

Towards UML 2 Extensions for Compact Modeling of Regular Complex Topologies

Arnaud Cuccuru, Jean-Luc Dekeyser, Philippe Marquet, and Pierre Boulet

Laboratoire d'Informatique Fondamentale de Lille
Université des sciences et technologies de Lille
France

Abstract. The MARTE RFP (Modeling and Analysis of Real-Time and Embedded systems) was issued by the OMG in February 2005. This request for proposals solicits submissions for a UML profile that adds capabilities for modeling Real Time and Embedded Systems (RTES), and for analyzing schedulability and performance properties of UML specifications. One of the particular request of this RFP concerns the definition of common high-level modeling constructs for factorizing repetitive structures, for software, hardware and allocation modeling of RTES. We propose an answer to this particular requirement, based on the introduction of multi-dimensional multiplicities and mechanisms for the description of regular connection patterns between model elements. This proposition is domain independent. We illustrate the use of these mechanisms in an intensive computation embedded system co-design methodology. We focus on what these factorization mechanisms can bring for each of the aspects of the co-design: application, hardware architecture, and allocation.

1 Introduction

The MARTE RFP [1] (Modeling and Analysis of Real-Time and Embedded systems) has been recently voted by OMG. This request for proposals solicits submissions for a UML profile that adds capabilities for modeling real time and embedded systems, and for analyzing schedulability and performance properties of UML specifications. MARTE is not the OMG's first attempt to define a UML standard for the embedded systems community. The SPT profile (Scheduling, Performance and Time analysis) has been adopted and in use for 2 years. However, other OMG standards having significant implications for the SPT profile (such as UML2 [2,3,4,5] and QoS [6]) have been adopted during that time. Moreover, the use of the profile has led to a significant number of suggestions for improvement and consolidation, that are now part of the MARTE requirements.

Working in the field of intensive computation embedded systems, some requirements expressed in the MARTE RFP are of primary concerns for us. Application domains such as signal processing, image processing or mobile devices usually require intensive data computation to be performed, possibly in parallel, with the help of several computation units. For this kind of applications,

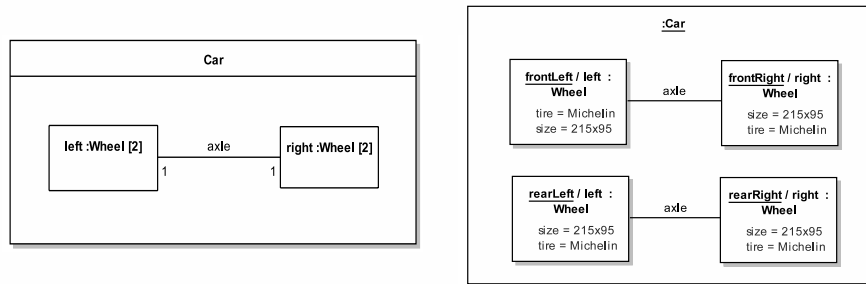


Fig. 1. Composite structure definition and instance examples

the RFP requires common high-level modeling constructs for factorizing repetitive structures for both hardware (available parallelism) and software (potential parallelism) and their allocation (temporal and spatial mapping of the software onto the hardware architecture). This paper addresses this particular requirement. Several OMG profiles (already adopted [7] or still under standardization process [8,9]) are more or less oriented toward the embedded systems domain. However, none of them proposes particular modeling constructs for factorizing repetitive structures. UML 2 proposes some mechanisms, though.

We show in this paper that these mechanisms are not suited to the needs expressed in the MARTE RFP, and that extensions are clearly required. We believe that the extensions we propose could be useful in other contexts than the modeling of embedded systems. That's why we introduce them independently of any domain consideration. Finally, we illustrate how we have experimented the use of these extensions in the context of an embedded systems co-design framework: Gaspard. Particularly, we will show that the same mechanisms are used for a compact modeling of the software, hardware and software/hardware allocation parts of the co-design methodology.

2 UML 2 Mechanisms for Compact Structural Modeling

Early versions of UML (1.x) already included such mechanisms enabling to express in a compact way the structure of a system, and the relations between entities that compose it. Relationships (associations, compositions and aggregations) can be specified between entities that exist at design-time (such as classes), and define the roles that will be played by instances of these entities at run-time (such as objects) in the context of these relations. Via a multiplicity mechanism, it also enables to specify in a compact way the potential number of occurrences concerned by these relations at run-time.

UML 2 goes further, and enables to refine the description of these relations in the context of composite structures. Composite structures refer to a composition of interconnected elements, representing potential run-time instances

collaborating over communication links to achieve some common objectives. In this kind of structures, elements are actually instantiated within the structure of a containing classifier. The number of potential occurrences of these elements and the number of communication links can also be specified with multiplicities (Fig.1).

2.1 Multiplicity UML 2 Metamodel Subset

According to UML 2 infrastructure, a multiplicity is a definition of an inclusive interval of non-negative integers beginning with a lower bound and ending with a (possibly infinite) upper bound. It specifies the range of allowable cardinalities that a set may assume. In the kernel package of UML 2 metamodel (Fig.2), the abstract concept of *MultiplicityElement* is introduced to embed this information.

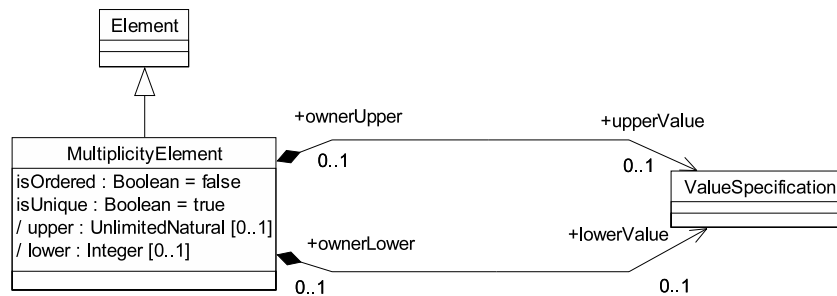


Fig. 2. Multiplicity diagram (from the UML 2 superstructure)

2.2 Limitations

However, this compact way of modeling carries very few (or no) information concerning topologies of links. In the context of composite structures, this fact can bring ambiguities in the models. Basically, two kinds of connection patterns can be considered as deterministic: The “array connector pattern” (Fig.1), or “one to one pattern”, and the star connector pattern (Fig.3), or “one to all pattern”. In other words, designers can have a clear idea of the topologies they are modeling only in the cases where the multiplicities of ends are equal to 1 and the different roles have the same multiplicity (array connector pattern) or when they match the multiplicities of the roles they are attached to (star connector pattern).

The general rule applied is the following: “Links will be created for each instance playing the connected roles according to their ordering until the minimum connector end multiplicity is reached for each end of the connectors”. In

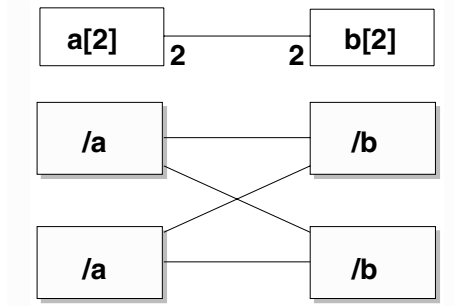


Fig. 3. Star connector pattern in a composite structure

the case illustrated in Fig.4, connectors are specified between 3 potential instances of “a” and 2 potential instances of “b”. Each “a” is connected to at least 1 “b”, and each “b” is connected to at least 2 “a”. Applying the general rule for links creation¹, we obtain two different topologies whether we start drawing links from “a” instances or “b” instances. The number appearing on each link shows the order in which links are created. Note that the two interpretations are valid.

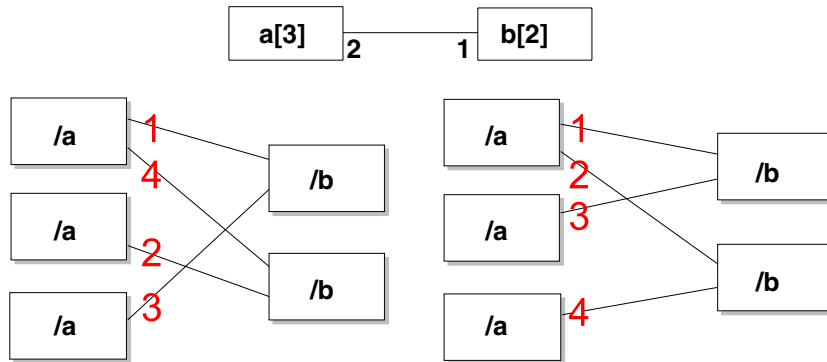


Fig. 4. Ambiguous specification of a composite structure

If we also take into account ports (another essential kind of structural feature appearing in composite structures) and their associated multiplicities, the problem is even more ambiguous. Moreover, we have shown that these mechanisms do not exhibit any topological information. The topologies modeled are a consequence of a building process rather than an identified property of the models.

¹ if we have correctly understood the ambiguous sentence that defines it...

If a designer wants to specify a particular topology, this information should be easily extracted from the model.

3 Extensions for Compact Structural Modeling

From the observations done in section 2.2, we propose powerful extensions inspired by the Array Oriented Language [10] that delete ambiguity problems and increase expression power of UML 2 structural modeling mechanisms. The extensions we propose enable to specify at design time all the links that will exist at run time in a deterministic way. The basic idea is to identify the relations between all the potential link ends concerned by each potential link. These extensions are used for the modeling of complex topologies, and concern basically multiplicities, connectors and dependencies, in the context of composite structures. An extension at the level of structured classifiers is also introduced to simplify the use and understanding of our repetition mechanisms. Even though the final architecture of the MARTE profile is not yet defined, we suppose that it will introduce a package containing common mechanisms for RTES modeling, such as the GRM (General Resource Modeling) package of the SPT profile. The extensions presented in the next sections are supposed to be defined in this package, and to be shared by the other parts of the profile².

3.1 Multi-dimensional Multiplicities

In section 2.1, we have defined the concept of multiplicity as an inclusive interval of non-negative integers beginning with a lower bound and ending with an upper bound. In Fig.2, we see that the *MultiplicityElement* metaclass carries an *IsOrdered* attribute which specifies whether the “collection” of elements is ordered or not. In the case where it is ordered, this “collection” can be seen as a mono-dimensional array, where elements are implicitly indexed.

The first extension we propose for topology modeling is to take into account multi-dimensional arrays for the description of “collections”. Multiplicities UML 2 metamodel subset is extended with the introduction of the *MultiDimensionalMultiplicityElement* concept (Fig.5). Lower and upper bound attributes are defined by *Vectors* instead of *Integers*³. In other words, the *MultiplicityElement* is seen as a particular case of the *MultiDimensionalMultiplicityElement* concept: lower and upper bounds attributes contain only one integer value. In Fig.6, we illustrate the use of multi-dimensional multiplicities to specify a cube topology ($2 \times 2 \times 2$). Each potential instance of “a” implicitly owns an index that identifies its position in the multi-dimensional array described by “a”’s multiplicity.

² such as Analysis

³ The *Vector* datatype is an ordered set of integer values. Here, each element of the *Vector* gives the size of the corresponding dimension of the multi-dimensional array.

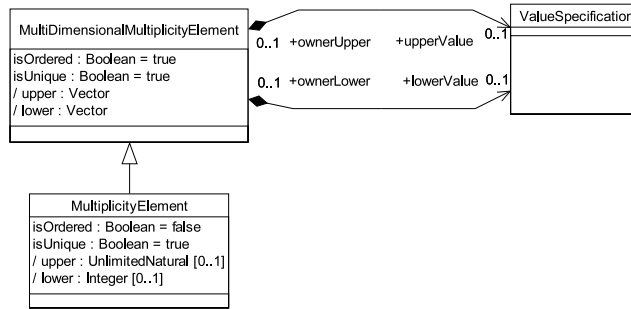


Fig. 5. Extended Multiplicity UML 2 metamodel subset for multi-dimensional aspects

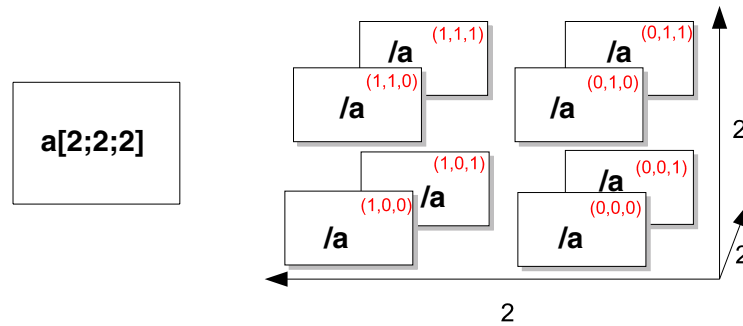


Fig. 6. Multi-dimensional multiplicities for the specification of a cube topology

3.2 Extended Relationships for Multi-dimensional Multiplicities

In order to handle modeling of links topologies, we introduce the abstract concept of *LinkTopology*. The *LinkTopology* is an optional information set that can be associated to a relationship between potential instances. It takes into account the multi-dimensional aspects introduced in the previous section. Two use cases are identified: links between several potential instances playing the same role, or playing different roles. These two use cases lead respectively to two refinements of *LinkTopology*: *InterRepetitionLinkTopology* and *RepetitiveLinkTopology*. Fig.7 is a simplified diagram⁴ summarizing the extensions introduced.

InterRepetitionLinkTopology. The systems concerned are composed of a repetition of a single element, such as in a grid or a cube topology. Each po-

⁴ Only *Connectors* and *Dependencies* are concerned by the *LinkTopology* extensions. In the context of a *Connector*, the *Elements* represent the *ConnectorEnds* associated to the *Connector*. In the context of a *Dependency*, the *Elements* represent the source and target ends of the *Dependency*.

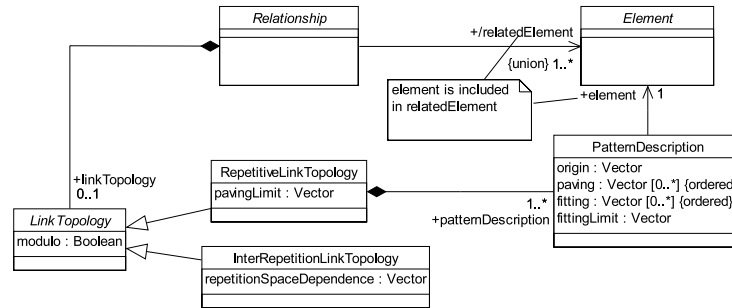


Fig. 7. Simplified diagram of the extensions for link topology description

tential instance of this element is connected to other potential instances of the same element, in a regular way. For example, in the case of a cyclic grid, each element instance is connected to neighbors located at north, south, east, and west. The first extension we propose via the *InterRepetitionLinkTopology* enables to specify the relative position of the “neighbors” of each potential instance of an element that carries a multi-dimensional multiplicity. Each potential instance is implicitly associated to one point of the multi-dimensional array described by the multi-dimensional multiplicity of the element. The *repetitionSpaceDependence* attribute is a translation vector on the space of the multi-dimensional array that identifies the position of a given neighbor. The *modulo* attribute (inherited from *LinkTopology*) indicates if the translation is applied modulo the size of the multi-dimensional array. If it is not the case, the translation is not applied on the borders of the array, and the corresponding link will not be created. In Fig.8, we illustrate the use of this mechanism for the modeling a 2d cyclic grid topology. Each connection is supposed to be bi-directional⁵.

RepetitiveLinkTopology. Complex topologies have to be modeled between different potential instances, playing different roles. Each repeated element typing each potential instance owns a multi-dimensional multiplicity. Each point of the multi-dimensional arrays identified by the multi-dimensional multiplicities corresponds to a potential link end. In the case where the repeated element owns ports and a connection is expressed on one of these ports, the ports are considered as the link ends and the multi-dimensional array is based both on the multiplicity of the repeated element and the multiplicity of the port. The mechanism we introduce via the *RepetitiveLinkTopology* enables to specify in a compact way all the correspondences that exist between the ends contained into two multi-dimensional arrays, and so all the links that will exist at run-time. Basically, the idea consists in identifying regular patterns inside of each of the

⁵ That’s why two connectors only are used to specify the relative position of the four neighbors.

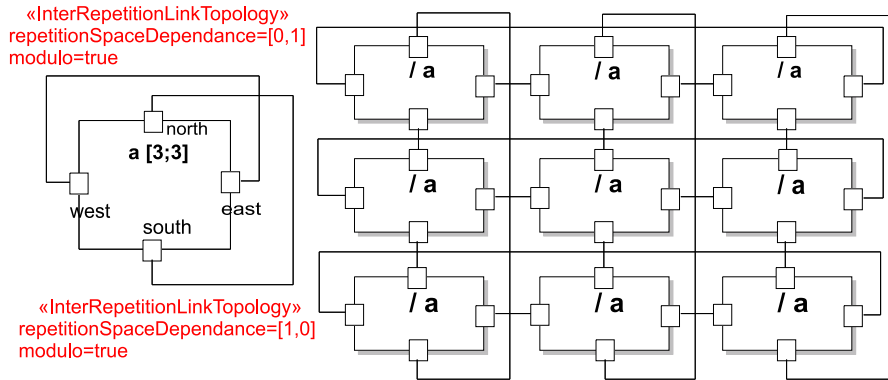


Fig. 8. A 3×3 cyclic grid topology modeled with *InterRepetitionLinkTopologies*

arrays, and to relate the points (and so the link ends) contained in these patterns. In the general case, a *PatternDescription* is associated to each of the relationships ends⁶ to identify the link ends belonging to a pattern. The *paving* attribute is a set of vectors that enable to identify the origin of each pattern inside of the array corresponding to a relationship end. The number of patterns contained inside of the array is determined by the *pavingLimit* attribute⁷. Identifying the origin of each pattern can be seen as an iterative process, where the iteration limits are given by the *pavingLimit* vector. Each pattern origin is computed by multiplying each iteration index by each paving vector, adding the related *origin* vector. From each of the identified origins, the points belonging to the patterns are identified with the *fitting* vectors. The *fittingLimit* attribute determines the number of points that belong to the patterns, or in other words, the shape and size of the patterns. Each point belonging to a pattern is computed by multiplying the *fitting* vectors by each index of the iteration space defined by the *fittingLimit* attribute, adding the origin of the current pattern.

For a given repetition index, $i, 0 \leq i < pavingLimit$, the pattern is composed of the points indexed by the following set

$$\{origin + paving.i + fitting.j \mid 0 \leq j < fittingLimit\} \tag{1}$$

if the *modulo* attribute is false and by

$$\{origin + paving.i + fitting.j \pmod{shape} \mid 0 \leq j < fittingLimit\} \tag{2}$$

if *modulo* is true. In that case, *shape* is the shape of the multidimensional array the patterns belong to. The points of the left i pattern are linked to those of the right i pattern.

⁶ A particular case is presented in section 3.3.

⁷ The number of patterns is the same for all the arrays concerned by the relationship.

In Fig.9, we illustrate the use of this mechanism with the definition of a “perfect shuffle connection pattern”⁸.

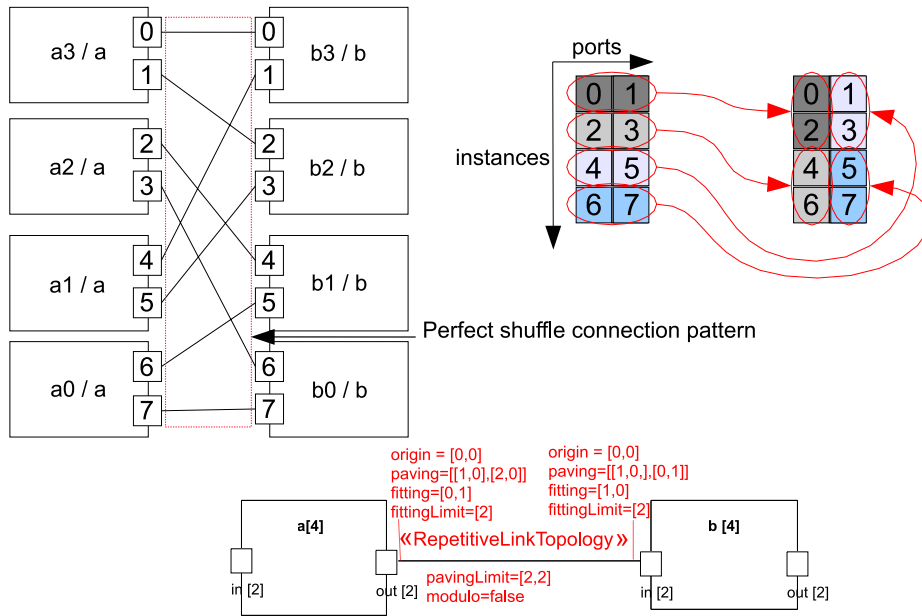


Fig. 9. Perfect shuffle modeling via a *RepetitiveLinkTopology*

3.3 Repetitive Structured Classifiers

To simplify the use of the *RepetitiveLinkTopology*, we introduce the concept of *RepetitiveStructuredClassifier*. It contains a single element with a multi-dimensional multiplicity. This element is connected to ports with multi-dimensional multiplicities on the boundary of the *RepetitiveStructuredClassifier* that contains it⁹.

In the previous section, we have introduced the concept of repetition (or iteration) space, via the *pavingLimit* and *fittingLimit* attributes of *RepetitiveLinkTopology* and *PatternDescription*. The *RepetitiveStructuredClassifier* represents a repetition space. The shape and size of the repetition space is determined by the multi-dimensional multiplicity of the element it contains. In other words,

⁸ This kind of topology is found in multistage networks such as the Omega interconnection network [11].

⁹ A *RepetitiveStructuredClassifier* is necessarily strongly encapsulated and requires the usage of ports.

each potential instance of the repeated element is implicitly associated to one point of the repetition space. Links concern ports on the boundary of the classifier and ports of each potential instance of the repeated element.

In the context of *RepetitiveStructuredClassifiers*, *RepetitiveLinkTopologies* can own only one *PatternDescription*, related to the port on the boundary of the *RepetitiveStructuredClassifier*. The *pavingLimit* is given by the multi-dimensional multiplicity of the repeated element and the *fittingLimit* is given by the multi-dimensional multiplicity of the concerned port on the repeated element. Note that *InterRepetitionLinkTopologies* can still be used in the context of *RepetitiveStructuredClassifiers*. Fig.10 illustrates the use of the *RepetitiveStructuredClassifier*.

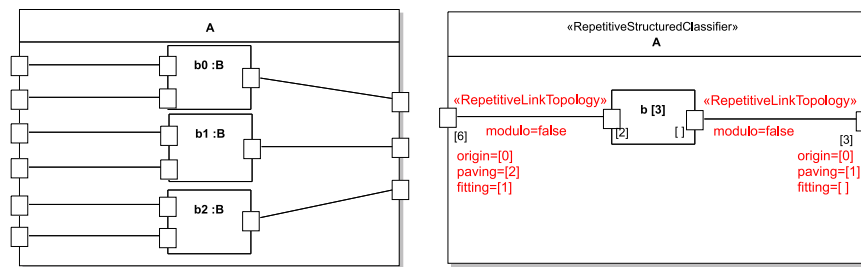


Fig. 10. *RepetitiveStructuredClassifier* example

4 Using Extensions for Embedded System Co-design

The extensions presented in the previous sections have been experimented in the context of a prototype framework: Gaspard [12,13] (Graphical Array Specification for PARallel and Distributed computing). Gaspard is an MDA (Model Driven Architecture) oriented environment for computation intensive embedded systems co-design. It follows a “Y” approach, and enables automatic model to model transformations and code generations, for various abstraction levels, from high level UML models, through the use of the MDA transformation tool ModTransf [14].

At the top level of the “Y”, software, hardware architecture and allocation abstract syntaxes are described by 3 different metamodels. However, these metamodels share common modeling constructs, such as a component oriented approach, or especially the mechanism for compact modeling we have introduced in this paper. After a brief presentation of the Gaspard metamodels and their implementation in UML profiles, this section illustrates the use of this common mechanism for the three parts of the co-design, and emphasizes on what it can bring for each of these aspects.

4.1 Gaspard Metamodels and Profiles

Software and hardware architecture metamodels share a common component oriented approach¹⁰ (Fig.11). Components are described by a composition of other component potential instances (parts), via connections between their ports. Ports enable to encapsulate the structure and the behavior of a component in order to make it independent of its environment, and increase its reusability. Interfaces are associated to ports, and a connection can be expressed between two ports only if their interfaces are compatible. The *ElementaryComponent* is a particular kind of component that does not own any structural or behavioral description. Its implementation is supposed to be available in the language that will be targeted by the Gaspard transformations.

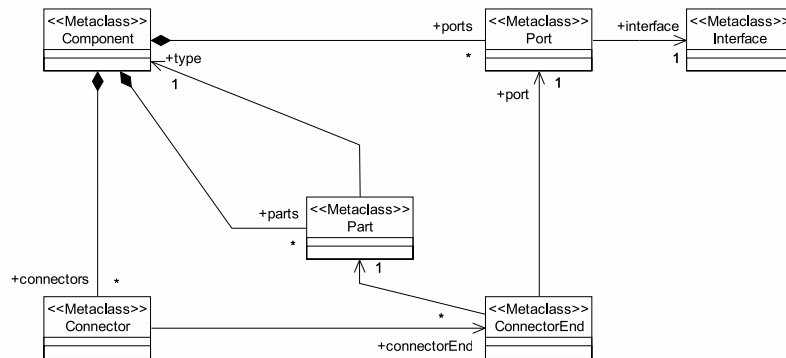


Fig. 11. Common component metamodel

In the software metamodel, the *Component* concept is refined into the *AppComponent* concept. Application components can be interpreted as functions, that performs some computations on data coming from their environment through their provided ports¹¹ and sending the results to their environment through their required ports¹². Computations are delegated to the parts of the components, or actually performed by the elementary components. Interfaces basically define the data types that can be handled by an application component.

In the hardware architecture metamodel, the *Component* concept is refined into the *HwComponent* concept. Hardware components are abstractions of physical hardware resources. Elementary components are refined in three categories, according to their function. *HwPassiveComponent*, *HwActiveComponent* and *HwInterconnectComponent* respectively represent resources able to store data

¹⁰ The concepts introduced are near to the concepts of UML 2 composite structures.

¹¹ e.g. ports with provided interfaces

¹² e.g. ports with required interfaces

(all kind of memories), perform some data transfers with or without data transformation (CPU, DMA...) and interconnect other hardware resources¹³. Interfaces associated to ports define a communication protocol between resources.

The allocation metamodel introduces concepts enabling to express the spatial mapping of a software onto a hardware architecture¹⁴. Some special dependencies can be expressed between ports of software parts and hardware architecture passive parts to model a mapping of the data (*DataAllocation*) and between software parts and hardware architecture active parts to express a mapping of computations (*TaskAllocation*).

The concrete syntax of these metamodels are implemented in UML profiles, with nearly a “one to one” equivalence between the concepts of the metamodels and the stereotypes of the profiles. The *Component* concepts are implemented via stereotyped *StructuredClasses*. Components structures are defined via internal structure diagrams. Application modeling follows a simple design pattern: One interface for each port, and one signal for each interface. The type of the signal represents the data type that can be handled by an application component. Dependencies from the allocation metamodel are implemented in stereotyped UML dependencies.

4.2 Case Study

In this section, we illustrate the use of the factorization mechanisms presented in section 3.2 for each part of the co-design modeling. The extensions have been implemented in a separate UML profile.

Application Example: Contour Detection. An image is an array of elementary values, named pixels. A contour detection of an image may be realized with a convolution. A convolution is a simple operation which produces each pixel of an output image from a linear combination of some pixels of the input image. The coefficient of the linear combination are given in a coefficient matrix.

The convolution **ContourDetection** is realized as a *RepetitiveStructuredClassifier* that can receive from its environment 514×514 signals¹⁵ representing the pixels composing the image, and 2×2 signals¹⁶ representing the values of the coefficient matrix. It can send to its environment 512×512 ¹⁷ signals representing the pixels of the computed image. Each potential instance of **t** is connected to ports on the boundary of **ContourDetection**. Basically, each **t** is able to emit one pixel via its **dataOut** port when all its input signals have been received. The order in which pixels are produced¹⁸ is determined by the order in which

¹³ Refinements of these concepts, with particular attributes, are also defined but will not be presented in this paper.

¹⁴ Temporal aspects are not presented in this paper.

¹⁵ from its 514×514 **dataIn** provided ports.

¹⁶ from its 2×2 **coeff** provided ports

¹⁷ via its 512×512 **dataOut** required ports

¹⁸ or The order in which the behavior associated to each *t* is executed

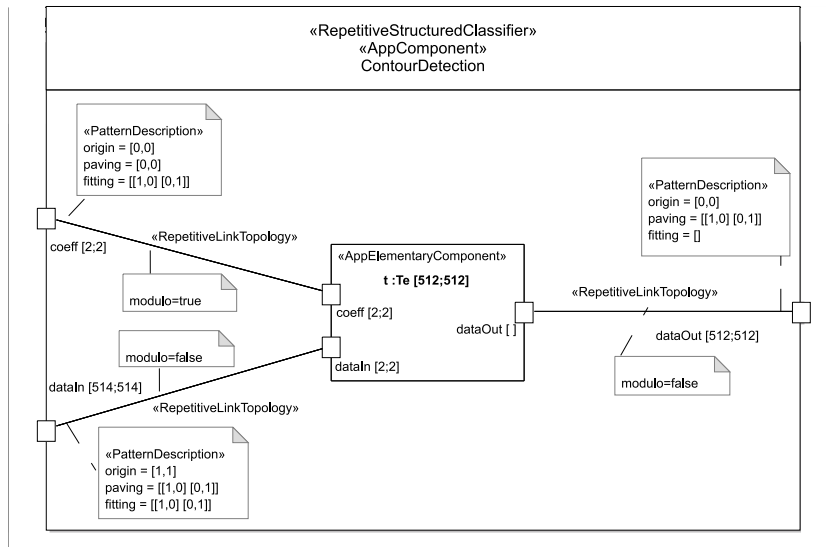


Fig. 12. Repetitive application example

signals are received from `ContourDetection` provided ports. In opposition to a classical sequential loop, the specification of `ContourDetection` does not induce any artificial execution order for the production of pixels.

Hardware Architecture Example: Bi-SPMD. We target a parallel architecture made of two sets of PE (processor elements) sharing a global memory `global:RAM` (Fig.13). Each set is made of 64 PE linked together in a ring via `east` and `west` communication channels as defined by the *InterRepetitionLinkTopology*. Each PE is associated to an element of a set of 2×64 local memories `local:ScratchPad`. The use of *RepetitiveLinkTopology* allows to specify both the link of each PE with the global memory and the link of each PE with its particular local memory.

Allocation Example: Bloc Mapping. We specify the distribution of the 512×512 potential instances of `t` on the 2×64 potential instances of `pe` so that each `pe` receives a bloc of 512×4 `t` (Fig.14). Note that the *RepetitiveLinkTopology* is here applied to a *Dependency*.

5 Conclusion

We have presented an answer to the MARTE RFP requirement concerning the definition of common high-level modeling mechanisms for factorizing repetitive structures. We have illustrated the use of these mechanisms with the embedded

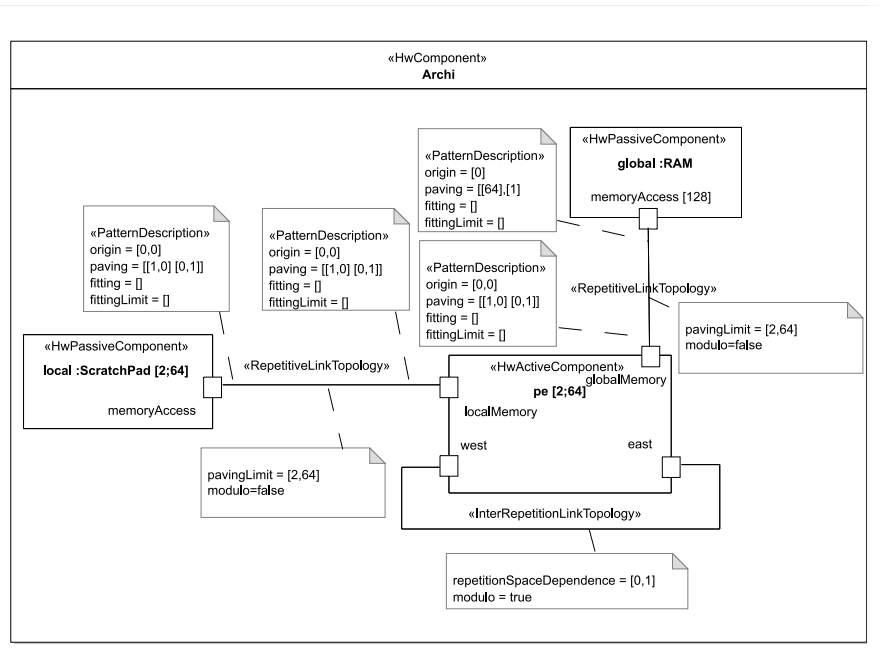


Fig. 13. Repetitive hardware architecture example

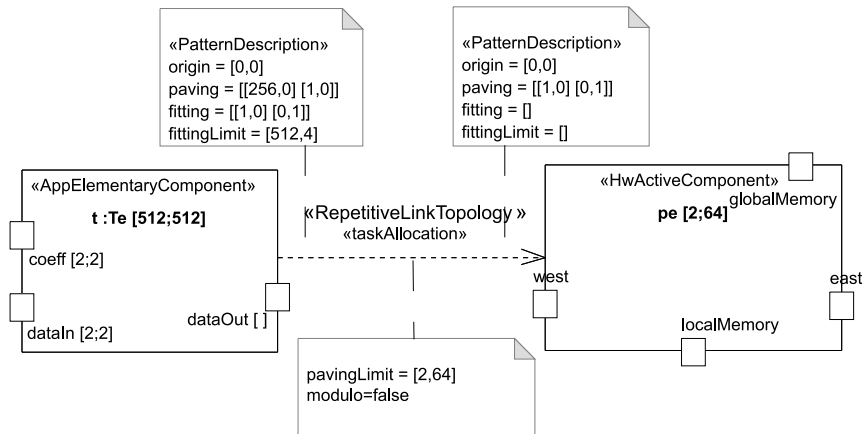


Fig. 14. Association example

systems co-design environment Gaspard, for the modeling of software, hardware and software/hardware allocation parts of a sample application. The extensions we have proposed concern only structural aspects of UML 2, and are designed to be easily integrated in a global answer to the RFP. We are now studying if the scope of these mechanisms can be extended to behavioral aspects of UML 2 (feasibility, usefulness). This work is in progress in the context of the Carroll PROTES project [15], which initiated the definition of the MARTE RFP.

References

1. Object Management Group, Inc., ed.: UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) RFP. <http://www.omg.org/cgi-bin/doc?realtime/2005-02-06> (2005)
2. Object Management Group, Inc., ed.: UML 2 Infrastructure (Final Adopted Specification). <http://www.omg.org/cgi-bin/doc?ptc/2003-09-15> (2003)
3. Object Management Group, Inc., ed.: UML 2 Superstructure (Available Specification). <http://www.omg.org/cgi-bin/doc?ptc/2004-10-02> (2004)
4. Object Management Group, Inc., ed.: UML 2 Diagram Interchange (final adopted specification). <http://www.omg.org/cgi-bin/doc?ptc/2003-09-01> (2003)
5. Object Management Group, Inc., ed.: UML 2 OCL (Final Adopted specification). <http://www.omg.org/cgi-bin/doc?ptc/2003-10-14> (2003)
6. Object Management Group, Inc., ed.: UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms. <http://www.omg.org/cgi-bin/doc?ptc/2004-09-01> (2004)
7. Object Management Group, Inc., ed.: (UML) Profile for Schedulability, Performance, and Time Version 1.1. <http://www.omg.org/technology/documents/formal/schedulability.htm> (2005)
8. Object Management Group, Inc., ed.: UML Extension Profile for SoC RFC. <http://www.omg.org/cgi-bin/doc?realtime/2005-03-01> (2005)
9. Object Management Group, Inc., ed.: SysML v0.9. <http://www.omg.org/cgi-bin/doc?ad/05-01-03> (2005)
10. Demeure, A., Lafage, A., Boutillon, E., Rozzonelli, D., Dufourd, J.C., Marro, J.L.: Array-OL : Proposition d'un formalisme tableau pour le traitement de signal multidimensionnel. In: Grets, Juan-Les-Pins, France (1995)
11. D.A., L.: Access and alignment of data in an array processor. *IEEE Trans. Comput.* **C-24** (1975) 1145–1155
12. Dekeyser, J.L.: Model driven co-design for system on chip. In: MDE for Embedded System Summer School, Brest, France (2004)
13. Laboratoire d'informatique fondamentale de Lille: Gaspard home page. <http://www.lifl.fr/west/gaspard/> (2005)
14. Dumoulin, C.: ModTransf: A model to model transformation engine (2004) <http://www.lifl.fr/west/modtransf>.
15. The Carroll Research Programme: (2005) <http://www.carroll-research.org/>.