

# Model Driven Engineering for Regular MPSoC Co-design

Pierre Boulet

Arnaud Cuccuru

Jean-Luc Dekeyser

Ashish Meena

Laboratoire d'Informatique Fondamentale de Lille

Cite scientifique, Villanelle d'Asce, 59 655, Cedex, France

Email: [Firstname.Lastname]@lifl.fr

## Abstract

*The evolution of technologies is enabling the integration of complex platforms in a single chip, called a System-on-Chip (SoC). Modern SoCs may include several CPU subsystems to execute software and sophisticated interconnect in addition to specific hardware subsystems. To manage and exploit this high degrees of provided parallelism in hw \$ sw, we need regular constructors both for hardware and software.*

*SoC co-design requires to master a lot of different abstraction levels, different simulation techniques, different synthesis tools. Due to the technology evolution, the best one is the one to come. Evolution of the embedded systems is not simple, both hardware and software, the business logic has to be kept and the technical aspect has to be thrown. To improve the permanence of System on Chip we have to abstract from the technical concerns. Model Driven Engineering proposes a separation of concerns: application and technical concerns. Use of modeling standard can capitalize system descriptions and improve system evolution and integration. We propose the use of UML2 as a modeling language for MPSoC system design.*

*To model regular hardware and software, we propose to introduce 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

### 1.1. Trend Towards Regularity

As compared to previous small embedded controllers designing, the designing complexity of today's embedded systems (SoC, MpSoC, etc) has grown exponentially. This complexity rise has been seen due to complex integration of multiple resources (DSP, GPP, FPGA, etc) on a single die [15, 14, 16, 4, 28] and further to the miniaturization of electronic and mechanical devices, the changes in the approaches to design as determined by the cognitive complexity of the task, the massive use of test automation tools, and the pervasive use of embedded systems in all kinds of applications. Real-time systems, beside high performance computing, are about strict timing constraints (deadline, periodicity), application-specific design, and high reliability (fault tolerance). The current trend of using multiple processors and controllers (SoC, MpSoC etc) has been promoted to satisfying the needs of real-time systems. This trend has made real-time systems a special case of chip level high performance distributed computing. Using multiple processors is an attractive solution because of its price-performance ratio, locality constrains (data processing must take place close to sensors or actuators), reliability (replicating of computing resources provides fault-tolerance), high throughput (parallel task execution) and high schedulability (many scheduling possibilities).

MpSoCs typically consist of heterogeneous processors among which some nodes can be homogeneous (SIMD) arrays of processors. Communication networks (NoC) [4, 16] consist of switches and routers. But in this communication platform nodes of SoCs are physically close to each other, have high link reliability and low latency.

The trend in MpSoC platforms concerns using regular homogeneous hardware units (SIMD, VLIW, etc) as a subsystem. The newcoming MpSoCs are multi-dimensional grids of elementary processors [15, 14, 16, 4, 28]. Some other regular architectures have already appeared and were

proposed on SoCs [5, 21, 25, 31, 27, 7]. Among them, there exist some reconfigurable VLIW and SIMD architectures templates (like ADRES [19], RaPiD [11], REMARC [20], MorphoSys [24]). These proposed regular cores and templates can be used as a subsystem in NoC based SoCs or in bus based SoCs. The known fact is that these regular subsystem are highly supportive to facilitate higher degrees of data parallelism [23, 18, 2, 30, 26, 3].

We focus on the applications that process large amounts of data arrays and tend to be data-intensive. Of course, they are generally found as a sub part(s) of some global application task graph. Application such as battlefield intelligence, surveillance, entertainment, tele-medicine, disaster management and wireless communication (OFDM) [12] etc) have many regular data intensive parts, which typically includes compression, decompression, FFT, IFFT, DCT, etc. In such data intensive applications, the profiling results have shown that, these regular parts (e.g loops) consume approximate 80% of the total execution time of the global application.

This kind of data parallelism can be exploited at fine levels of granularity, i.e instruction level by using VLIW (Very Long Instruction Word) [3, 26] architectures and at data level by SIMD (Single Instruction Multiple Data) architectures [23, 18, 2, 30].

Architecture proposals like Raw [1] have shown the flexibility to explore many forms of parallelism. It can support both the SIMD and MIMD programming models used by conventional multiprocessors. This SIMD concept can be a special instruction set provided by implementing through separate special processor or by just incorporate these special (multimedia) instruction set along with other instruction set.

As seen currently and in the future, MpSoCs can be considered as NoC enabled multi-processor architectures. We believe the use of regular homogeneous parallel computation units will be more in practice with high order of density. Especially for Multiprocessor Systems on Chips, which are called NoC (Network on Chip) based SoCs [4]. The on-chip communication backbone connects a large number of heterogeneous or homogeneous processing clusters and storage elements. The NoC based designs do not need to be synchronized with other subsystem cores like in bus based MpSoC designs. An other advantage of NoC is scalable data diffusion to the cores and provides more possibilities towards power aware optimizations.

We believe that regular parallel computation units will be more and more present in embedded in systems in the future, especially for Systems on Chips.

This belief is driven by two considerations:

**Time-to-market constraints.** They are becoming so tight that the massive reuse [?] of a few existing computation units is one of the only ways for fast development,

to meet the computation power needed for next generation embedded applications, and to remain in competition.

**Homogeneous hardware.** Parallelism is one among the good potential solutions to reduce power consumption in MpSoCs [6, 13].

With a large number of processing units, the only way to make the most of an MpSoC is to take account of the homogeneous hardware topology. The repetitive constructs we propose can be used to model parallel computation units, such as grids, but also complex static or dynamic interconnection networks, or memory banks. Keeping a compact representation of regular homogeneous structures is the only way to avoid a complexity explosion. No one would ever think about programming computation intensive applications without loops. The problem is the same when dealing with repetitive hardware structures. One wants to describe the repeated element only once and have a compact way of expressing the repetition. When dealing with mapping of repetitive application parts onto repetitive hardware parts, it is necessary to use regular mappings expressed in a compact way to avoid an explosion both in optimization time and in code size (which costs a lot in resource usage and power consumption).

## 1.2. Regularity and CAD Tools

These rising complexities and required flexibilities have raised many questions against today's existing design tools, techniques, their methodologies, the quality of their solutions, and their time and space complexity.

Commonly these tools are sub parts of a CAD tool and used at different levels of abstraction. Many CAD tools follow the concept of Co-design. Most of the CAD tools start with the highest level of abstraction i.e modeling of hardware platform and application, and follow some transformations between different levels (TLM, RTL etc). At last a final realization of hardware with some HDL (hardware description language) and application into C, JAVA etc is produced.

Our project Gaspard2 [17] is a kind of CAD tool based on Model Driven Engineering. The main objective of our project is to provide a new set of intermediate models and model transformations for Co-design. Co-design can be defined as a simultaneous designing environment for hardware, software and their relative mapping and scheduling.

These models and methodology are focused to meet current and future embedded system designing, considering changing trends and trade-offs. Gaspard2 follows OMG (Object Management Group) [29] specifications. The Model Driven approach we opted in Gaspard2 helps us to remain open with other platforms, tools

and standards. It is based on the modeling specifications such as the MOF, the UML and XMI [29]. This paper gives an overview on the UML 2.0 Gaspard2 profile.

To support Multiprocessor SoC co-design, we propose powerful extensions inspired by the Array Oriented Language [9], which allow multidimensional repetitive expressions, to increase the 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. 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.

The rest of this paper is structured as follows: we introduce our modeling extensions in section 2, then, in section 3, we illustrate how we have experimented the use of these extensions in the context of an embedded systems co-design framework: Gaspard2. We finally conclude this paper in section 4.

## 2. Regular repetitive modeling in UML 2

### 2.1. Multi-Dimensional Multiplicities

UML 2 has defined the concept of multiplicity as an inclusive interval of non-negative integers beginning with a lower bound and ending with an upper bound. This 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. Lower and upper bound attributes are defined by *Vectors* instead of *Integers*<sup>1</sup>. 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.1, we illustrate the use of multi-dimensional multiplicities to specify a grid topology ( $3 \times 3$ ).

### 2.2. Extended relationships for multi-dimensional multiplicities

In order to handle modeling of links topologies, we introduce the abstract concept of *LinkTopology*. The *LinkTopol-*

<sup>1</sup> 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

*ogy* 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*. 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*. summarizing the extensions introduced.

**2.2.1. InterRepetitionLinkTopology.** The systems concerned are composed of a repetition of a single element, such as in a grid or a cube topology. Each potential 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 *repetitionSpaceDependance* 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.1, we illustrate the use of this mechanism for the modeling a 2d cyclic grid topology. Each connection is supposed to be bi-directional<sup>2</sup>.

**2.2.2. 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

<sup>2</sup> That’s why two connectors only are used to specify the relative position of the four neighbors.

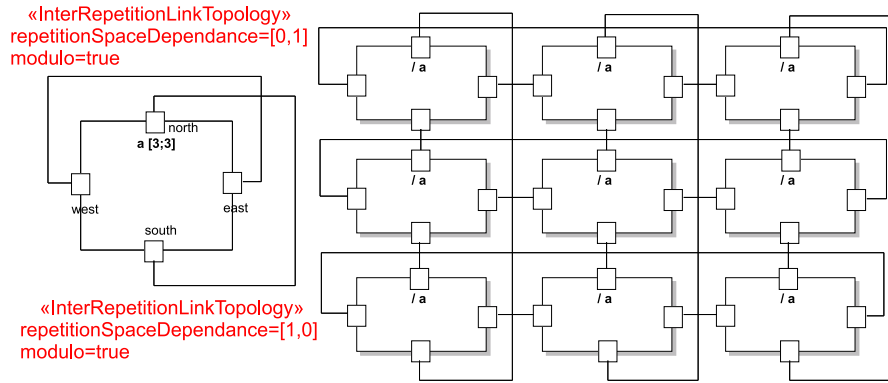


Figure 1. A 3×3 cyclic grid topology modeled with *InterRepetitionLinkTopologies*

links that will exist at run-time. Basically, the idea consists in identifying regular patterns inside of each of the 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 ends<sup>3</sup> 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* attribute<sup>4</sup>. 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.

In Fig.2, we illustrate the use of this mechanism with the definition of a “perfect shuffle connection pattern”<sup>5</sup>.

### 2.3. 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 it<sup>6</sup>.

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, 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.

## 3. Using Extensions for Embedded System Co-Design

The extensions presented in the previous sections have been experimented in the context of Gaspard2. Gaspard2 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

<sup>3</sup> A particular case is presented in section 2.3

<sup>4</sup> The number of patterns is the same for all the arrays concerned by the relationship.

<sup>5</sup> This kind of topology is found in multistage networks such as the Omega interconnection network [8]

<sup>6</sup> A *RepetitiveStructuredClassifier* is necessarily strongly encapsulated and requires the usage of ports

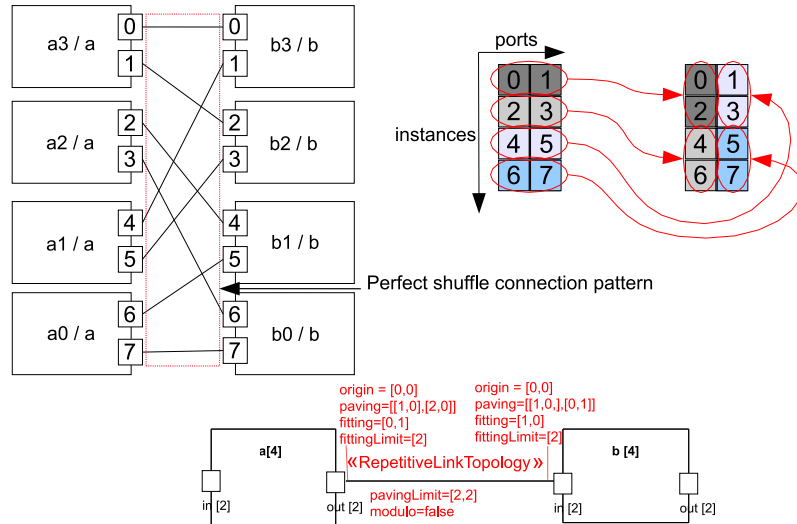


Figure 2. Perfect shuffle modeling via a *RepetitiveLinkTopology*

model transformations and code generations, for various abstraction levels, from high level UML models, through the use of the MDA transformation tool ModTransf [10].

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 Gaspard2 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.

### 3.1. Gaspard2 metamodels and profiles

Software and hardware architecture metamodels share a common component oriented approach<sup>7</sup>. 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 Gaspard2 transformations.

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 trough their provided ports<sup>8</sup> and sending the results to their environment trough their required ports<sup>9</sup>. 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 (all kind of memories), perform some data transfers with or without data transformation (CPU, DMA, ...) and interconnect other hardware resources<sup>10</sup>. 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 an hardware architecture<sup>11</sup>. 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

<sup>7</sup> The concepts introduced are near to the concepts of UML 2 composite structures.

<sup>8</sup> e.g ports with provided interfaces

<sup>9</sup> e.g. ports with required interfaces

<sup>10</sup> Refinements of these concepts, with particular attributes, are also defined but will not be presented in this paper.

<sup>11</sup> Temporal aspects are not presented in this paper

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.

### 3.2. Case Study

In this section, we illustrate the use of the factorization mechanisms presented in section 2.2 for each part of the co-design modeling. The extensions have been implemented in a separate UML profile.

**3.2.1. 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  $514 \times 514$  signals<sup>12</sup> representing the pixels composing the image, and  $2 \times 2$  signals<sup>13</sup> representing the values of the coefficient matrix. It can send to its environment  $512 \times 512$ <sup>14</sup> 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 produced<sup>15</sup> is determined by the order in which 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.

**3.2.2. 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.4). 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  $2 \times 64$  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.

**3.2.3. Allocation example: Bloc mapping** We specify the distribution of the  $512 \times 512$  potential instances of *t* on the  $2 \times 64$  potential instances of *pe* so that each *pe* receives a bloc of  $512 \times 4$  *t* (Fig.5). Note that the *RepetitiveLinkTopology* is here applied to a *Dependency*.

## 4. Conclusion

As we said in the introduction, regular structures in both the applications and the hardwares will be more and more common in future embedded system designs.

In the context of high-level modeling of such systems based on UML 2, we have presented the definition of common modeling mechanisms for factorizing repetitive structures. We have illustrated the use of these mechanisms with the embedded systems co-design environment *Gaspard2*, for the modeling of application, hardware and software/hardware allocation parts of a sample application.

Such regular modeling constructs are a requirement of the MARTE RFP [22] (Modeling and Analysis of Real-Time and Embedded systems) recently voted by OMG. Our mechanisms will be proposed for standardization of this profile.

## References

- [1] A. Agarwal, S. Amarasinghe, R. Barua, M. Frank, W. Lee, V. Sarkar, D. Srikrishna, and M. Taylor. In *The Raw Compiler Project*.
- [2] D. Barretta, W. Fornaciari, M. Sami, and D. Pau. SIMD extension to VLIW multicluster processors for embedded applications. In *Euro-Par 2000, LNCS 1900*, Oulu, Finland, 2002. iccd.
- [3] D. Barretta, M. S. William Fornaciari, and D. Bagni. Multi-threaded extension to multicluster VLIW processors for embedded applications. February 2005.
- [4] L. Benini and G. D. Micheli. Networks on chips: A new SoC paradigm. *Computer*, 35(1):70–78, 2002.
- [5] M. Berekovic and P. Pirsch. A scalable, distributed network-on-chip architecture for digital signal processing based on simultaneous multithreading(SMT). In *5th Workshop on Media and Streaming Processor*, San Diego, Dec. 2003.
- [6] P. Boulet and A. Meena. The case for globally irregular locally regular algorithm architecture adequation. In *Journées Francophones sur l'Adéquation Algorithme Architecture (JFAAA'05)*, Dijon, France, Jan. 2005.

<sup>12</sup> from its  $514 \times 514$  *dataIn* provided ports.

<sup>13</sup> from its  $2 \times 2$  *coeff* provided ports

<sup>14</sup> via its  $512 \times 512$  *dataOut* required ports

<sup>15</sup> or The order in which the behavior associated to each *t* is executed

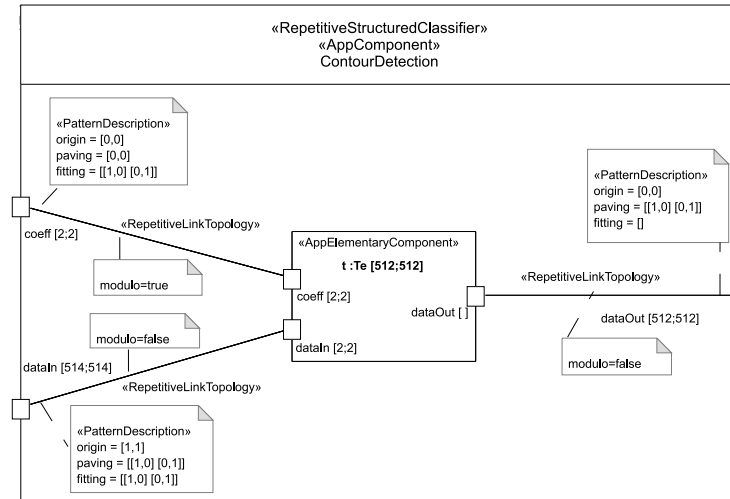


Figure 3. Repetitive application example

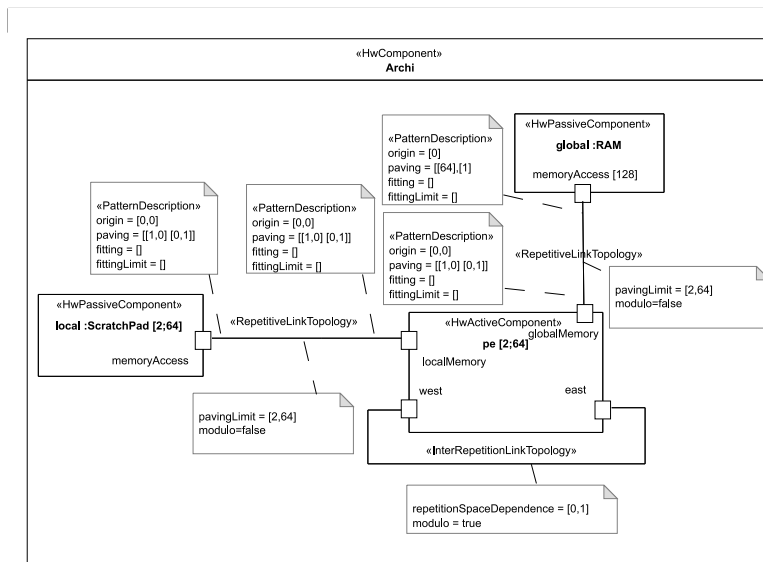


Figure 4. Repetitive hardware architecture example

- [7] Caarls, P. Jonker, and H. Corporaal. Data-and-task parallel image processing on a mixed SIMD-ILP platform using skeletons and asynchronous RPC. pages 27–34, Nieuwegein, Oct 20 2004.
- [8] L. D.A. Access and alignment of data in an array processor. *IEEE Trans. Comput.*, C-24(12):1145–1155, Dec. 1975.
- [9] A. Demeure, A. Lafage, E. Boutillon, D. Rozzonelli, J.-C. Dufourd, and J.-L. Marro. Array-OL : Proposition d'un formalisme tableau pour le traitement de signal multidimensionnel. In *Gretsi*, Juan-Les-Pins, France, Sept. 1995.
- [10] C. Dumoulin. ModTransf: A model to model transformation engine, Nov. 2004. <http://www.lifl.fr/west/modtransf>.
- [11] C. Ebeling, D. C. Cronquist, and P. Franklin. RaPiD - reconfigurable pipelined datapath. In *FPL '96: Proceedings of the 6th International Workshop on Field-Programmable Logic, Smart Applications, New Paradigms and Compilers*, pages 126–135, London, UK, 1996. Springer-Verlag.
- [12] M. Engels and M. Engels. In *Wireless OFDM Systems: How to Make Them Work?*, page 236, July 31 2002).
- [13] M. Forsell. Exploiting parallelism - an essential aspect for future computing. In *5th Annual Telecommunication Systems*

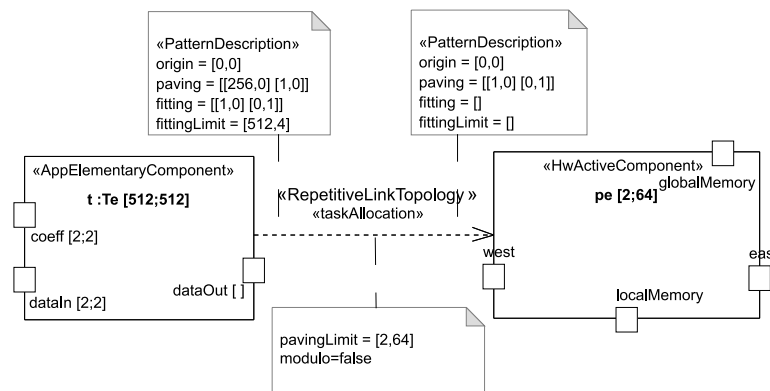


Figure 5. Association example

- Seminar, VTT Electronics, Oulu, Finland, August 2003.
- [14] A. A. Jerraya. Long term trends for embedded system design. In *CEPA 2 Workshop-Digital Platforms for Defence*, Brussels, Belgium, March 15-16 2005.
- [15] T. Kogel and H. Meyr. Heterogeneous MP-SoC: the solution to energy-efficient signal processing. In *DAC*, pages 686–691, 2004.
- [16] S. Kumar, A. Jantsch, M. Millberg, J. Oberg, J.-P. Soininen, M. Forsell, K. Tiensyrja, and A. Hemani. A network on chip architecture and design methodology. In *VLSI '02: Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pages 105 – 112, Washington, DC, USA, 25-26 April 2002. IEEE Computer Society.
- [17] Laboratoire d’informatique fondamentale de Lille. Gaspard home page. <http://www.lifl.fr/west/gaspard/gaspardMaven>, 2005.
- [18] R. Manniesing, I. Karkowski2, and H. Corporaal. Automatic SIMD parallelization of embedded applications based on pattern recognition. Oulu, Finland, 2003. Euro-Par 2000, LNCS 1900.
- [19] B. Mei, S. Vernalde, D. Verkest, H. D. Man, and R. Lauwereins. ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix. In *FPL*, pages 61–70, 2003.
- [20] T. Miyamori and K. Olukotun. REMARC (abstract): reconfigurable multimedia array coprocessor. In *FPGA '98: Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays*, page 261, New York, NY, USA, 1998. ACM Press.
- [21] NEC Corporation. NEC unveils a new class of system LSI solutions, the dynamically reconfigurable processor LSI architecture, at microprocessor forum. <http://www.nec.co.jp/press/en/0210/1601.html>, 2002.
- [22] Object Management Group, Inc., editor. *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>, Feb. 2005.
- [23] O. Ozturk, M. Kandemir, M. J. Irwin, and I. Kolcu. Tuning data replication for improving behavior of MPSoC applications. In *GLSVLSI '04: Proceedings of the 14th ACM Great Lakes symposium on VLSI*, pages 170–173, New York, NY, USA, 2004. ACM Press.
- [24] H. Parizi, A. Niktash, N. Bagherzadeh, and F. Kurdahi. MorphoSys: A coarse grain reconfigurable architecture for multimedia applications. In *Lecture Notes in Computer Science*, volume 2400, page 844, Jan 2002.
- [25] picoChip. PC101 and PC102 datasheets. <http://www.picochip.com/technology/picoarray>, 2003.
- [26] Y. Qian, S. Carr, and P. Sweany. Loop fusion for clustered VLIW architectures. In *LCTES/SCOPE '02: Proceedings of the joint conference on Languages, compilers and tools for embedded systems*, pages 112–119, New York, NY, USA, 2002. ACM Press.
- [27] M. J. Rutten, J. T. van Eijndhoven, E.-J. D. Pol, E. G. Jaspers, P. van der Wolf, O. P. Gangwal, and A. Timmer. Eclipse: A heterogeneous multiprocessor architecture for flexible media processing. In *IEEE Design and Test of Computers Special issue on Embedded Processor Based Designs*, Ed.: Peter Marwedel, July-August 2002.
- [28] M. Schlett. Trends in embedded-microprocessor design. *Computer*, 31(8):44–49, aug 1998.
- [29] C.-R. Soley. Object Management Group, inc. <http://www.omg.org/>.
- [30] D. Talla, L. K. John, V. S. Lapinskii, and B. L. Evans. Evaluating signal processing and multimedia applications on SIMD, VLIW and superscalar architectures. In *ICCD*, page 163, 2000.
- [31] W. Wolf. The future of multiprocessor systems-on-chips. In *41st Conference on Design Automation (DAC'04)*, San Diego, California, USA, June 2004.