UML/MARTE Methodology for DSE

April, 2015

Microelectronics Engineering Group
TEISA Dpt. , University of Cantabria
Authors: P. Peñil

Index:

1 INTRODUCTION 5

2 DSE VARIABLES 5

2.1 DSE variables for components 6
2.1.1 DSE parameters definition in the stereotype attributes 6
2.1.2 DSE parameters definition in the ExpressionContext 7

2.2 DSE variables for instances 7

2.3 DSE Allocation variables 8

2.4 Default Values 8
2.4.1 Default Values for DSE variable in components 8
2.4.2 Default Values for DSE variable in instances 9

2.5 DSE variables for concurrency exploration 10
2.5.1 DSE variables for channel types Default Values for DSE variable in instances 10
2.5.2 DSE variables of application component 11

3 DSERULE 12

3.1 Definition of DseRule parameters 12

3.2 Definition of DseRule expression 13
3.2.1 Conditional structure 14
3.2.2 Or logic operand 15
3.2.3 And logic operand 15
3.2.4 Combined rules 15
3.2.5 Allocation DSE rule 16

4 METRICS 17

5 ANNEX I: METHODOLOGY STEREOTYPES 20

Index of Tables:

Table 1 Logic operands 14
Table 2 Algebraic operands 14
Table 3 Decision structure 15
Table 4 Logic structures 15
Table 5 Allocation operands 16
Table 6 Processor metrics 17
Table 7 Caches metrics 18
Table 8 HW Bus metrics 18
Table 9 System metrics 18

Index of Figures:

Figure 1 Example of DSE parameters with DSE variable declaration 6
Figure 2 Example of DSE parameters without DSE variable declaration 7
Figure 3 Example of DSE parameter definition by using ExpressionContext constraint 7
Figure 4 Example of DSE parameter with a DSE variable declaration for an instance 8
Figure 5 Example of DSE parameter without a DSE variable declaration for an instance 8
Figure 6 DSE parameter with MARTE Assign 8
Figure 7 Definition of default values of an explicit DSE variable 9
Figure 8 Default values of DSE variables defined in implicit style 9
Figure 9 Default values of DSE variable defined in a stereotype attribute 9
Figure 10 Definition of default values of DSE allocation parameters 10
Figure 11 DSE parameter definitions for channel types 11
Figure 12 DSE parameter definition for application components 12
Figure 13 DseRule stereotype definition 12
Figure 14 DseRule specification 13
Figure 15 DseRule specification for allocation 13
Figure 16 DseRules with logic operands 14
Figure 17 Assign Examples 16
Figure 18 DSE allocation examples 16
Figure 19 Combined DseRules 17
Figure 20 HW Metrics specification 18
Figure 21 System Metrics specification 19

1 Introduction

An important modelling feature in the UML/MARTE methodology is the possibility to define exploration parameters. It enables the use of the model to feed design exploration activities, and specifically, the application of analytical and simulation-based DSE methodology. The information captured in the model enables to connect to the VIPPE host simulator and the MOST DSE exploration tool.

2 DSE Variables

The most flexible and compact MARTE mechanism devised so far to specify the DSE parameter is to use a VSL expression called input VSL parameters in the specification of an attribute. The expression would be of the type:

dir$ParameterName = DSEValueSpecification

There "dir" is of VariableDirectionKind type. Literally, the MARTE standard states the semantics of the VariableDirectionKind type as "*Nature of the created variable: input, output, input/output. The complete semantics of this attribute depends on the context on which the variable is created*". In the modelling methodology we can state that a value "in" means a parameter of the model, which a DSE exploration tool can tune. There "dir" is the direction and its value should be "in". In this notation the direction could be omitted, so it can be self-understood that the parameter is an input parameter if the direction is not present.

The DSE values specification can be:

1. An expression of all the potential values as a VSL collection: ({v1,v2,v3}, unit)
2. An expression of all the potential values as a VSL interval: ([v_{min}…V_{max}], unit)

In the case DSE parameter does not have associated a physical unit, it can be omitted.

There is an issue with the latter style. VSL does not contemplate the specification of a quantization step. A proposal could be to add the step annotation, of the type ([v_{min}…V_{max}, step], unit). This involves a minor extension of VSL.

There is a special case in the definition of an interval DSE parameter definition. As a general rule, the *step* is defined by a number. However, in this methodology a different step specification is considered; the step is defined as exponent 2. In this case, the step is defined as *exp2*; the values of the interval follow a geometrical progression, i.e., the second value is *“v_{min} x2”*, and so on.

2.1 DSE variables for components

In order to define DSE parameters in components two expressive mechanism: DSE parameter in the stereotype attributes or in *ExpressionContext* constraints.

2.1.1 DSE parameters definition in the stereotype attributes

Each type of HW component is specifed by a specific MARTE stereotype (<<HwProcessor>>, <<HwBus>>…). Each of these stereotypes, specific component characteristics can be defined (frequency, band width…).

For this set of properties, for instance, *NFP_Frequency* for the processor frequency example, would state the range of the variable (and so its contribution to the dimension of the design space).

The DSE parameter can be annotated in two different styles:

1. **explicit DSE parameter declaration**:
   a. *$frequencyProc=({100,200,300}, MHz)*
   b. *in$frequencyProc=([100…300,100], MHz)*
   c. Examples of Figure 1.

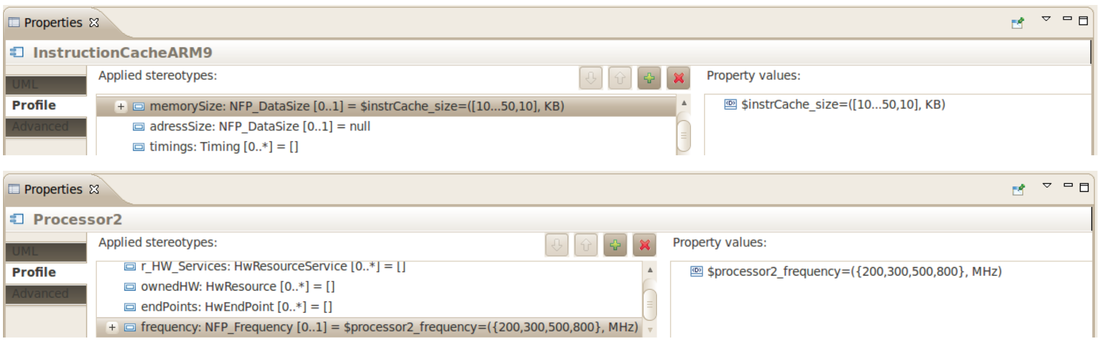

Figure 1 Example of DSE parameters with DSE variable declaration

2. **implicit DSE parameter declaration**:
   a. *({100,200,300}, MHz)* associated to a frequency attribute
   b. Examples of Figure 2. In these examples, the specification of the DSE parameters is captured by annotating the values. In these cases, the DSE variable is inferred from the model element and the attribute to be explorer: nameElement_attributeName. In the examples of Figure 2, the DSE variables are “$Bus_bandwidth” and “$RAMMemory_memorysize”. The attributes considered are: *frequency*, *memorySize*, *memoryLatency*, *wordWidth*, *bandwidth, cycle, hit, miss, staticConsumption, acces*.

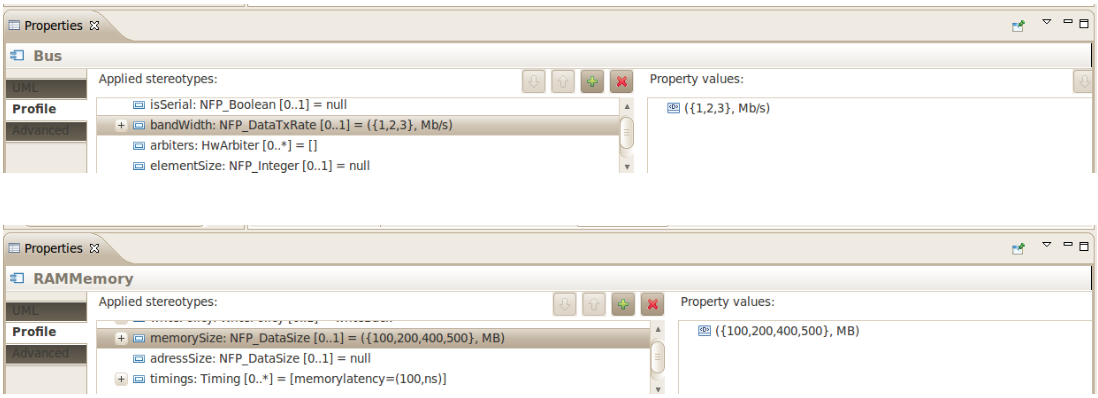

Figure 2 Example of DSE parameters without DSE variable declaration

2.1.2 DSE parameters definition in the ExpressionContext

The other modelling mechanism considered for DSE parameter definition in components is by using a UML constraint specified by the MARTE stereotype <<ExpressionContext>>. The *ExpressionContext* constraints are owned by the component which the DSE parameters are defined for.

In this style, an explicit DSE parameter is defined in the corresponding attribute. In the example of Figure 3, the attribute frequency is parameterized by the DSE variable "$frequency_processor". Then, in an *ExpressionContext* the potential values of this DSE variable are specified.

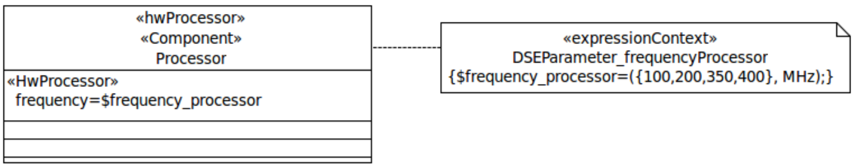

Figure 3 Example of DSE parameter definition by using ExpressionContext constraint

The implicit style for this DSE variable specification is not allowed.

2.2 DSE variables for instances

With the previous DSE variables specification styles, all component attributes can be parameterized. However, fixing a value on a component parameter fixes the same value on all the instances of the component. Therefore, an instance-level parameterization mechanism is necessary for enabling a more flexible DSE.

The mechanism proposed is to use a UML constraint and link it to the UML property which represents the component instance. The UML constraint is then stereotyped with <<ExpressionContext>>, which enables the capture of the VSL expression.

Again, two different types of DSE variable specification can be considered; the first one a DSE variable is explicitly declared as can be seen in Figure 4.

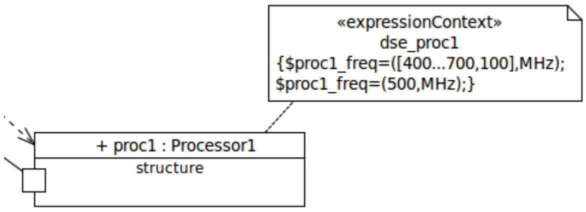

Figure 4 Example of DSE parameter with a DSE variable declaration for an instance

The second style only the name of the attribute to be explorer is annotated as can be seen in Figure 5.

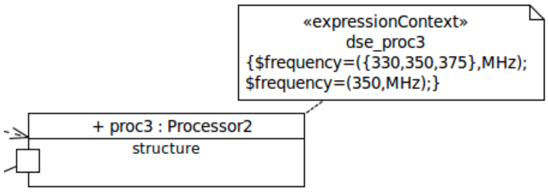

Figure 5 Example of DSE parameter without a DSE variable declaration for an instance

2.3 DSE Allocation variables

Another different DSE parameter enables to capture DSE allocations in order to explore different application-platform resources mapping. This is captured in a UML comment specified by the MARTE stereotype <<Assign>>. In the attribute *from*, the set of application or memory spaces elements to explore their mapping are attached; in the attribute *to*, the set of HW resources used as mapping targets are attached.

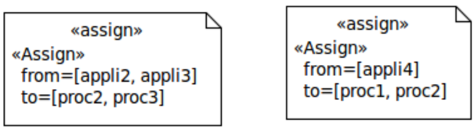

Figure 6 DSE parameter with MARTE Assign

An application or memory space can only be included in a *from* attribute once in all *Assign*s.

2.4 Default Values

In addition to the previous DSE expressions, in some cases, the designer can specify a default value of a DSE variable. This can be useful for system simulation in cases where the designer wants to simulate the system without considering the complete DSE process.

The way to define default values depends on the style used for defining the DSE variable.

2.4.1 Default Values for DSE variable in components

The *System* componente of the *ApplicationView* can have can have associated all the previous modelling variables.

As was describe in the section 2.1.1, the DSE variables of a component can be done in two ways: in the stereotype attributes and annotated in an *ExpresionContext* constraint.

In the first case, the default values are annotated in a UML constraint that must be owned by the component. There are another annotation styles:

1. **explicit DSE parameter declaration:**
    a. The default values are annotated in UML constraints according to the declaration of the DSE parameter: $nameDSEVariable = (value, unit). Figure 7 shows examples of default DSE variables specification.

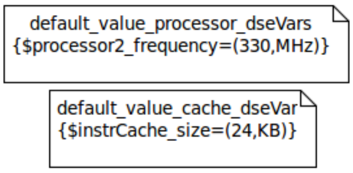

Figure 7 Definition of default values of an explicit DSE variable

2. **implicit DSE parameter declaration**:
    a. The default values are annotated in UML constraints according to the declaration of the DSE parameter: $nameAtribute = (value, unit). Figure 8 shows examples of default DSE variables specification.

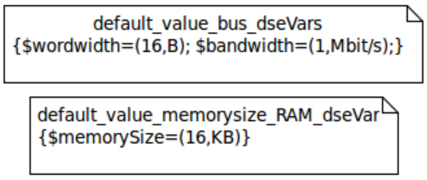

Figure 8 Default values of DSE variables defined in implicit style

A same UML constraint can be used for defining all the default values of a component.

In this case, each default value is separated by semicolon.

For the other DSE variable definition way, (using *ExpressionContext* constraint), the default value is annotated in the attribute of the stereotype where the DSE variable is defined (Figure 9).

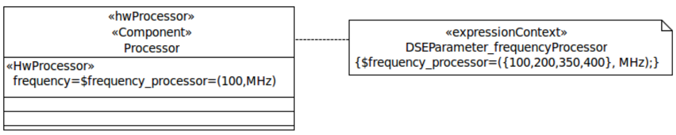

Figure 9 Default values of DSE variable defined in a stereotype attribute

2.4.2 Default Values for DSE variable in instances

In the case of DSE parameter specification for instances, in the same UML constraint where the DSE parameter is defined, the default values should be annotated

(Figure 4 and Figure 5): annotating the name of the DSE variable and its value (Figure 4) or annotating the name of the attribute and its value (Figure 5).

A special case is the default value specification of the allocation DSE parameters (defined with *Assign* comments). An UML constraint owned by the *System* component of the *ArchitecturalView* is used. The notation to use is

$allocation=(to1, from1); $allocation=(to2, from2);…

Where to_i are the names of the HW resources where the $from_i$ elements are mapped.

The UML constraint is associated to the *Assign* comment by using a UML link. There should be so many allocation definitions as elements in the attribute *to*.

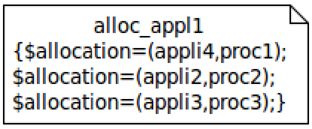

Figure 10 Definition of default values of DSE allocation parameters

2.5 DSE variables for concurrency exploration

At PIM level, the concurrency structure of the application can be explored. For that purpose, two different types of attributes are considered: attributes associated to communicating channels and attributes associated to application components.

2.5.1 DSE variables for channel types Default Values for DSE variable in instances

The channels that connect the application components are specified by channel types defined in the *CommunicationView*. These channel types are modelled as component specified by the MARTE stereotype <<CommunicationMedia>>. Then, a set of properties can be attached to the *CommunicationMedia* by using the stereotypes <<ChannelTypeSpecification>> and <<StorageResource>>.

The properties that *ChannelTypeSpecification* captures are *blockingFunctionDispatching*, *blockingFunctionReturn*, *priority*, *timeout* and *ordering*.

The channel types can have associated a storing capacity which is captured through the *resMult* attribute of the MARTE stereotype <<StorageResource>>.

The designer can explorer the potential values of these properties in order to evaluate the impact in the performance.

In order to define DSE parameters to the previous attributes in a component a different technique should be used. In this case, the DSE parameters associated to the component are captured in a UML constraint specified by the MARTE stereotype <<ExpressionContext>> instead of capturing the DSE on the attributes of the stereotypes, that is, using the modelling technique for specifying DSE parameters of instances. This is due to the different Boolean attributes (*blockingFunctionDispatching*, *blockingFunctionReturn* and *ordering*) that can be explored. In this case, the default values of the DSE parameters are defined by the values of the stereotypes attributes, instead of using a constraint.

So, an *ExpressionContext* constraint is associated to the corresponding *CommunicationMedia* component: the *ExpressionContext* constraint is owned by the *CommunicationMedia* component (Figure 11).

Then, all the previous properties considered for the channel type specifications should be annotated in the *ExpressionContext* constraint. The properties no annotated, a default value will be considered according to the values captured in the stereotypes applied on the *CommunicationMedia* (Figure 11). These default values of the DSE parameters are annotated in the different attributes of the <<ChannelTypeSpecification>> and <<StorageResource>> stereotype applied on the *CommunicationMedia* component (Figure 11).

The DSE parameters of the properties *blockingFunctionDispatching*, *blockingFunctionReturn* and *ordering* are defined as a collection "({true, false})" (Figure 11).

The rest of properties can be defined as a collection or interval DSE parameter.

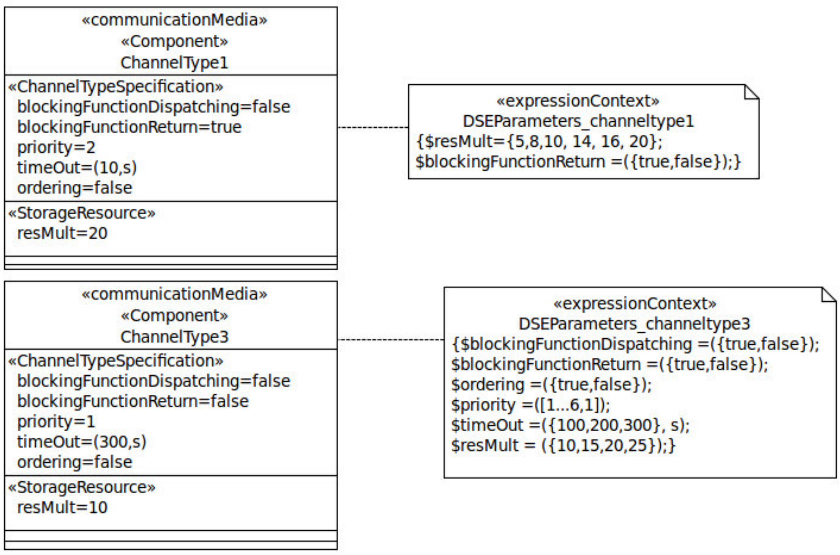

Figure 11 DSE parameter definitions for channel types

2.5.2 DSE variables of application component

Another attribute has can be considered for the concurrency structure exploration. The attribute *srPoolSize* defines the maximum number of schedulable resources to attend to the request for the services provided by the *RtUnit*.

Again, the DSE parameter is defined in a <<ExpressionContext>> constraint owned by the *RtUnit* application component. Then, the DSE variable is specified as "srPoolSize". The potential values are captured as a collection or interval. The default value of the DSE parameter is captured in the corresponding attribute of the stereotype <<RtUnit>> (Figure 12).

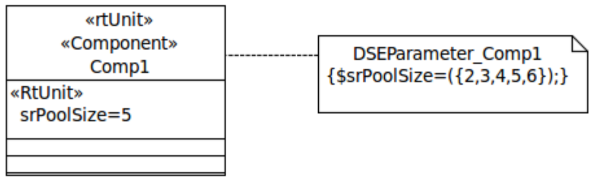

Figure 12 DSE parameter definition for application components

3 DseRule

The stereotype <<DseRule>> is used to limit the possible DSE combinations defined rules that reduce the entire DSE design options. The stereotype <<DseRule>> is applied to an UML constraint.

<<stereotype>> DseRule
parameters: String [1..*]
expression: String [1]

Figure 13 DseRule stereotype definition

3.1 Definition of DseRule parameters

In the attribute *parameters* there are defined the DSE parameters that are involved in the rule. The definition of the DSE parameter is:

dseParameterName=(nameModelElement, identifier)

According to the style selected for the specification of the DSE variables, the value to be annotated in the "identifier" is different. In the case of explicit DSE variables declaration, de definition of the rule parameters should be:

- dse1=(Processor2, processor2_frequency)
- dse2=(InstructionCacheARM9, instrCacheSize)

Where "identifier" denotes the specific name of the DSE variable to be annotated.

In the case of explicit DSE variable declaration, the "identifier" denotes the attribute name:

- dse1= (proc3,frequency)
- dse2=(DataCacheARM9, memorySize)

Figure 14 shows the declaration of two DSE rule parameters ("dse1" and "dse2"); "dse1" is related to the DSE parameter "processor2_frequency" and "dse2" with the memory size of the cache memory "DataCacheARM9".

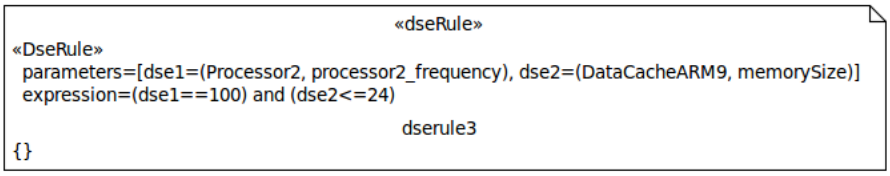

Figure 14 DseRule specification

In the case of DSE rule where DSE allocation variables are involved, the way to specify the rule is:

dseParameterName=(parameterName, allocation)

Where "toNameElement" identifies a model element included in the *to* attribute of a Assign comment. Examples of that:

- alloc1= (appli2,allocation)
- alloc2=(appli3,allocation)

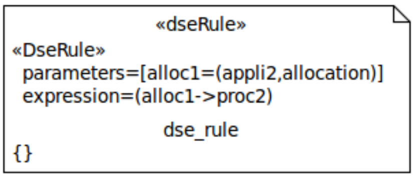

Figure 15 DseRule specification for allocation

3.2 Definition of DseRule expression

In the *expression* attribute there is annotated the specific DSE rule composed of the parameters associated by means of operands.

The style to annotate the component rules (*compoRule*) is expressing a logic operand, the DSE parameter name and a value:

compoRule=(dseName[logicOperand]Value)

Where:

- **dseName**: name of DSE parameter defined in the *parameters* attribute of *DseRule*
- **logic operand:** see Table 1
- **value**: value of the variable

The logic operands are shown in Table 1**.**

Logic operands	annotation
greater than	>

greater or equal than	>=
less than	<
less or equal than	<=
equal	==
not equal	!=

Table 1 Logic operands

The examples of Figure 16 shows rules with logic operands.

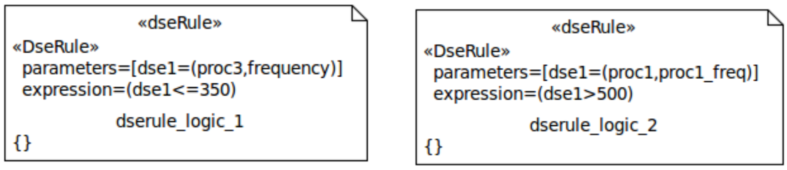

Figure 16 DseRules with logic operands

There is a constraint in the rule annotation: there are not allowed internal spaces among the different elements that composed the component rule.

1. (ds1<=300): **OK**.
2. (ds1 <= 300): **WRONG**.

Additionally, the arguments of the DSE rules can be specified by adding algebraic operations. The notation of these kind of operands should be:

((dseName[algebraicOperator]value)[logicOperand](value1[algebraicOperator]value2))

The algebraic operands are shown in Table 2.

Algebraic operands
+
*
-
/

Table 2 Algebraic operands

An example of rule with algebraic operands:

- *((dse1-25)==300)*
- *((dse1*2)==350)*

3.2.1 Conditional structure

A conditional rule has the key words shown in Table 3. The way of annotating a conditional rule is:

if*/spa/*(compoRule 1)*/spa/*then*/spa/*(compoRule2)*/spa/*else*/spa/*(compoRule3)

Example of conditional rule:

if (dse1>200) then (dse2==300) else (dse3==350)

Note that among the key words of conditional rules and the operands of the DSE rule it is required to have a space (*[spa]*):

1. if (dse1!=150) then (dse2>=200). **OK**.
2. if (dse1==200) then (dse2>=200) else (dse2<200). **OK**.
3. if(dse1!=150)then(dse2>=200). **WRONG**.
4. if(dse1==200)then(dse2>=200)else(dse2<200). **WRONG**.

It is not allowed nested conditional structures.

Decision structure	annotation
conditional structure	if…then…else

Table 3 Decision structure

3.2.2 Or logic operand

The way for annotating a DSE rule with *or* logical structure is:

(compoRule1)*[spa]*or*[spa]*(compoRule2)*[spa]*or*[spa]*(compoRule3)…

An example of *or* logic structure:

(dse1>100) or (dse4==100) or (dse2!=300)

Logic structures	annotation
and	and … and …
or	or …or…

Table 4 Logic structures

3.2.3 And logic operand

The way for annotating a DSE rule with *and* logical structure:

(compoRule1)*[spa]*and*[spa]*(compoRule2)*[spa]*and*[spa]*(compoRule3)…

Example of *and* logic operand:

(dse2<500) and (dse3>=150) and (dse4==100)

3.2.4 Combined rules

The methodology enables the rule specification where the *conditional* structure and the *or* logic structure and the *and* logic structure can be combined. This kind of rules combines conditional structures with:

1. and logic structures

   if ((dse2>200) and (dse3==250)) then (dse4==300))
2. or logic structures

   if ((dse1>=200) or (dse2!=250) or (dse4<400)) then (dse4==200))

3.2.5 Allocation DSE rule

Another kind of DSE rules are that make reference to the allocation assignment of the application entities to the platform resources. It is feasible that during the design exploration process, designer does not cover all the allocation possibilities and wants to restrict them. For that purpose, the methodology provides the allocation DSE rules.

Allocation operands	annotation
Applied to	->
Not applied to	!->

Table 5 Allocation operands

Table 5 shows the operands used for specify this allocation DSE rules. The allocation operand -> involves that an application is applied to a specific platform resource. The allocation operand **!->** involves that an application can not be applied to a specific platform resource.

The way to annotate this kind of rules is:

(applicationName[allocationoperand]resourceName)

where:

- **applicationName:** name of the application entity specified in the attribute “from” of a <<Assing>> comment.
- **allocationOperand:** Table 5**.**
- **resourceName:** name of the platform resource specified in the attribute “to” of a <<Assign>> comment.

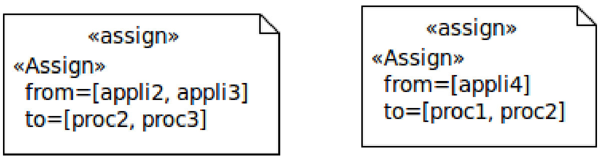

Figure 17 Assign Examples

Considering the DSE allocation parameters shown in Figure 17, examples of allocation DSE rule are shown in Figure 18.

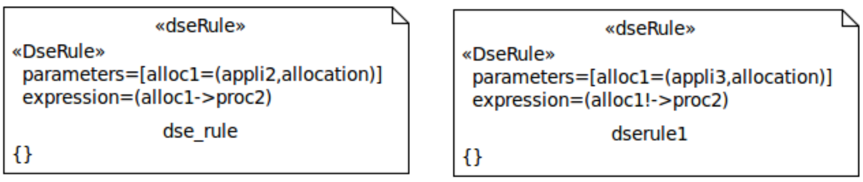

Figure 18 DSE allocation examples

The allocation DSE rules can be combined with the previous DSE rules; for instance with a conditional rule:

if (alloc1->process1) then (alloc2!->process2) else (alloc2->process3)

Another combination of the allocation DSE parameters with other different DSE parameters (Figure 19). In this case, these last ones have to be specified as was previously defined:

if ((dse1>100) and (appli2!->proc2)) then (appli2->proc3) else (appli2-> proc3)

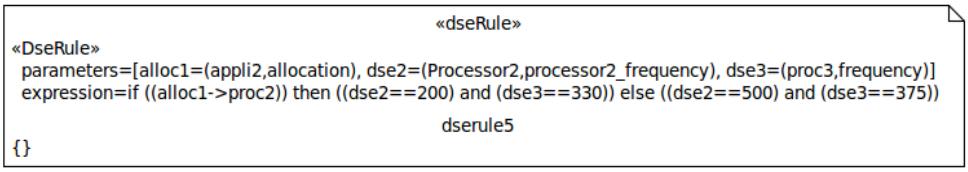

Figure 19 Combined DseRules

4 Metrics

A feature of a DSE process is to define metrics that quantify a specific property. The metrics are captured by using UML constraints specified by the MARTE stereotype <<ExpressionContext >>. Then, a VSL expression denotes the metric as:

out$nameMetric (Unit, est)

The metrics can be associated to elements of the HW platform. The metrics considered are shown in Table 6 (for processors), Table 7 (for caches) and Table 8 (for bus).

Metrics	Unit
load	%
instructionsExecuted	Integer
energy	Nfp_Energy
power	Nfp_Power
runningTime	Nfp_Time
idleTime	Nfp_Time

Table 6 Processor metrics

Metrics	Unit
misses	Integer
instructionCacheEnergy	Nfp_Energy
instructionCachePower	Nfp_Power
totalInstructionMissTransfers	Integer
dataCacheHits	Integer
dataCacheMisses	Integer

dataCacheWriteBacks	Integer
dataCacheEnergy	Nfp_Energy
dataCachePower	Nfp_Power
totalDataMissTransfers	Integer

Table 7 Caches metrics

Metrics	Unit
accessTime	Nfp_Time
transfers	Nfp_DataSize

Table 8 HW Bus metrics

Then, the *ExpressionContext* constraint is associated by using a UML link to the corresponding HW element, a processor or a bus instance. In the case a metric is related to caches, the *ExpressionContext* is associated to the processor owns the caches.

In a same *ExpressionContext* constraint several metrics of a same HW element can be annotated; each metrics expression should be separated by semicolons (Figure 20).

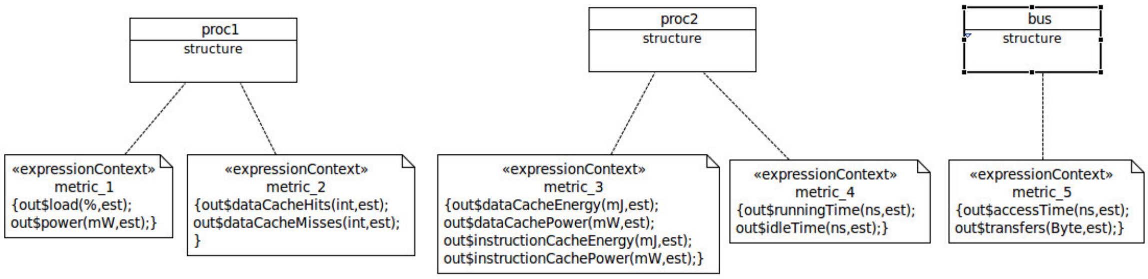

Figure 20 HW Metrics specification

The *ExpressionContext* constraint should be owned by the *System* component included in the *ArchitecturalView*.

Other metrics reference estimations for overall system, no for a specific element of it. The system metrics are shown in Table 9.

Metrics	Unit
Latency	Nfp_Time
energy	Nfp_Energy
power	Nfp_Power
instructionCount	Nfp_Integer

Table 9 System metrics

Again, the *ExpressionContext* should be owned by the *System* component included in the *ArchitecturalView* and the *ExpressionContext* constraint should be associated to this *System* component by using a UML link (Figure 21).

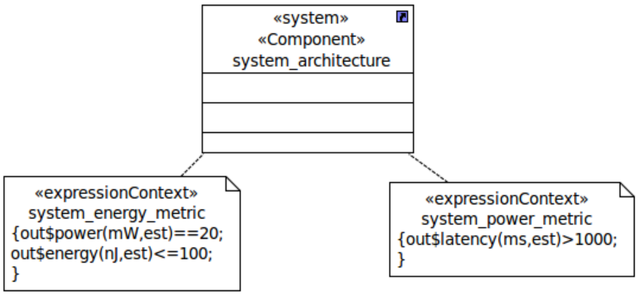

Figure 21 System Metrics specification

The values estimations can be constrained by means of logical expressions, reducing the potential values of that metric estimation to a smaller one (Figure 21). The operands that can be used are shown in Table 1.

5 Annex I: Methodology Stereotypes

Stereotype	Attributes	Profile
ExpressionContext		MARTE
DSERule	parameters: String [1..*] expression: String [1]	ESSYN
Assign	to: Element [1..*] from: Element [1..*]	MARTE