. The *doActivity* behavioural specification is not supported and neither is the *stateInvariant* constraint.

  The body of the entry and exit behaviour must be written by using the UAL action language syntax.

  Concerning the semantic variation point in composite states where no initial pseudostate exists, the state machine will stay in the composite state without entering its region or any of the regions substates. To enter the region of the composite state a transition originating from the composite state border has to be taken by the triggering of that transition.

- *StateMachine*: orthogonal state machines are not supported and the number of regions owned directly by a state machine is therefore limited to one.

- *Transition*: all semantics related to orthogonal state execution is not supported and therefore invalid in CHESS ML.

  The effect of the transition, i.e. the (optional) behaviour to be performed when the transition fires, must be written by using the UAL action language syntax.

Note: constraints about state machine redefinitions (i.e. how it is possible to apply generalization between state machines) are not provided in the current version of this specification.

Support for DeferredEvent is under investigation.

A.10.2 ProtocolStateMachines

This package is not addressed in CHESS; it will be probably addressed in future versions probably through dedicated profile, for instance in order to formalize protocol conformance definition between interfaces and between interfaces and realizing classifiers.

A.11 Use Cases

No exclusions. Use cases are imported in CHESS ML as requested by the SA profile.

Appendix B MARTE Subset for CHESS ML

In this section MARTE extensions to the CHESS UML subset are addressed. MARTE is introduced here to enhance some basic UML structural feature and to provide PIM-level real time modelling support by using separation of concern. Moreover mapping to CHESS component model is addressed.

In general:

- Real time concerns in CHESS ML are only addressed through static structural features. So behavioural entities for real time information expression are currently excluded.

- Support for separation of concerns is provided following the HRT-UML/RCM approach, so real time information is attached to ports. Minimal extension to MARTE features are proposed to support modelling quantitative information at instance level, i.e. information attached to port instances.

MARTE packages imported in CHESS ML are declared in the following (MARTE import dependencies are not addressed).

B.1 Core Elements (CoreElements)

The concepts presented in this package serve as a general basis for the description of most elements of the rest of MARTE specification. They are imported in CHESS to support modeling of operational modes for components.

B.2 Generic Component Model (GCM)

Features from this package extend basic UML structural feature (e.g. the port entity): these extensions are provided in GCM as generally as possible without targeting a specific domain (e.g. real-time), so they can be imported in CHESS ML and possibly extended to cover modelling needs from different domains.

GCM package is applied to the CHESS UML subset with the following restrictions:

- *FlowPort*. Not currently addressed in CHESS ML (modelling of flows in CHESS ML needs further investigation.)

- *FlowProperty*. Not currently addressed in CHESS ML.

- *FlowSpecification*. Not currently addressed in CHESS ML.

- *DataEvent*. Not currently addressed in CHESS ML.

- *DataPool*. Not currently addressed in CHESS ML.

- *ClientServerPort*. *ClientServerPort::featureSpec* excluded from the CHESS ML; this is due to the fact that provided and required interfaces are modelled through *ClientServerPort::provInterface* and *ClientServerPort::reqInterface* features respectively.

- *ClientServerSpecification*. Excluded from the CHESS ML (follows the previous item).

- *GCMInvocationAction*. Not currently addressed in CHESS ML.

- *GCMTrigger*. Not currently addressed in CHESS ML.

B.3 High-Level Application Modeling (HLAM)

Elements from this packages are not fully compatible with the CHESS approach: some of them have been redefined from scratch in the CHESS profile.

B.4 Analysis Modelling (GQAM and SAM) (UPM)

SAM and GQAM extensions are used for the description of scheduling analysis models. All these annotations are independent of architecture but in CHESS we reuse all of them for component based architectures. These extensions support the description of:

1. Work load in the system. Annotations are used to describe the temporal distribution of events that specifies the load in the system.
2. Platform resources. These resources include processing resources, schedulable resources and mutual exclusion shared resourced.
3. Specification of responses for external events and consumption of resources in the responses.
4. Deadlines and resource consumption.

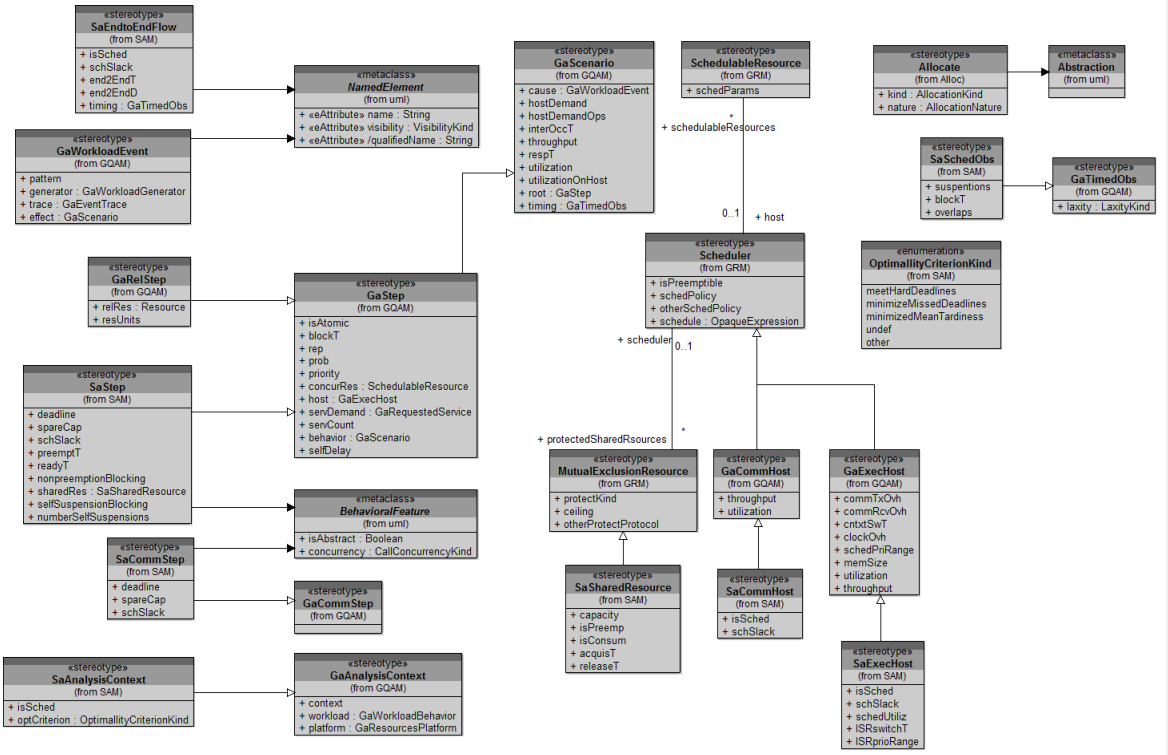

Figure B-1: Profile Extension for the Description of Scheduling Analysis Concepts

GQAM and SAM packages are applied to the CHESS UML subset with the following restrictions:

- *Scheduler*: scheduling policies supported are EDF and fixed priority.
- *SaSharedResource*: shared resources policies handled are ceiling and highest locker protocols.
- *GaWorkloadEvent*: patterns for the description of external events supported are sporadic and periodic.
- *SaStep*: the description of resource consumption is based on the worst case execution times.

MARTE is a large and complex extension and it includes several levels of abstraction. Because of this the same data can be represented with several properties and it includes properties not handled in CHESS. Next enumeration includes the set of GRM, GQAM and SAM stereotypes handled in CHESS and the minimum properties considered in transformation tools. Some additional properties could be considered to represent some analysis or end-to-end deadlines, but most of these additional properties require some restrictions and some specific analysis tools. We include non-abstract stereotypes and the stereotypes that they depend on and the properties of abstract stereotypes that we consider:

- *GQAM::GaAnalysisContext*: This extension represents the root of an analysis model. A UML model can include more than one *GaAnalysisContext* but each one would represent the root of an analysis scenario.
  - *platform[1..*]:GaResourcesPlatform*. This property identifies the set of resource platform specifications to consider in the analysis.
  - *workload[1..*]:GaWorkloadBehavior* This property identifies the set of behaviours and their work load descriptions, to consider in the analysis.
- *GQAM::GaResourcesPlatform:*This extension enumerates the set of resources to consider in the analysis.
  - *resources[*]:GRM::Resource*. This property represents the set of resources to consider in the analysis. Specific kinds of resources are considered in the scheduling analysis.
- *GQAM::GaWorkloadBehavior:* This extension includes the specification of work load events and their responses:
  - *demand[*]:GaWorkloadEvent*. This property identifies the set of events to consider in the analysis. Each *GaWorkloadEvent* includes the description of time properties and pattern of event.

- *behavior[*]:GaScenario*. This represents the specification of responses to work load events. *GaScenario* can be directly used but some specializations (*SaStep*) are more common.

- *SAM::SaExecHost->GQAM::GaExecHost->GRM::Scheduler->GRM::Resource).* This extension describes a executable host resource (e.g. CPU). Many properties in *GaExecHost* represent the same concepts as properties in *SaExecHost*, but we do not consider properties in *GaExecHost*:

  - *ISRprioRange:IntegerInterval (SaExecHost).* This property describes the range of priorities handled in the resource.

  - *ISRswitchT:NFP_Duration (SaExecHost)*. This property represents the worst case execution time needed for a context switch.

  - *isSched:NFP_Boolean (SaExecHost).* This property represents an analysis results: wheter the constratints for this host are respected.

  - *schedUtiliz:NFP_Real (SaExecHost).* This is an analysis result that represents the total utilization of the host.

  - *schedPolicy:SchedPolicyKind (GRM::Scheduler)*. This property represents the scheduling policy for the host.

  - *mainScheduler:Scheduler (GRM::ProcessingResource).* This property can reference the scheduler specification that handles the host. If this property is set, the properties inherited of *Scheduler* should not be considered.

- *SAM::SaCommHost->GqCommHost->Scheduler->Resource.* This extension represent communication resources needed to deliver remote messages.

  - *isSched:NFP_Boolean (SaExecHost).* This property represents an analysis results: wheter the constratints for this host are respected.

  - *schedPolicy:SchedPolicyKind (GRM::Scheduler)*. This property represents the scheduling policy for the host.

  - *mainScheduler:Scheduler (GRM::ProcessingResource).* This property can reference the scheduler specification that handles the host. If this property is set, the properties inherited of *Scheduler* should not be considered.

- *GRM::MutualExclusionResource->Resource.* This extensions specifies the shared resources handled in mutual exclusion:

  - *ceiling:NFP_Integer.* This is the ceiling for the resource when *protectKind* is ceiling protocol.

  - *protectKind:ProtectProtocolKind.* This property defines the scheduling protocol for the shared resource.

- *GQAM::GaWorkloadEvent*. This extension supports the specification of events that define the activation of behaviours in the model.
    - *effect:GaScenario*. This property references the specification of behaviour associated with this event.
    - *pattern:ArrivalPattern.* This data type describes the arrival pattern for the event. This data type includes several values depending on the pattern. These values are exclusive (model only includes one value). Here we only consider two patterns:
        - *periodic:PeriodicPattern.* This is the description for periodic events:
            - *period:NFP_Duration*
            - *jitter:NFP_Duration*
            - *phase:NFP_Duration*
        - *sporadic:SporadicPattern. This is the description of sporadic events:*
            - *minInterval:NFP_Duration*
            - *jitter:NFP_Duration*
- *GQAM::GaScenario:* this extension represets the system level behaviour that handles workload events
    - *cause:GaWorkloadEvent.* This is the opposite feature of property *effect* in *GaWorkloadEvent*
    - *hostDemands:NFP_Duration*. This property represents the CPU demand when all steps are in the same host
    - *repT:NFP_Duration*. This property constrains the worst case response time for the event.
    - *Root:GaStep*. This property represents the first step that represents the behaviour execution
- *SAM:SaStep->GQAM::GaStep->GQAM::GaScenario:* This extension represents one step in the sequence of step behaviours that handle the workload event. It inherits features of *GaScenario.*
    - *deadline:NFP_Duration*. This property is redundant with *repT* in *GaScenario*
    - *host:GaExecHost (GaStep)*

Allocate stereotype in *Alloc* profile is used for the allocation of shared resource in execution host. We haven't included the fields used in the description of *NFP_Duration* values. Properties considered are:

- clock:String
- precision:Real
- unit:TimeUnitKind
- worst:Real. When the *NFP_Duration* is used to represent a worst case value (e.g. worst case response and execution time) this field is used and *value* field is not considered.

value:Real (NFP_Real). When the *NFP_Duration* is used to represent a general values (e.g. period and jitter) this field is used and *worst* field is not considered.

Appendix C SysML Subset for CHESS ML

This section addresses the SysML features which are imported and excluded from the CHESS ML.

C.1 ModelElement

Features from these packages are excluded from the CHESS ML.

This package addresses views and viewpoints: these features can be used to support the CHESS methodology definition, so they can be instantiated and made available to the CHESS modeller together with the CHESS ML.

C.2 Blocks

Features from these packages are excluded from the CHESS ML. CHESS ML addresses software and hardware entities, these are well covered by UML and MARTE subsets.

C.3 Ports and Flows

Features from these packages are excluded from the CHESS ML. Equivalent features are (can be) imported from the MARTE profile.

C.4 Constraint Blocks

Features from this package are not currently addressed in CHESS ML. (Maybe useful for WP3).

C.5 Activities

Features from these packages (addressing control as data, continuous systems, probability, activities as blocks, timelines) are excluded from the CHESS ML.

C.6 Requirements

In general: CHESS ML aims to support traceability from requirements to design entities, so only basic requirement modelling capabilities are imported in the CHESS ML.

From Package Requirements.

- *Requirement.* Requirements are imported in the CHESS ML.
- *Copy*. Copy relationship is currently excluded from the CHESS ML, not a main feature for CHESS ML.
- *DeriveReqt.* Derive relationship between requirements is imported in the CHESS ML.
- *RequirementRelated:* imported in the CHESS ML (this feature is used to add properties to those elements that are related to requirements via the various dependencies)
- *TestCase.* Support for model test cases is excluded from CHESS ML.
- *Satisfy*. Satisfy relationships are imported in the CHESS ML; this is a basic feature to model traceability.
- *Verify*. Support for verification relationships (from requirement to test case) is excluded from CHESS ML.

Appendix D An investigation about UML deferred event support in CHESS

D.1 Introduction

This investigation aims to address a possible issue with the concrete realization of *separation of concerns* in the regard of the inclusion of UML state machine deferred event in CHESS Modeling Language.

D.2 UML State Machine Deferred Events

UML deferred events (or triggers) are a state machine feature which permits to model the following pattern: in a given state, events declared as deferred are not processed to trigger state transitions, but they are stored and processed only when leaving the state.

The first point to note is that deferred events are independent from any communication patterns; i.e., the event marked as deferred in a state-machine can be the manifestation of either synchronous or asynchronous communication activities alike. In the former case the client performing the synchronous communication (which delivers the event) will wait until the corresponding deferred event in the supplier state machine will have taken place. In the latter case instead the client simply deposits the request of execution that will be processed as an event by the supplier.

D.3 Use Cases

To better understand and analyse the impact of deferred triggers on the CHESS ML, the following scenarios[1] coming from the telecommunication domain addressing resource configuration and error handling should be considered:

Resource configuration:

In the case of resource configuration a component might terminate some interfaces with some containing re-configuration requests while others communicate with hardware that is being configured. In this scenario it might be far too complicated to permit the processing of a new re-configuration request before the previous one has completed. To design this state-machine the designer would therefore defer all new re-configuration requests while in the "re-configurating" state and not worry about merging new requests with ongoing ones or try to come up with another design schema to avoid getting into this situation.

Error handling:

In the case of error handling we might have a scenario where an error can occur in a component and while recovering from this error the component might be unable to process certain events before fully recovered. In this case we need to defer these events on this specific state and we cannot put this decoration on the non-functional interface since the state information, which is functional, is a crucial component in this scenario.

D.4 Deferred event: which concern?

In order to introduce deferred events in CHESS ML, the following approaches can be considered:

1. to allow deferred events in the functional component state machine and then extract/attach non functional concerns from/to it in the extra functional (EF) view;

2. to allow deferred events in the NF view only, for instance by:

    a. using inheritance between real-time and functional component to specialize functional state machines and add deferred event to states;

    b. modeling deferred events as interface decorations in the NF view.

D.5 Modeling Deferred Event through Functional View

Figure D-1 below shows a state machine for a functional component addressing an error handling case. In particular: the component can be in the *Normal* state executing *activate* and *modify* operations. At the time an error condition occurs the component must switch to the *Resuming* state where it waits for *modify* operation execution request. In the *Resuming* state, event *activate* is set as deferred, so any *activate* execution request made in state *Resuming* is postponed until the component receives a *modify* event and transitions to the *Normal* state.

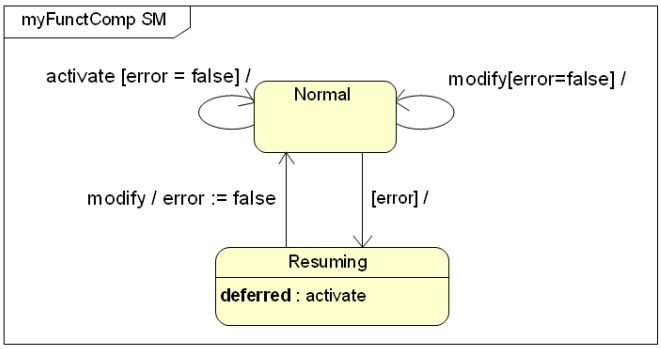

Figure D-1: ***State Machine with deferred event specification***

As previously noted, UML deferred events are independent from the synchronization mechanisms, hence in theory from figure 1 we should be able to derive different EF views. But in fact in CHESS we only allow a more constrained use: for instance if we consider Figure D-1 as the state machine associated to a protected object (so that *activate* and *modify* are protected operations), then incompatibilities with schedulability analysis would arise. In fact the synchronization algorithm supporting the invocation of protected execution requests should block callers of deferred operations when the resource is in a certain state: if used unrestrictedly, this is a synchronization behaviour exposed to temporal unpredictability that defeats static analysis and must therefore be prohibited.

Hence, considering the CHESS restrictions, is the above state machine (and thus the functional view) really a separate concern from the EF view? Starting from this functional design, what CHESS EF (real-time) views could be designed and so derived?

It therefore seems that the only allowable solution for the EF view is ***to model deferred triggers as sporadic invocations***; in fact the condition of the servicing of an invocation request being deferred implies that the event (invocation request) has to be stored in an interface queue until the deferral condition holds. This makes the sporadic entity the fittest for this purpose since it by nature holds an interface queue for inbound invocations which can be used to implement the deferred event UML feature.

D.6 Modeling Deferred Event by inheritance in the Non Functional e View

Assumption: deferred events are for sporadic communication only.

Figures 2 shows the error handler model that implements the following solution: the deferred events are introduced in the EF view, more precisely in the state machine of the RT component, by using the state machine redefinition feature.

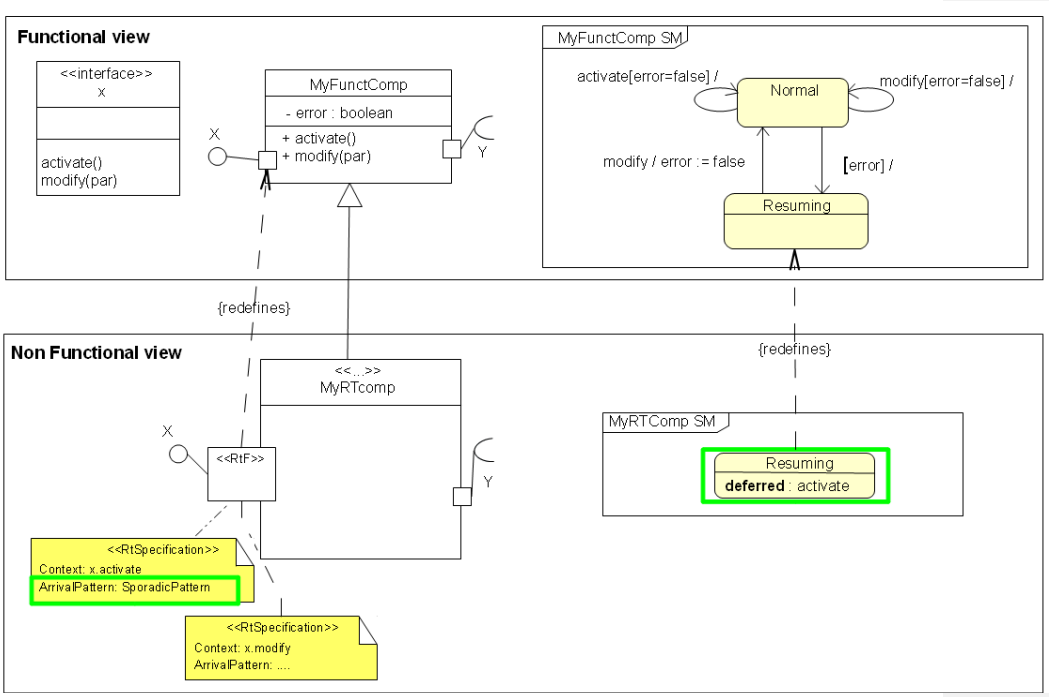

Figure D-2*: StateMachine and Component redefinition in the NF view*

It is important to note that in order to realize the above model a ***generalization*** relationship between the functional and the real time (RT) component has been added (instead of a ***composition*** relation a-la HRT-UML/RCM); moreover the model assumes that the RT component port specializes the functional component port to decorate it with the RT attributes needed for the provided services.

Note that in this case the "deferred" condition is attached to the operation *activate* provided by the RT Component *MyRTcomp*.

D.7 Modelling deferred events as interface operation decorations in the Extra Functional view: using Protocol State Machines

Assumptions:

- deferred events are for sporadic communication only,
- protocol state machine (PSM) are attached to the interface; the protocol state machine is used in the functional view to model relationships between states and triggers (operation invocations); the protocol has no information about actions performed during transition triggering. In particular the protocol state machine defines:

- an associations of allowable operations for states (which operations can be executed in a given state);
- the state(s) to be enabled after a (valid) operation invocation.

Figure D-3 shows an example of protocol state machine for interface *X* appearing in the previous examples.

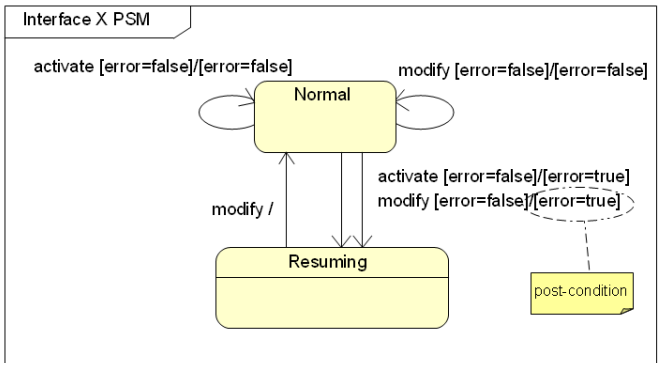

Figure D-3*: Protocol State Machine for Inteface X*

A transition in a protocol state machine can be semantically interpreted in terms of pre- and post-conditions on the associated operation. For example, in Figure D-3 the transitions concerning *activate* operation can be interpreted in the following way:

1. The operation *activate* can be called when it is in the protocol state *Normal* under the condition "*error=false"*;
2. When *activate* is called in the state *Normal* under the condition "*error=false"* then the protocol state *Resuming* must be reached under the condition "*error=true".*

Given the previous assumptions the "deferred" attribute for an operation can be set

1. either in the interface definition,
2. or as decoration for an operation defined as sporadic in the NF view.

Approach 1 implies a sporadic activation for the operation. Since this is similar to the case presented in section D.5, we don't discuss it further. Figure D-4 below shows approach 2.

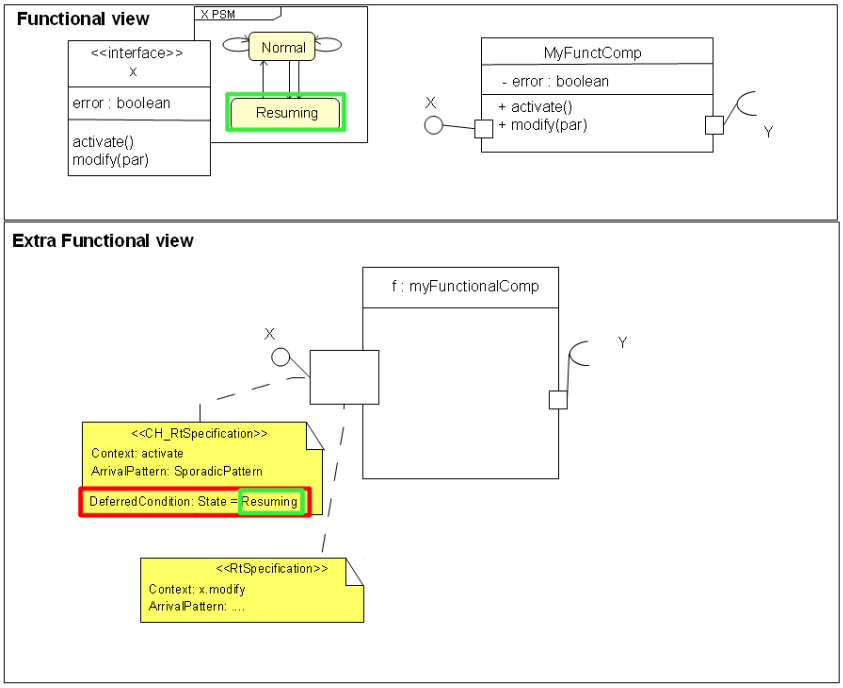

Figure D-4*: Modeling deferred event as RtSpecification attribute*

In this case the "deferred" attribute for the operation is not attached to the operation itself but it is modelled as an operation decoration through a port stereotyped as MARTE::HLM::RtSpecification (as for other non functional information). In force of this decision, *MyRTcomp* could have another instance *f2:myFunctionalComp* as internal part and provide another interface *X* with a different RT pattern for the declared operations.

In order to be adopted in the CHESS ML this solution requires:

- analysis of compatibility between the state machine of a component and the protocol state machine defined for the component interface;
- the definition of a pattern of use of protocol state machines in CHESS, to determine among others what actions need to be taken when the client violates the protocol defined in the protocol state machine.

D.8 Conclusion

We have shown that in the CHESS ML, the deferred event UML state machine feature can only be realized as the manifestation of a sporadic invocation, which justifies our recommendation to classify it among the extra functional properties.

If we allow the specification of deferred events through the functional view then synchronization properties for operations associated to those events should be automatically derived in the EF view: the methodological drawback of this solution is that it would break separation of concerns, thus for example killing the reuse potential of the same functionality in different real time scenarios.

Conversely, the approach of allowing the specification of deferred events through the EF view is fully consistent with the CHESS principle of separation of concerns. This solution does indeed require some methodological adaptation effort from the user; in fact she/he has to use different views to model information that in UML are usually collapsed into a single view/diagram. In particular:

- state machine inheritance permits to attain separation of concerns and thus greater potential for reuse. However it needs more investigation as it implies separation of concerns *by inheritance*;
- protocol state machines permit the decoration of interfaces as regards deferred conditions for sporadic activations, thereby offering a high degree of separation of concerns and reuse. The drawback of this choice is that this way of modelling deferred-in-state activation is not standard in UML.

In order to further investigate and better evaluate the approaches the study of further examples from EAB and TAS would be useful; in particular, regarding the space domain, it could be interesting to investigate mode change situation.

[1]taken from Andreas Henriksson's e-mail, CHESS mailing list, 15/02/2010

Appendix E Modeling operation WCET and transferred data

This section considers possible approaches to represent WCET information and transferred data in CHESS models.

E.1 Extending the MARTE RtSpecification stereotype

As for other quantitative information already introduced in the component instance view, operation WCET, expressed in time unit, depends on the platform which is responsible to execute the given software. This implies that different instances of the same class can have different WCET for their operations, for instance depending on the selected hardware and deployment. So the proposal is to model operation WCET information at instance level.

Following the approach already used to model quantitative information at instance level through composite structure diagram (e.g. period and deadline), a possibility is to further extend RtSpecification MARTE stereotype to introduce WCET values for operations provided by instances. Following figure shows an example of this approach for the Satellite Management Unit (SMU) Space example (see Deliverable 2.1 Appendix D); in this way WCET results are associated to an instance (part of the composite structure) and to an operation (*context* attribute of the RtSpecification stereotype).

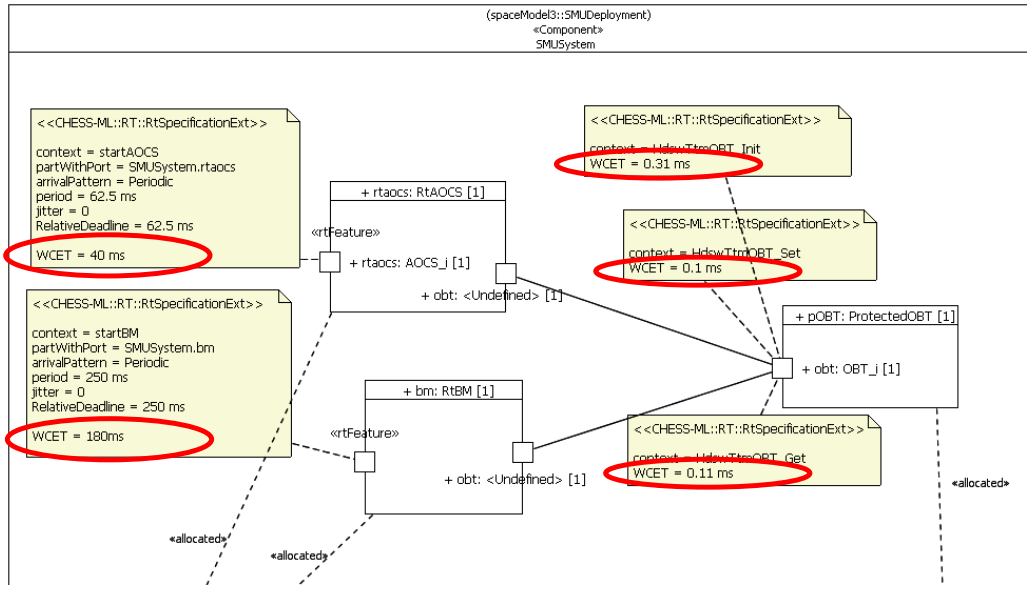

Figure E-1: modeling WCET in composiste structure diagram

E.2 Using scenarios

MARTE allows to model resources, of different kinds, and their usage. MARTE BehaviorScenario entity (from Generic Quantitative Analysis package GQAM, named GaScenario in the UML profile) allows to "*capture any system level behavior description or any operation in UML, and attaches resource usage to it*".

A BehaviorScenario is composed of sub-operations called Steps (named GaStep in the UML profile).

More from MARTE specification about Behavior Scenario and Step:

> *"Steps and BehaviorScenarios have quantitative attributes. A Step can be optional (with a probability less than one of being executed), or repeated (with a repetition count). It can be refined as another BehaviorScenario (its "behavior" association)".*
>
> *"**A Step has** a host association, a process (a SchedulableResource), **and a hostDemand for its own execution time, which can be represented either as a time, or a number of operations on the host processor**. It also may have optional requests (servDemand, with mean count servCount) for services from system components. To support demands for multiple services, these are expressed as ordered sets of service requests and counts, with the order corresponding one to the other."*

In MARTE profile, Step stereotype derives from UML::NamedElement, so it can be applied in different modeling contexts, static and behavioral.

Through sequence diagram it is possible to represent instances exchanging messages and then apply Step to messages to express execution time; in fact on MARTE specification v1.0 pag. 330 it is stated that "*in general, steps annotating UML messages represent the execution load of the associated UML ExecutionSpecification at the reception of the message*" (however this seems an informal interpretation of the MARTE semantic and should be discussed). Moreover it is possible to apply GaCommStep, a derivation of the Step stereotype, to messages to express data transferred during a communication.

The space example in the deliverable D2.1 provides WCET values for AOCS and BM main operation, moreover it provides CPU consumption and size of the transferred data of the HDSW services.

Next figure shows a sequence diagram modeling an example of services use for AOCS; quantitative information for WCET and message data size are provided through MARTE Step and CommStep stereotypes. Note about the editor: this kind of diagram is not currently supported in MDT-Papyrus so it has been designed with another graphical editor.

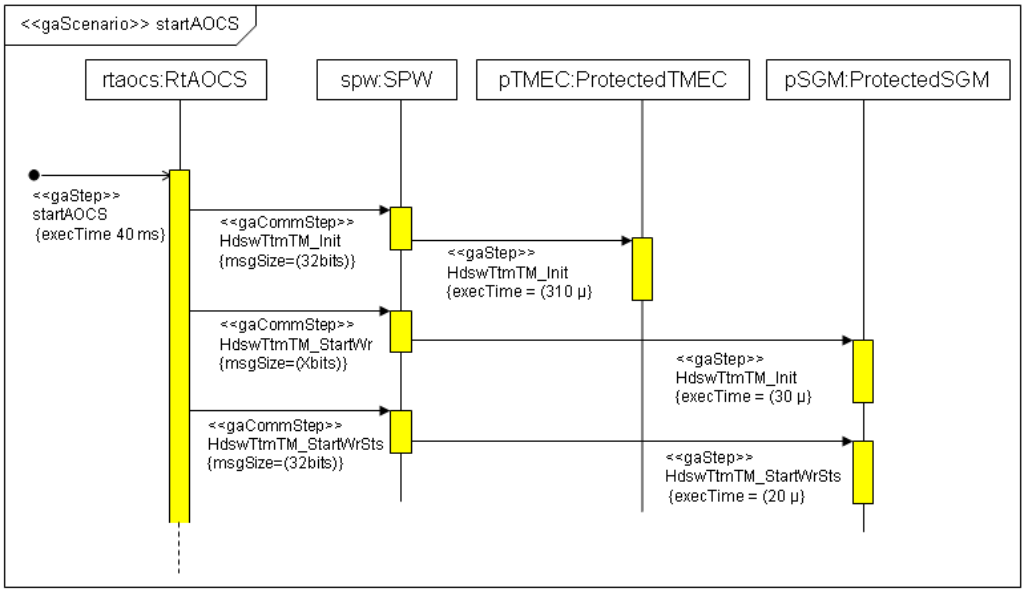

Figure 5-1: Analysis Scenario

Note about this approach: the main goal of sequence diagram is to models a particular scenario of collaboration between components and so a feasible sequence of operation calls on which quantitative information can be attached. Several scenarios can be depicted for the same system/subsystem and each scenario can allow selective analysis(!?). This can be a useful approach: however it is worth noting that in this case execution time is actually associated to a message invoking a given operation rather than on the operation itself: this allows for instance to provide different execution time

for different messages addressing the same operation on the same instance; we have to be sure that we can support this case in CHESS.