INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Modeling of Topologies of Interconnection Networks based on Multidimensional Multiplicity

Imran-Rafiq Quadri — Pierre Boulet — Jean-Luc Dekeyser

N° 6201

28 May, 2007

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--6201--FR+ENG

Modeling of Topologies of Interconnection Networks based on Multidimensional Multiplicity

Imran-Rafiq Quadri* , Pierre Boulet† , Jean-Luc Dekeyser‡

Thème COM — Systèmes communicants
Projet dart

Rapport de recherche n° 6201 — 28 May, 2007 — 53 pages

Abstract: Modern SoCs are becoming more complex with the integration of heterogeneous components (IPs). For this purpose, a high performance interconnection medium is required to handle the complexity. Hence NoCs come into play enabling the integration of more IPs into the SoC with increased performance. These NoCs are based on the concept of Interconnection networks used to connect parallel machines. In response to the MARTE RFP of the OMG, a notation of multidimensional multiplicity has been proposed which permits to model repetitive structures and topologies. This report presents a modeling methodology based on this notation that can be used to model a family of Interconnection Networks called Delta Networks which in turn can be used for the construction of NoCs.

Key-words: SoC, NoC, Multistage Interconnection Networks, Delta Networks, UML 2 Templates

* Imran-rafiq.quadri@lifl.fr
† Pierre.boulet@lifl.fr
‡ Jean-luc.dekeyser@lifl.fr

Unité de recherche INRIA Futurs
Parc Club Orsay Université, ZAC des Vignes,
4, rue Jacques Monod, 91893 ORSAY Cedex (France)
Téléphone : +33 1 72 92 59 00 — Télécopie : +33 1 60 19 66 08

Modélisation de topologies de Réseaux d'Interconnexion à base de Multiplicités Multidimensionnelles.

Résumé : Les Socs modernes sont devenus plus complexes suite à l'intégration de composants hétérogènes (IPs).Un Système évolué d'interconnexion est alors requis pour surmonter la complexité. Les NoCs permettent de bien intégrer les IPs dans les SoC et d'en augmenter l'évolution.Les NoCs sont basés sur le concept des réseaux d'interconnexion utilisés pour relier les machines parallèles. En réponse à MARTE RFP de l'OMG, la notation de multiplicité multidimensionnelle a été proposée pour permettre de modéliser les structures répétitives et les topologies .Ce rapport présente une méthodologie de modélisation basée sur cette notation et qui peut etre utilisée pour modéliser des réseaux d'interconnexion appelés "Réseaux Delta" qui permettraient par la suite la construction de NoCs.

Mots-clés : SoC, NoC, Réseaux d'Interconnexion Multi étage, Réseaux Delta, mécanisme générique d'UML 2

Contents

1 Introduction

A System-on-chip (SoC) is defined as the integration of a complete system on a single silicon chip. With the simultaneous increase in performance and miniaturization of integrated circuits, SoCs are becoming more and more flexible and complex. The complexity of Interconnection Networks is also an important issue as the problems linked to the propagation delays in the interconnections are amplified by the reduction of the technological dimensions.

Networks-on-Chip (NoC) are thus considered as a solution for these problems. In summary, they can be seen as a layered approach of communication similar to the Interconnection Networks (INs) [31] defined by the communication networks community to address the problem of connecting a large number of computers on a wide-area, but used for on-chip communications. Reusability of communication networks and resources (whether a single processor or cluster of processors, memories, switches, etc.) is an important advantage of NoCs. We focus specifically on the Delta networks which are included in the family of Interconnection Networks and more precisely Multistage Interconnection Networks (MINs). The layered approach employed in the design of NoCs permit to separate the computation and communication resources. NoCs undeniably incur a significant change in the SoC architecture and improve the design productivity of these complex systems.

Model driven engineering at the OMG's MDA [4] is concerned with the development of standards and technologies to support the design of model based systems. These standards and their technological support provide the techniques and the tools necessary to address the dilemma between cost reduction and system development time, and help to design massively complex systems.

GASPARD 2 is an Integrated Development Environment for visual co-modeling of SoCs. It is based on a model driven approach and permits modeling, simulation and code generation of SoC applications and hardware.

In this report, we are interested in utilizing the existing GASPARD 2 UML Profile [12], [23] for the modeling of Multistage Interconnection Networks, mainly Delta Networks. We use the UML 2 Template mechanism for this purpose and expand it for our need.

The remainder of this paper is organized as follows: section 2 briefly explains the GASPARD environment. We then recall some basics of MINs and Delta networks. The UML 2 template mechanism is presented in section 4. Section 5 illustrates the modeling of certain Delta networks before the last section where we conclude our work.

2 GASPARD 2 Environment

GASPARD 2 [6, 41] is a co-modeling environment supporting the design of massively parallel SoCs. It is based on a model driven methodology in accordance with the Y design flow. It proposes a UML profile allowing designers to model both massively parallel applications and their architectures. An association mechanism is provided to link the two aspects together with a set of transformations for simulation and synthesis. Initially, application, hardware architecture and association are modeled using the GASPARD 2 UML profile. Then these models are analyzed and transformed by applying mapping and scheduling algorithms and automatic SystemC code generation. A component oriented methodology is used as a foundation for the model definitions of the GASPARD 2 environment. This

methodology allows separation of the different parts of the Y model and makes possible the re-use of existing software and hardware IPs.

3 Multistage Interconnection Networks

As the complexity of software increases along with its scale and solution quality, demand for faster high processors have also increased. Specially in the areas of aerospace, defense, automotive applications and science, grand challenge problems are present which require tremendous amount of computational power ranging from Gigaflops (10^9 floating point instructions per second) to Teraflops (10^{12} floating point instructions per second). This is where parallel computers with multiple processors come into play. Parallel computing, or parallelism, can be defined as the usage of more than one processing unit in order to solve one particular problem. A parallel architecture can be defined as an explicit, high-level framework for the development of parallel programming solutions by providing multiple processors, whether simpler or complex, which cooperate to solve problems through concurrent execution.

Communication subsystems play a very significant role in the today's parallel computers. These subsystems are used to interconnect the various processors, memories, disks and other peripherals. The specific requirements of these communications subsystems depend on the architecture of the parallel computer.

Multistage Interconnection Networks (MINs) are widely used in parallel multiprocessors systems to connect processors to processors and/or to memory modules. Their popularity is due to the high switching cost of crossbar networks. Various topologies of MINs have been proposed ad studied in the last few decades. Most of these topologies are derived from the well know undirected graph topologies including mesh, star, shuffle exchange, tree networks and cube-connected networks, among various others.

3.1 Parallel Architecture and Memory Organization

In a multiprocessor system, also called shared memory, all processors share the same memory space. In order to permit parallel access to this shared memory, it is divided into several memory modules. The granularity of the memory system is defined by the size of the memory modules. Granularity is an essential aspect in the design of a parallel architecture. There must also be a medium present which allows the processors to share memory and also should be capable of transferring data among all processors and memory modules. Shared memory parallel computers are distinguished by their programming facility.

Every PE (Processing Element) in a distributed memory system, also called multi-computer, has its own memory; and data access to another memory node is achieved by communicating with the processors connected to it.

Figures.1 and .2 shows the difference between the two systems. Note that in modern shared memory systems, each processor has a small cache memory which is not accessible directly by other processors.

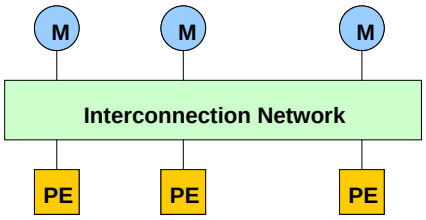

Figure 1: *Shared Memory Abstraction*

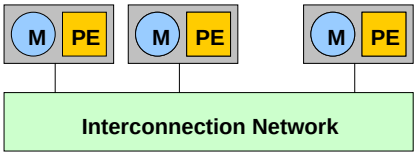

Figure 2: *Distributed Memory Abstraction*

In both architectures, a communication medium (interconnection network) is used to connect different nodes of the system. In the distributed memory system, it links the different processors using a message passing network, and for the shared memory architecture, it connects processors to processors and/or to memory modules.

When more than one PE needs to access a memory module for a read or write operation, conflicts might arise. In a parallel system, conflicts can also occur in the communications systems. The following is a brief discussion of conflicts in parallel computers.

3.2 Conflicts in Parallel Architectures

In a parallel computer communication system, a conflict occurs when more than one message tries to utilize the same communication medium. We call a communication resource a link or a Switching Element (SE) output; and in a buffered communication system, an input buffer. When a conflict occurs in a buffered system, one message passes to its destination and others are queued in order to be routed in the subsequent cycles. In an unbuffered system, or in case of full buffer, conflicts cause only one message to pass and other messages are rejected and can be retransmitted later.

Three types of conflicts can occur in parallel computers [22] : Network conflicts, bank busy conflicts, and simultaneous bank conflicts. The last two can be grouped as to form memory conflicts. The memory conflicts can be removed by a technique called data skewing [7] which causes data arrangement in memory.

When a memory conflict is unavoidable, consistency rules are used. One rule is the EREW (Exclusive Read, Exclusive Write) where only one processor can execute an R/W operation on the same memory bank at the same time. On the other hand, CRCW (Concurrent Read, Concurrent

Write) enables more than one processor to read/write data from/to the same memory module at the same time. Conflicts on write requests can be solved by special algorithms. A practical solution is the usage of CREW (Concurrent Read Exclusive Write) mechanism.

Practical parallel systems use different techniques in order to avoid or resolve conflict issues. Some practical examples are given below to give an idea about the differences present between machines belonging to same family or having same architecture model. We focus on two architecture families: SMP Machines and MPSoCs.

3.3 SMP

SMP (Symmetric Multiprocessors or Shared Memory processors according to some references) architectures as well as their NUMA extensions are used to build nearly all parallel servers. They have advantages of symmetry, unique address space and low communication latency.

SMP also do not suffer from the problems of Asymmetrical parallel systems where the unavailability of the master processor may lead to a degradation of system performance or even total system blockage. Communications in SMP are simple load/store operations.

Cache coherence is controlled, generally, by the hardware. For multicomputer systems, communication tasks may be more difficult, as they have to take place between different processors. A detailed example of an SMP architecture performance evaluation study can be found in [26] .

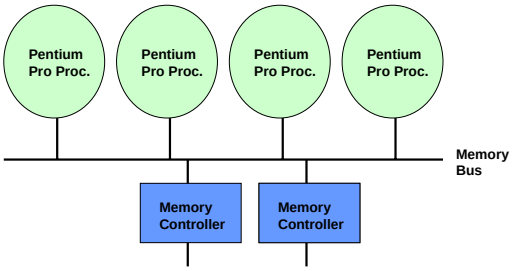

Figure 3: *The Intel SVH SMP Board Architecture*

There are also cases of SMP machines, where the communication system is a simple or improved bus. An example is of the Intel Standard High-Volume (SHV) Server. Figure.3 shows only the communication system to connect the processors to the memory modules. The system supports cache coherence. It was the result of collaboration between Microsoft and Intel which led the latter to built Windows NT SHV systems with more than 4 processors [27] . However due to scalability problems with NT, Windows 2000 was selected as it had better processor and memory managing capabilities. Yet due to traffic bottle neck on the bus, most of the systems supporting this technology are limited to 4 or 8 processors.

In order to avoid the bottleneck caused by the use of bus, IBM proposed an architecture with a bus for the snoopy and a switch (crossbar) used for the interconnections among the processors as well as their communications with the memory [16]. This was a Crossbar-Bus Hybrid structure. The switch allows multiple parallel communications which are faster than those that can be routed on a bus. The use of a crossbar allows increasing of the memory bandwidth by providing multiple buses.

3.4 Multiprocessor System-on-Chip (MPSoC)

A System-on-Chip (SoC) is defined as an integration of a complete system on only one silicon chip. No exterior software or hardware components are expected to interfere in the task execution of a SoC.

SoCs are a novel possibility for the construction of computing systems as they solve a key problem of memory latency, from which the traditional systems suffer greatly. 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). Even if the memory access time of a SoC is still much higher than a processor timing cycle, it is still remarkably less compared to that of a traditional computer. Also, with evolving technology, it is possible to increase the performance of the processors and/or multiprocessing.

The study of the communication system of a MPSoC is a recent research branch dealing with what is called Network on Chip (NoC) [21]. The on-chip communication backbone connects a large number of heterogeneous or homogeneous processing clusters and memory modules.

NoCs can be seen as a layered approach of communication similar to that defined by the communications networks community to address the problem of connecting a large number of computers on wide-area, and is used for on-chip communication design. An advantage of the NoC based designs is that they do not need to be synchronized with other subsystem cores like in bus based MPSoC designs. An other advantage of NoC is that it provides better power aware optimizations.

While bus structures in SoC such as [2], [11] and [20] are attractive because of their simplicity and crossbars because of their performance, neither are totally practical solutions for large scale computer systems. An intermediate solution is the interconnection networks. They are presented in the next subsection.

3.5 Interconnection Networks

As described earlier, communication between the different PEs themselves and/or communication with the memory system must be carried out by means of a medium. In fact, interconnecting processors and linking them effectively to the memory modules in a parallel computer is of paramount importance. However, this task is a complicated one, due to the complexity/performance tradeoff that must be made.

Use of bus architectures is not a practical solution, because bus is only a good choice when the number of connected components is small. However, as the number of components in SoC is increasing with time, it seems that a simple bus is no longer a preferable solution for SoC intercon-

nection requirements. They are not scalable, testable and have lack of modularity, resulting in poor fault tolerance performance of buses.

On the other hand, a crossbar as shown in figure.4, which provides full interconnection between all the nodes of a system is deemed very complex, expensive to design, and hard to control. For this reason, Interconnection Networks (INs) [31] are considered a good communication medium for parallel systems. They limit the paths between different communicating nodes in order to minimize the switch complexity, while giving a certain level of parallelism which is superior to that of a bus.

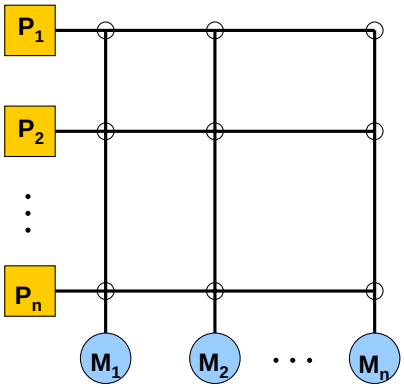

Figure 4: *The Crossbar Architecture*

Functionally, the role of an IN in a parallel system is to transfer *information* between the source nodes to the destination nodes. The following section lists the most important characteristics of an IN from a high level architecture point of view. An architectural classification of INs is proposed and then it is surveyed. After that, some networks of special interest for the dissertation, are presented in detail.

3.5.1 Characteristics of INs

In general, an Interconnection network is characterized by its topology, communication (switching) strategy, synchronization philosophy [3], control strategy, and routing mechanism [13]. Some informal definitions of the properties of Interconnection networks are given ahead.

Topology
The physical structure of an Interconnection network is defined by its topology. The topology of an Interconnection network is defined mathematically by a graph $G = (V,E)$, where V is a set of nodes (processors, memory modules, computers and/or intermediate SEs) and E is a set of links. It is evident that the routing algorithm, which defines the path of a message to be routed between a source

and a destination, depends largely on the network topology.

Switching Strategy
Basically, two switching strategies are used, circuit switching and packet switching. In the former, the whole path between the source and the destination of a message has to be reserved before the communication takes place and this reservation has to be valid until the message reaches its destination.

In the latter, a message is divided into a number of information sequences, of the same or different sizes called packets. These packets are routed individually to their destinations. The transmission is established by steps. Only the path between intermediate nodes must be reserved at each step of a communication.

While modern telephone switching systems use packet switching, circuit switching was largely used for INs. Today, modern optical parallel and communication systems use circuit switching because of technical difficulties imposed by packet switching in optical systems. Improved communication strategies based on these basic strategies can be found in literature.

Synchronization
In a synchronized interconnection network, a central clock controls the operation of SEs and I/O nodes. Handshaking strategies are needed in asynchronous systems.

Control Strategy
The control of a network can be centralized or distributed. In a centralized control strategy, a central controller must have at each moment, all the information concerning the global state of the system. It will generate and send control signals to different nodes of the network according to its collected information. Obviously, the complexity of such a system increases rapidly with the increase in the number of nodes and its breakdown causes the whole system to stop. In contrast, routed messages on non centralized networks (also called self routing) contain necessary routing information. This information is added to the message and will be read and used by the SEs [35].

Routing Algorithm
The routing algorithm defines, depending on the source and destination of the message, the interconnection links to be used while traversing through the network. Routing can be adaptive or deterministic. Paths with deterministic routing mechanisms can not be changed according to the existent traffic in the network.

Before describing the architecture of Interconnection networks, some classical definitions must be mentioned. They are presented in the following section.

3.5.2 Related Definitions

In order to consider the functionality of an IN, some classifications and definitions should be recalled as first defined in [1].

An *Alignment IN* is a network capable of providing access to a certain number of data structures with maximum performance.

A *permutation IN* is a network in which all $N!$ permutations can be realized where N is the number of inputs and outputs of the network.

An IN is characterized by its *size* and *degree*. By size, we mean the number of inputs and outputs of an IN where as the degree of a IN is defined as the size of SEs used to build it.

3.5.3 Classification of Interconnection Networks

Interconnection networks can be either static, dynamic or hybrid in nature. Hybrid networks are those INs which have complicated structures such as hierarchical or hyper graph topologies. In the following sections we will present the two network families of static and dynamic networks. Since we are only concerned with Dynamic INs in the report, the section related to static networks is not exhaustive and only Dynamic Interconnection networks are explored in detail. Fig.5 shows the overall classification of Interconnection Networks.

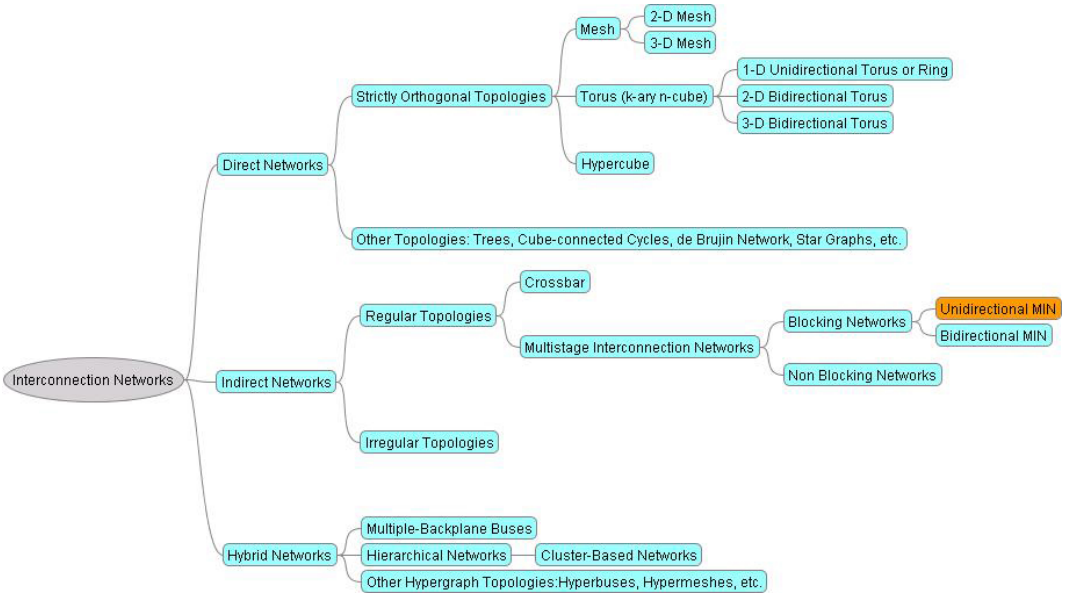

Figure 5: *Classification of Interconnection Networks*

3.5.4 Static Interconnection Networks (Direct Networks)

In a Static Interconnection network, links among different nodes of the system are considered *"passive"* and *"only graph theoretical adjacent processors can communicate in a given step"* [5]. Thus each node is directly connected to a small subset of nodes by interconnecting links. Each node performs both routing and computing. Important topology properties of the network include:

- *Node Degree*: (the number of links connected to the node linking the node to its neighbors);
- *Diameter*: (the maximum distance between two nodes in the network);
- *Regularity*: (a network is regular when all its nodes have the same degree);
- *Symmetry*: (a network is symmetric when it looks the same from each node's perspective) and
- *Orthogonal property*: (a network is orthogonal if its nodes and interconnecting links can be arranged in n dimensions such that the link is placed in exactly one dimension). In a weakly orthogonal topology, some nodes may not have any link in some dimensions.

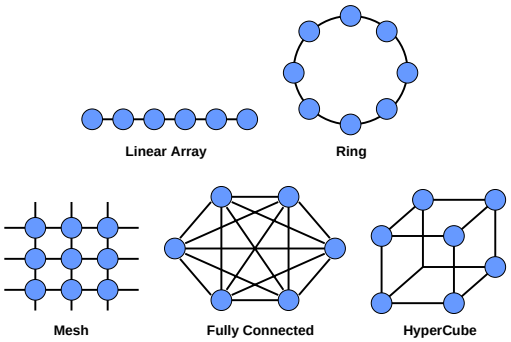

Figure 6: *Some Examples of Static Networks*

In static networks, the paths for message transmission are selected by routing algorithms. The switching mechanisms determine how inputs are connected to outputs in a node. All the switching techniques can be used in direct networks.

The simplest static network is the bus. As described earlier, the use of a simple bus is not a practical choice for parallel computers as only one message can be transferred at a time and improved bus architectures such as hierarchical buses, cannot afford acceptable level of parallelism. Other static INs such as shown in fig.6 can contain among others, linear arrays, rings, meshes, hypercube, trees, etc. In a linear array, each processor is connected to its two neighbors.

3.6 Dynamic (Indirect Networks) and Multistage Interconnection Networks

As compared to static networks, in which the interconnection links between the nodes are passive, the linking configuration in a Dynamic IN is a function of the SEs states. In layman terms, the paths between the graph nodes of a Dynamic IN change as the SEs states change. As Dynamic networks are built using crossbars (especially of size 2×2); fig.7 illustrates the different states of crossbars of size 2×2.

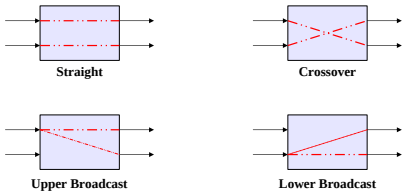

Figure 7: *Different states of Crossbars* 2×2

3.6.1 Single Stage INs

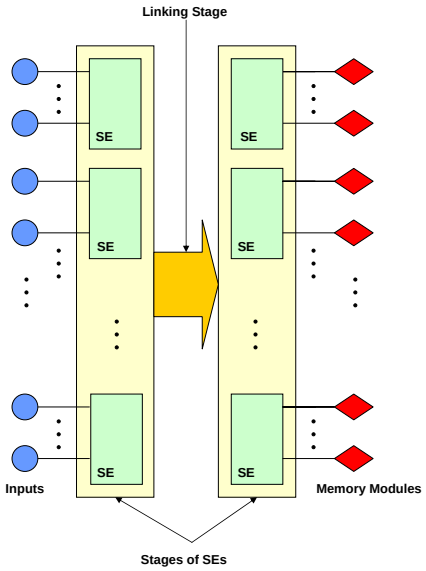

Figure 8: *The schematic of a single stage MIN*

Dynamic INs can be built as a single stage or Multistage INs (MINs). A single stage IN is a dynamic network composed of one linking stage and two end SE stages. It should be noted that in some references, a single stage IN is composed of only one switching stage and one linking stage. Figure.8 shows a general schematic of a single stage IN. Crossbars, which provide a full connection between all nodes of the system, are considered as non blocking single stage intercommunication networks for parallel computers.

The linking stage in the figure is a permutation function connecting the outputs of the SE to the stage furthest to the left to the inputs of the others SE stage. It should be noted that more than one path through the network may be required for effective communication between a source and a

destination. Also, not all permutation configurations can lead to a connected network, i.e. capable to connecting any source to any destination. A study of such single stage INs can be found in [8].

3.6.2 Multistage Interconnection Networks

A MIN can be defined as a network used to interconnect a group of *N* inputs to a group of *M* outputs through a number of intermediate stages of small size SEs followed (or leaded) by interconnection linking stages.

More formally, a MIN is a succession of stages of SEs and interconnection links. SEs in their most general architecture are themselves small size interconnection networks. The most used SEs are hyperbars and more specifically crossbars.

Linking stages are *interconnection functions* [32], each function is a bijection of the group of the previous stage switches addresses which connect all SEs outputs from a given stage to the inputs of the next stage.

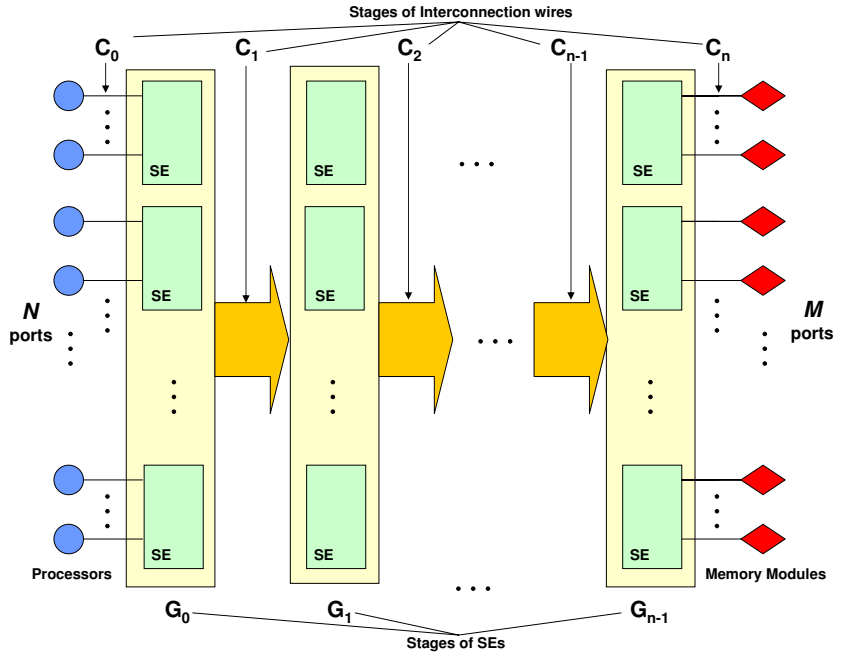

Figure 9: *Architecture of a Multistage Interconnection Network*

In a multiprocessor environment, the first stage of links is connected to the sources (usually processors) and the last stage is connected to the destinations (memory modules). The minimum numbers of stages of a MIN must provide a full connection of input nodes to output nodes. Formal definitions of generalized self routing MINs are given in [32]. A generalized MIN is shown in the

fig.9. The SEs in MINs may have input and/or output buffers. The buffers serve as temporary storage for blocked messages when conflicts occur. In this case, the MIN is called a buffered MIN. Only MINs in their simplest configuration (without buffers) are considered in this report.

As shown in the figure, the MIN has N inputs, and M outputs. It has n stages, G_0 to G_{n-1}. Each stage G_i has w_i switches of size $a_{i,j} \times b_{i,j}$ where $1 \leq j \leq w_i$, thus stage G_i has p_i inputs and q_i outputs such that

$$p_i = \sum_{j=1}^{w_i} a_{i,j} \text{ and } q_i = \sum_{j=1}^{w_i} b_{i,j} \tag{1}$$

A connection pattern between two adjacent stages, G_{i-1} and G_i ,denoted C_i, defines the connection pattern for $p_i = q_{i-1}$ links where $p_0 = N$ and $q_{n-1} = M$. Thus a MIN can be represented as

$$C_0(N)\, G_0(W_0)\, C_1(p_1)\, G_1(W_1) \ldots\, G_{n-1}(W_{n-1})\, C_n(M) \tag{2}$$

Where C_0 is the connection pattern from sources to the first switching stage and C_n is the connection pattern from the last switching stage to the destinations. A connection pattern $C_i(p_i)$ defines how those p_i links should be connected to the $q_{n-1} = p_i$ outputs from the stage G_{i-1} and the p_i inputs to the stage G_i. Different connection patterns give different characteristics and topological properties of MINs. The links are labeled from 0 to $p_i - 1$ at C_i.

3.6.3 Classification of MINs

We propose a classification of MIN and restate some definitions necessary for the proposed classification. The topological classification of MINs based on the following definitions is given in figure.10.

A uniform MIN is one, in which all the switching elements of a stage are of the same degree.

A rectangular network is one that has the same number of inputs and outputs.

A Square MIN is one, in which a MIN of degree $\mathbf{r}$ *is built from SEs of size* $\mathbf{r}$.

MINs have been classified into three classes depending on the availability of paths to establish new connections. They are:

1. *Blocking*. A connection between free input/output pair is not always possible because of conflicts with existing connections. Typically, there is a unique path between every input/output pair, thus minimizing the number of switches and stages. A uni-path network is also called a Banyan Network.

   A Banyan network is defined as a class of multistage interconnection networks in which there is one and only one path from any input node to any output node.

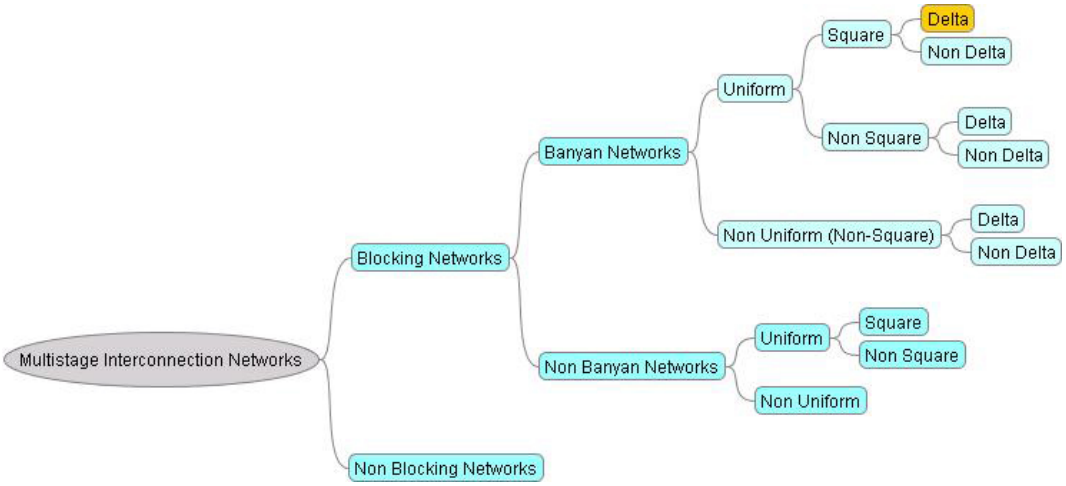

Figure 10: *Classification of MINs based on the definitions*

One of the most critical issues concerning an IN topology is the existence or absence of multiple paths. By providing multiple paths in Blocking networks, conflicts can be reduced and fault tolerance can be increased. These Blocking networks are also known as *multipath networks*.

2. *Non Blocking*. Any input can be connected to any free output port without affecting the existing connections. They require extra stages and have multiple paths between every input and output. A popular example of Non-blocking networks is a Clos network [9].

3. *Rearrangable*. Any input port can be connected to any free output. However the existing connections may require rearrangement of paths. These networks also require multiple paths between every input and output, but the number of paths and the cost is smaller than in the case of Non blocking networks.

Depending on the kind of channels and switches, MINs can be either:

1. *Unidirectional MINs*. Channels and switches are unidirectional.

2. *Bidirectional MINs*. Channels and switches are bidirectional. This implies that information can be transmitted simultaneously in opposite directions between neighboring switches.

Additionally, each channel can be either multiplexed or be replaced by two or more channels. The latter case is referred to as a *dilated MIN*. Since we are concerned only with unidirectional Delta Square Uniform Banyan networks which are a subset of Banyan networks, we only specify the characteristics of unidirectional MINs now.

Usually the inputs of an MIN are equal to the number of the outputs, i.e. $N = M$, however, this may not always be the same case as we will see later on in the dissertation.

For MINs with $N = M$, it is common to use switches with the same number of inputs and output ports, that is $a = b$. If $N > M$, switches with $a > b$ will be used. Such switches are also called *concentration switches* [18]. In the case of $N < M$, distribution switches with $a < b$ will be used.

It should be noted that with N inputs and M outputs, a unidirectional MIN with $k \times k$ elements needs at least $\log_k N$ interconnection functions to be implemented. For this reason, an N size MIN contains at least this number of stages, otherwise, some sources may never be connected to some destinations.

Delta Networks, which form a subset of Banyan networks, were first proposed by Patel [28, 29]. Every Delta network is a Banyan network while the reverse is not always true. Delta networks have a routing property called *Self routing Property* or *Delta Property*. Delta networks are explained in detail in the following subsection. Figure 11 shows a Delta network.

Since we concern ourselves with only Delta networks, therefore, Non-Delta Banyan networks are not of interest for this report. Uniform Banyan MINs can be either square or non square. Thus according to the above mentioned definitions, a non uniform network is also non square. A DSUB [Delta, Square (also called SW in some references), Uniform, Banyan] network is a delta network with all its SEs having the same size and is the focus of this research.

Normally, a DSUB network uses a size of power of 2, however, a Delta network can be built which does not have a size based on the power of 2 i.e. a DnSUB MIN. Figure.12 demonstrates an Omega network (a type of Delta network) of size 6.

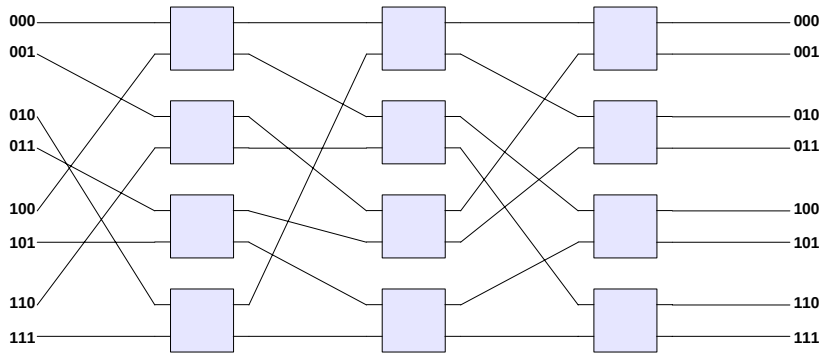

Figure 11: *A Typical Delta Network*

Non Banyan networks are more expensive and complex to control then Banyan networks. Yet they may offer fault tolerance and may solve some conflicts. They are usually constructed by augmentation of a Banyan network or by construction of a multipath network such as Clos MIN.

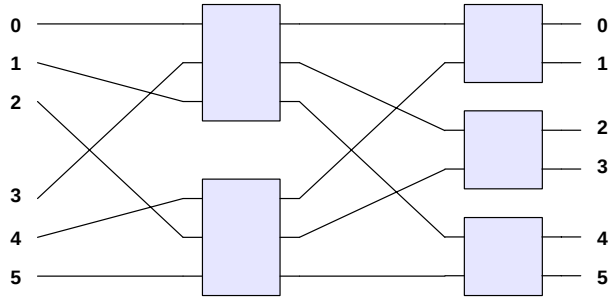

Figure 12: *A Delta Network of size 6*

3.6.4 Delta Networks

In the formal definition given by Patel [28, 29], Delta networks are built using $a^n \times b^n$ (where n is the number of stages) digit controlled crossbars of which no input and output can be left unconnected. The total number of crossbars required to construct a Delta MIN is:

$$\sum_{1 \le i \le n} a^{n-i} \times b^{i-1} = \frac{a^n \times b^n}{a - b}; a \neq b \tag{3}$$

$$= n\, b^{n-1}; a = b \tag{4}$$

It should be noted that a N-node Delta Network ($N = k^n$) contains n stages where each stage contains $\frac{N}{k}$ switches. The delta or self routing property of Delta MINs allows automatic self routing of messages from a source to a destination.

The self routing property of Delta networks as shown in figure.13 allows the routing decision to be determined by the destination address, regardless of the source address. Self-routing is performed by using routing tags. For a $k \times k$ switch, there are k output ports. If the value of the corresponding routing tag is i where $0 \le i \le k - 1$, the corresponding packet will be forwarded via port i. For an n-stage MIN, the routing tag is $T = t_{n-1} \ldots t_1\ t_0$, where t_i controls the switch at stage C_i. A mathematical generalization of Delta property is given in [1].

There may exist, large numbers of link patterns available for $a^n \times b^n$ delta network. It should be noted that the probability of acceptance or blocking of messages is identical in all delta networks. That means one type of Delta network can be replaced by another one.

As a result, each different setting of $a \times b$ switch generates $(b!)^{nb^{n-1}}$ distinct permutations. This is a small fraction of possible permutations. For example, the probability that a random permutation of 32 inputs can be generated by a $2^5 \times 2^5$ delta network is 4.6×10^{-12} [28].

In order to simplify the construction of a Delta MIN as well as the design of a routing algorithm, Patel proposed a *regular* link pattern which can be used between all stages and thus avoid the difficult construction procedure for every different delta network. Patel termed the regular link pattern : the q-shuffle. The q-shuffle of a group of qr elements is a permutation of these elements defined by:

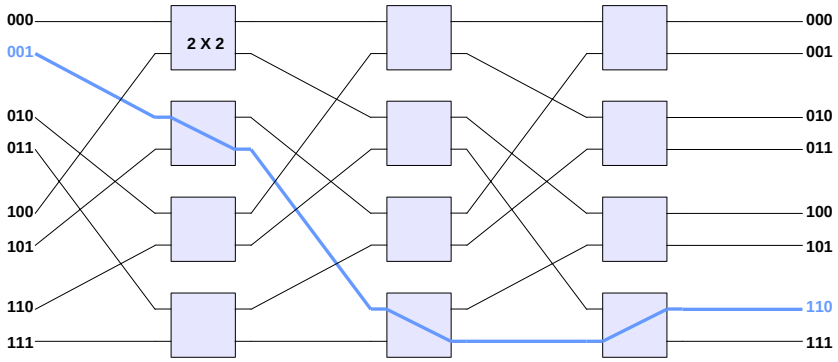

Figure 13: *Self Routing Property of Delta Networks*

$$S_{q\times r}(i) = (qi + \lceil \frac{i}{r} \rceil) \; mod \; qr; 0 \leq i \leq qr - 1 \tag{5}$$

Alternatively, the same function can be expressed as:

$$S_{q\times r}(i) = qi \; mod \; (qr - 1); 0 \leq i < qr - 1 \tag{6}$$

$$= i \qquad\qquad i = qr - 1 \tag{7}$$

A q-shuffle of qr playing cards can be viewed as follows. Divide the deck of qr cards into q piles of r cards each; top r cards in the first pile, next r cards in the second pile and so on. Now pick the cards, one at a time, from the top of each file; the first card from the top of pile one, second pile from the top of pile two, and so on in a circular fashion until all the cards are picked up. This new order of cards represent a $S_{q\times r}$ permutation of the previous order. For determining the values of q and r for an Delta MIN of size $N \times M$, we use the formula $S_{a\times b^{n-1}}$, and a and b corresponding to the inputs and outputs of the crossbar, and n corresponding to the number of stages. The final values in place of a and b^{n-1} are taken as values of q and r for the q-shuffle formula. Consider, a Delta MIN of size 16×9 as shown in the fig.14, i.e. a $4^2 \times 3^2$.

Putting the values in the formula, we obtain the values of 4 and 3 for q and r respectively. Thus we obtain a $S_{4\times3}$ function, figure.15 shows an example of 4-Shuffle of 12 indices, which corresponds to the $S_{4\times3}$ function. As it can be observed, the link pattern is the same in both figure.14 and figure.15.

Furthermore, applying the q-shuffle function on a number represented in base q corresponds to the application of a cyclic shift on said numbers. This leads to a construction of a class of MINs called *"shuffle-exchange MINs"* [28], [37]. Omega networks [24] which were first defined by Laurie, and one of the most popular types of Delta networks are usually described as shuffle exchange MINs. In fact all delta networks are *shuffle-exchange* MINS. An exchange function is defined by changing the least significant digit of the output address [19].

The q-shuffle is the general case, of which the perfect shuffle [37] is a special case. Further explanation of perfect shuffle will be provided in the subsection describing Omega networks which utilize it.

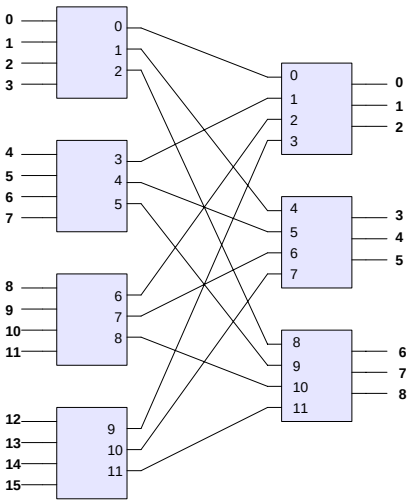

Figure 14: *Delta Network of size* 16×9

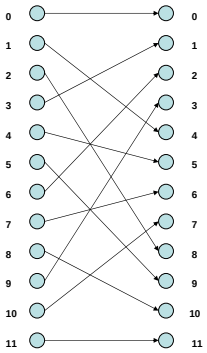

Figure 15: *A 4-Shuffle of 12 indices*

In the next subsection, we describe the different types of Delta networks which are considered the most popular. It should be noted that we focus primary on the DSUB network which use a size of power of 2, i.e. Delta MINs which have crossbars of size 2×2.

3.7 Types of Delta networks

There exists various popular MINs which we have grouped to be considered as different types of Delta networks. The difference between each of these networks is the topology of interconnection links between the crossbar stages. A study of equivalence of various types of Delta MINs has been studied in [10]. All delta networks are considered to be topologically equivalent as well as functionally equivalent [42]. We thus classify the following popular types of Delta MINs.

-:- Omega networks

-:- Butterfly networks

-:- Baseline networks

-:- (Generalized) Cube networks

and their reverse networks

-:- Flip networks

-:- Reverse Butterfly networks

-:- Reverse Baseline networks

-:- Indirect Binary n-cube networks

In current literature, usually the first four network types are discussed. This is due to the reason because the last four types are mirror images of the first four types respectively, (i.e. Flip network is a reverse image of Omega network and so on). In the report, we also explain the last four types in the respective sections of the first four types. We assume that these networks are built using $k \times k$ switches, and that there are $N = k^n$ inputs and outputs, however some of the permutations for the interconnection links are only defined for the case where N is a power of 2. With $N = k^n$ ports, Let $X = x_{n-1}\ x_{n-2}\ \ldots\ x_0$ be an arbitrary port number, $0 \leq X \leq N-1$ where $0 \leq x_i \leq k-1$ and $0 \leq i \leq n-1$.

3.7.1 Omega Network

Omega networks are considered to be the most popular of Delta networks. They use the perfect shuffle which is a special case of a q-shuffle. A more intelligent way to describe the perfect k-shuffle permutation σ^k defined as:

$$\sigma^k(x_{n-1}\ x_{n-2} \ldots\ x_1\ x_0) = x_{n-2} \ldots\ x_1\ x_0\ x_{n-1} \tag{8}$$

The perfect k-shuffle permutation performs a cyclic shifting of the digits in x to the left for one position. For $k = 2$, a perfect shuffling of a deck of N cards take place. The perfect shuffle cuts the deck into two halves from the center and intermixes them evenly. Figure.16 and figure.17 show schematics of Omega (16,4) and Omega (16,2) respectively. (Here the first parameter refers to the

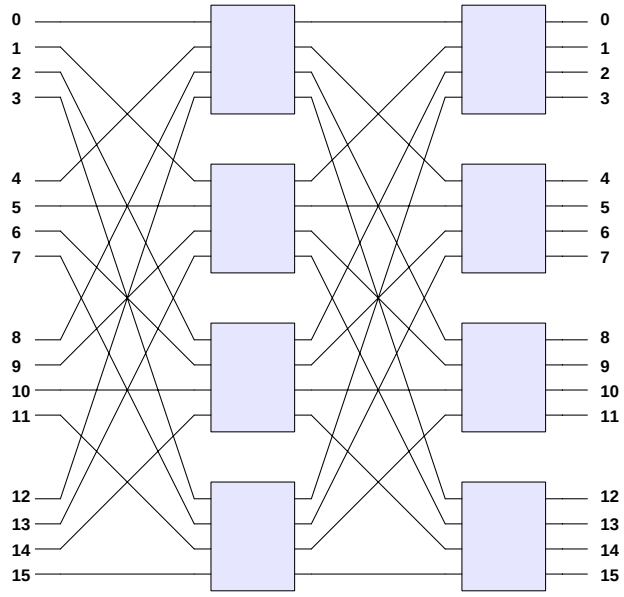

Figure 16: *Omega (16,4) Network*

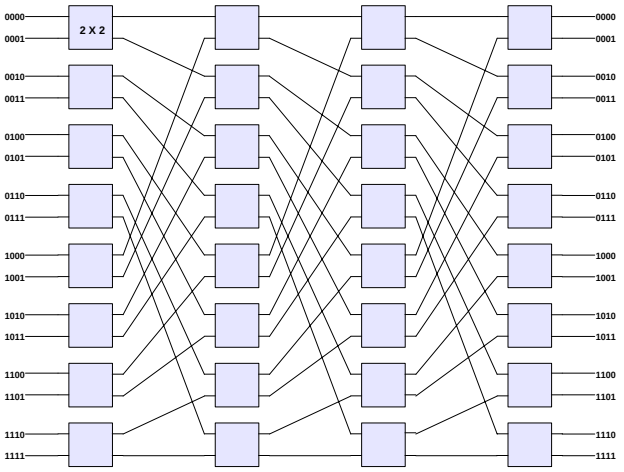

Figure 17: *Omega (16,2) Network*

size of the network and the second parameter corresponds to its degree. It should be noted that the

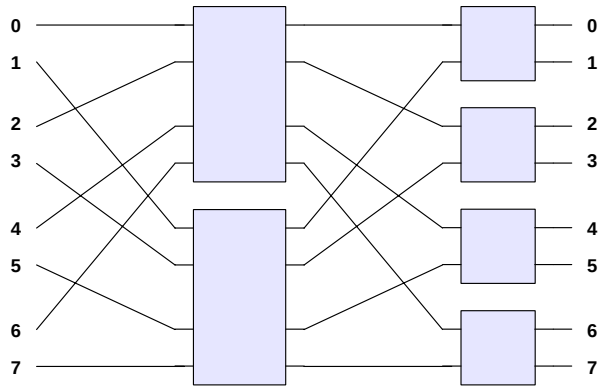

Figure 18: *An Omega Network using* 4×4 *and* 2×2

values of q and r as defined in k-shuffle formula are respectively $S_{4\times4}$ and $S_{2\times2}$ for these Omega networks.

Normally an Omega network has the same interconnection links between the crossbar stages as seen in the mentioned figures. However, as described by Laurie [24], an Omega network can be constructed with different interconnection links between the switching stages as shown in the figure.18.

In an Omega network, connection pattern C_i is described by the perfect k-shuffle permutation σ^k for $0 \leq i \leq n-1$. Connection pattern C_n is selected to be β_0^k. Thus, all the connection patterns but the last one are identical. The last connection pattern which is β_0^k does not produce any permutations. Therefore the patterns can be summed up as

$$C_i(0 \leq i \leq n-1), \sigma^k; C_n, \beta_0^k \tag{9}$$

A Flip network [3] is considered to be a mirror image of the Omega network. It uses the *inverse perfect shuffle permutation* $\sigma^{k^{-1}}$ which is defined by

$$\sigma^{k^{-1}}(x_{n-1}\ x_{n-2} \ldots\ x_1\ x_0) = x_0\ x_{n-1} \ldots\ x_2\ x_1 \tag{10}$$

The connection patterns of Flip network are

$$C_0, \beta_0^k; C_i(1 \leq i \leq n), \sigma^{k^{-1}} \tag{11}$$

Fig.19 shows a Flip (8,2) network.

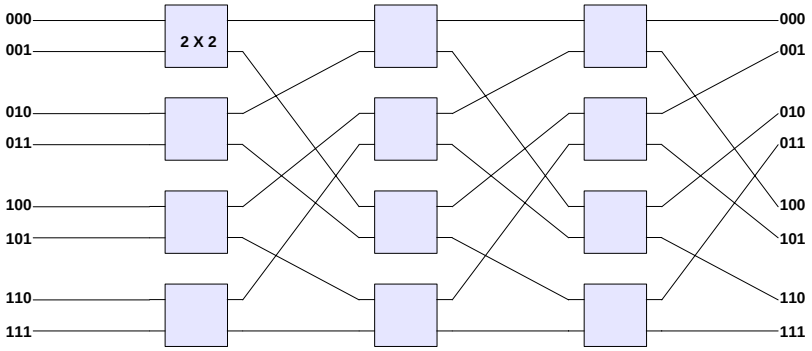

Figure 19: *A Flip Network of size* 8×8

3.7.2 Butterfly Network

A Butterfly network is basically an unfolded hypercube. The dimensions of the hypercube correspond to the number of interconnection links between the crossbar stages of the Butterfly networks. The ith k–ary butterfly permutation β_i^k, for $0 \leq i \leq n-1$, is defined by

$$\beta_i^k(x_{n-1}\ x_{n-2} \ldots\ x_{i+1}\ x_i\ x_{i-1} \ldots\ x_1\ x_0) = x_{n-1}\ x_{n-2} \ldots\ x_{i+1}\ x_0\ x_{i-1} \ldots\ x_1\ x_i \tag{12}$$

$$C_0, \beta_0^k; C_i(1 \leq i \leq n-1), \beta_{n-i}^k; C_n, \beta_0^k \tag{13}$$

The ith butterfly permutation interchanges the zeroth and ith digits of the index. It should be observed that β_0^k defines a straight one-to-one permutation and is also called *identity permutation*. In a Butterfly MIN, connection pattern C_i is described by the ith butterfly permutation β_i^k for $0 \leq i \leq n-1$. Connection pattern C_n is selected to be β_0^k. Thus the patterns can be summed up as

A Butterfly (16,2) network is shown in the fig.20. It should be observed that in some references, this schematic is represented as a Reverse butterfly network.

An important point to be considered is that except Omega and its reverse network, all other Delta Networks are built using recursive composition as illustrated in Figure.21. For a Delta network of size $N \times N$, the first stage consists of $\frac{N}{k}$ crossbars and 2 smaller Delta networks of size $\frac{N}{2}$. That is the inverse case for reverse networks. figure.22 shows the recursive nature of a Butterfly (8,2) network. Butterfly (8,2) means a Butterfly network of size 8 and degree 2. Here, the first stage of a Butterfly (8,2) network contains $\frac{N}{2} = 4$ switching elements followed by two smaller Butterfly networks of size 4×4.

A Reverse Butterfly (16,2) network is shown in the fig.23. It uses the same permutations as the Butterfly network, however in a reverse order such that the connection patterns can be described as:

$$C_0, \beta_0^k; C_i(1 \leq i \leq n-1), \beta_i^k; C_n, \beta_0^k \tag{14}$$

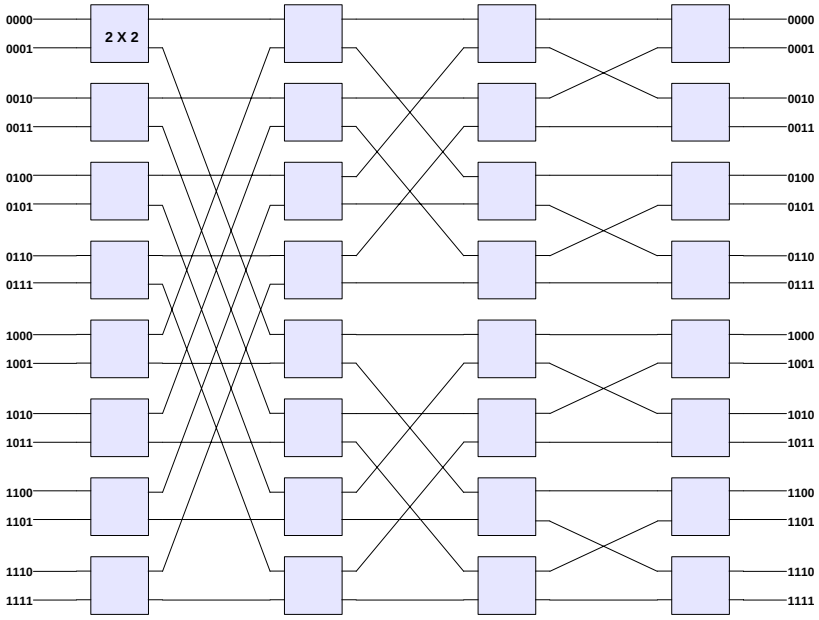

Figure 20: *A Butterfly (16,2) Network*

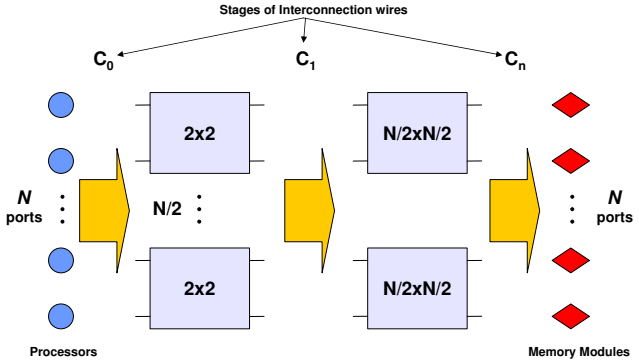

Figure 21: *Recursive composition of Delta networks*

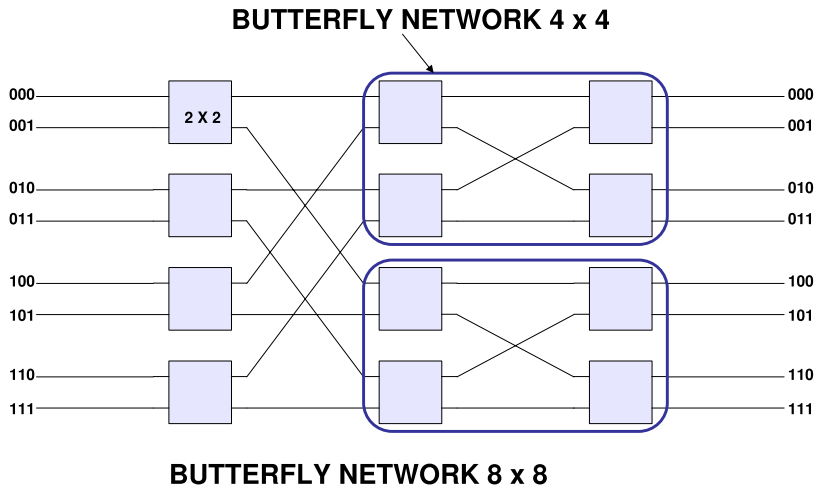

Figure 22: *A Butterfly (8,2) network*

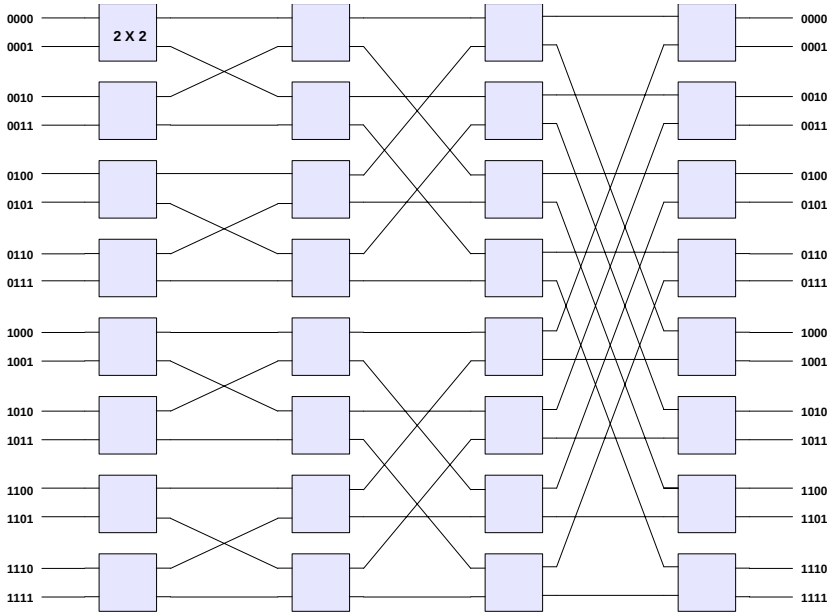

Figure 23: *A Reverse Butterfly (16,2) Network*

3.7.3 Baseline Network

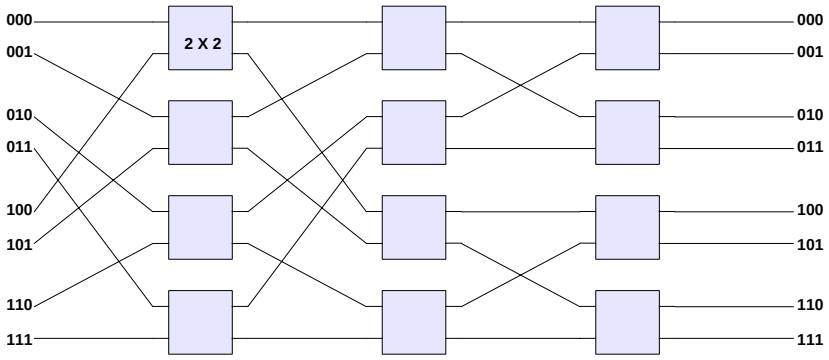

Figure 24: *A Baseline (8,2) Network*

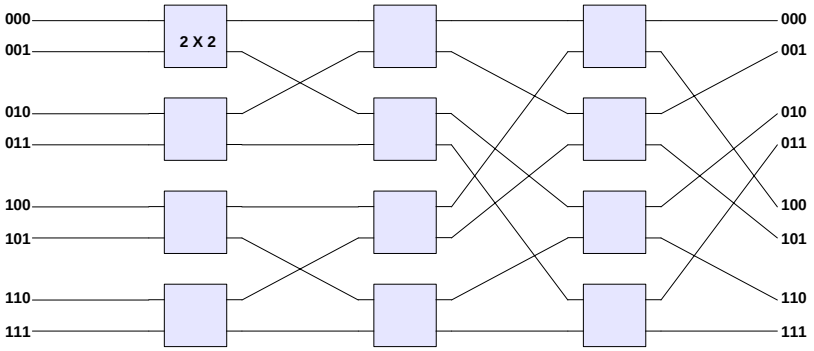

Figure 25: *A Reverse Baseline (8,2) Network*

In a Baseline network [42], the ith k–ary *baseline permutation* δ_i^k, for $0 \leq i \leq n-1$, is defined by

$$\delta_i^k\ (x_{n-1}\ x_{n-2} \ldots\ x_{i+1}\ x_i\ x_{i-1} \ldots\ x_1\ x_0) = x_{n-1}\ x_{n-2} \ldots\ x_{i+1}\ x_0\ x_i\ x_{i-1} \ldots\ x_1 \tag{15}$$

The ith baseline permutation performs a cyclic shifting of the i+1 least significant digits in the index to the right for one position. It should be observed that δ_0^k also defines the *identity permutation* I. The patterns for the baseline network can be summed up as

$$C_0, \sigma^k; C_i(1 \leq i \leq n), \delta_{n-i}^k \tag{16}$$

Thus the initial pattern from the sources to the first switching stage is an omega permutation and the rest are according to the baseline permutation. A Baseline network is also composed recursively like a butterfly network. Fig.24 shows a Baseline (8,2) network.

For a Reverse Baseline network, the reverse baseline permutation for $0 \leq i \leq n-1$, is defined by

$$\delta_i^{k^{-1}} (x_{n-1}\ x_{n-2} \ldots\ x_{i+1}\ x_i\ x_{i-1} \ldots\ x_1\ x_0) = x_{n-1}\ x_{n-2} \ldots\ x_{i+1}\ x_{i-1} \ldots\ x_1\ x_0\ x_i \tag{17}$$

Thus the patterns for the reverse baseline network can be summed up as

$$C_i(0 \leq i \leq n-1), \delta_i^{k^{-1}}; C_n, \sigma^{k^{-1}} \tag{18}$$

Thus the final pattern from the last switching stage to the destinations is a flip permutation. The Figure of a Reverse Baseline (8,2) network is shown in the fig.25.

3.7.4 (Generalized) Cube Network

In a Generalized Cube Network [34], [33], the *i*th *cube permutation* E_i , for $0 \leq i \leq n-1$; is defined only for $k = 2$ by

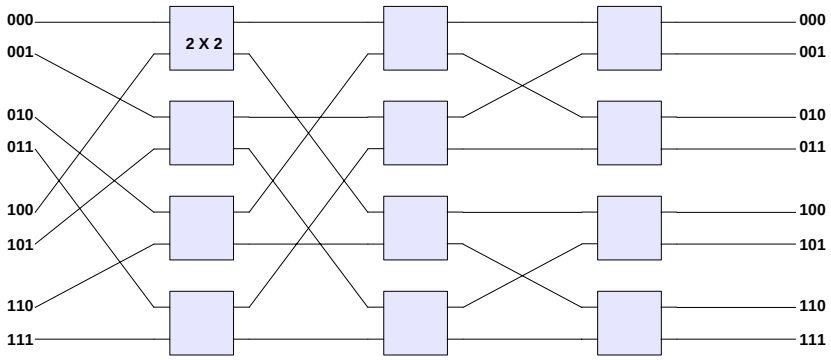

Figure 26: *A Generalized Cube (8,2) Network*

$$E_i(x_{n-1}\ x_{n-2} \ldots\ x_{i+1}\ x_i\ x_{i-1} \ldots\ x_1\ x_0) = x_{n-1}\ x_{n-2} \ldots\ x_{i+1}\ \overline{x_i}\ x_{i-1} \ldots\ x_1\ x_0 \tag{19}$$

The *i*th cube permutation complements the *i*th bit of the index. The permutation E_0 is also called exchange permutation.

For a cube MIN (or Multistage cube network [33]), connection pattern C_i is described by the n - *i*th butterfly permutation β_{n-i}^k for $1 \leq i \leq n$. Connection pattern C_0 is selected to be σ^k. Therefore the connection patterns can be summed up as

$$C_0, \sigma^k; C_i(1 \leq i \leq n), \beta_{n-i}^k \tag{20}$$

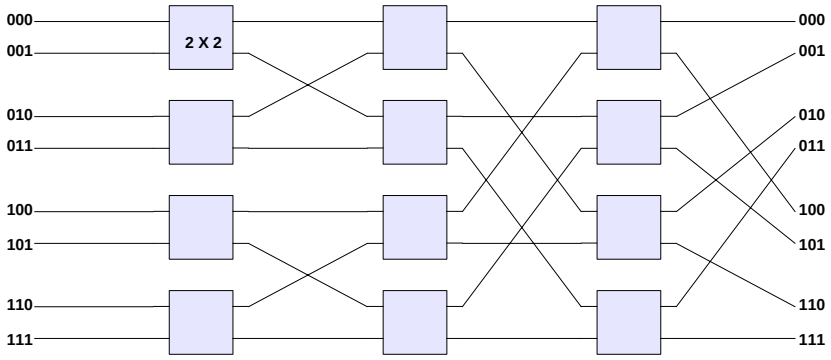

Figure 27: *An Indirect Binary n-Cube (8,2) Network*

For a Reverse Cube Network (Indirect Binary n-Cube [30]), the connection patterns can be defined as

$$C_i(0 \leq i \leq n-1), \beta_i^k; C_n, \sigma^{k^{-1}} \tag{21}$$

Both Cube network and its reverse are composed recursively just like a butterfly network. Figure.26 shows a Generalized Cube (8,2) network while figure.27 shows a reverse cube network which is an Indirect Binary n-Cube (8,2) network.

3.8 Equivalence of Delta Networks

The topological equivalence of these MINs can be viewed as follows: consider that each input link to the first stage is numbered using a string on n digits $s_{n-1}\ s_{n-2} \ldots\ s_1\ s_0$, where $1 \leq s_i \leq k-1$, for $0 \leq i \leq n-1$. The least significant digit s_0 gives the address of the input port at the corresponding switch, and the address of the switch is given by $s_{n-1}\ s_{n-2} \ldots\ s_1$.

At each stage, a given switch is able to connect any input port with any output port. This can be viewed as changing the value of the least significant digit of the address. For the connection to work, it should be possible to change the values of all the digits. As each switch only changes the value of the least significant digit of the address, connection patterns between stages are specified so that the position of digits is permuted; and after n stages, all the digits have occupied the least significant position. Thus the difference of the above mention types of delta networks is the order in which the digits occupy the least significant position.

3.9 Multilayer and Replicated MINs

In literature, there exist also Multilayer MINs and Replicated MINs. Replicated MINs enlarge MINs by replicating them L times. Multilayer MINs can be considered an enhanced extension where the

number of layers can be increased in each stage. A review of Multilayer MINs and Replicated MINs can be found in [40].

4 UML 2 Templates

Templates are used in many different application areas and can be summed up as a technique for generic programming. Constructs like C++ Templates [38] and Java Generics [14] offer templates for programming languages. Templates were first introduced in UML 1.3, and later deeply revisited in UML 2 Standard [18]. It should be made evident that the current UML 2 standard still lack precision regarding templates. The concept of nested classifier templates has not been addressed in UML 2 and the *"binding"* mechanism is very ambiguous as described in the specifications. While [15], [39] do use the component template concept to some extent, the notion of using the template parameter of a component template for multiplicity of its ports has not been undertaken before.

4.1 The UML 2 Template Notation

In the UML 2 Standard [18], a template is basically a model element which is parameterized by other model elements. These parameterizable elements can be *classifiers, packages* or *operations*. Classifier and package template elements are respectively called *Classifier Templates* and *Package Templates*. For the specification of the parameterization, a template element owns a *Template Signature* that relates to a list of formal *Template Parameters* where each parameter chooses an element that is part of the template. Template elements have also a specific standard notation which consists in superimposing a small dashed rectangle containing the formal template parameters along with their signatures on the right hand corner of the standard symbol.

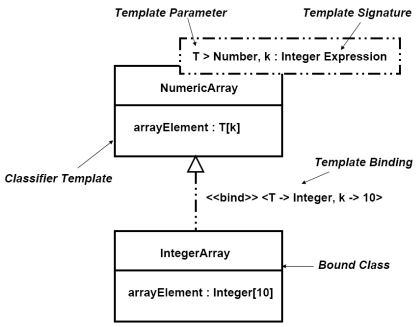

Figure 28: *Example of a Class Template*

Using the *Template Binding* relationship, a template can create other model elements. This binding relationship links a *"bound"* element to the signature of a target template causing a set of template parameter substitutions in which formal template parameters are replaced by actual parameters.

Simply, the binding of a bound element means recopying of the contents of a template element in the bound element with any element exposed as a formal parameter being replaced by an actual element specified during the binding.

Fig.28 shows a class template as described in the UML 2 Standard. This class *NumericArray* is graphically represented as a standard UML class with a dashed rectangle containing its signature. Here, the signature states two elements as formal parameters: *T* of type *Number* and *k* of type *Integer*. The class *IntegerArray* is bound to the *NumericArray* template through a *"bind"* relationship. This class is the result of the substitution in the template of its formal parameters, *T* and *k* by the actual values *Integer* and *10* respectively.

4.2 The UML 2 Template Metamodel

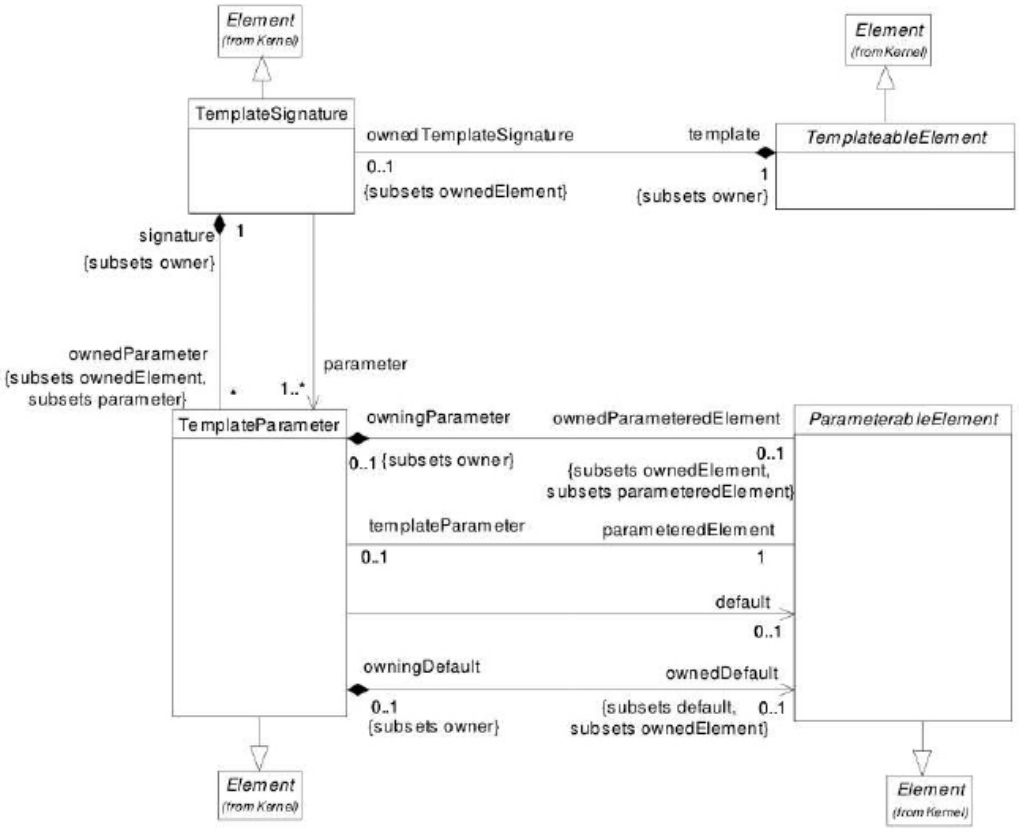

Figure 29: *Template Metamodel*

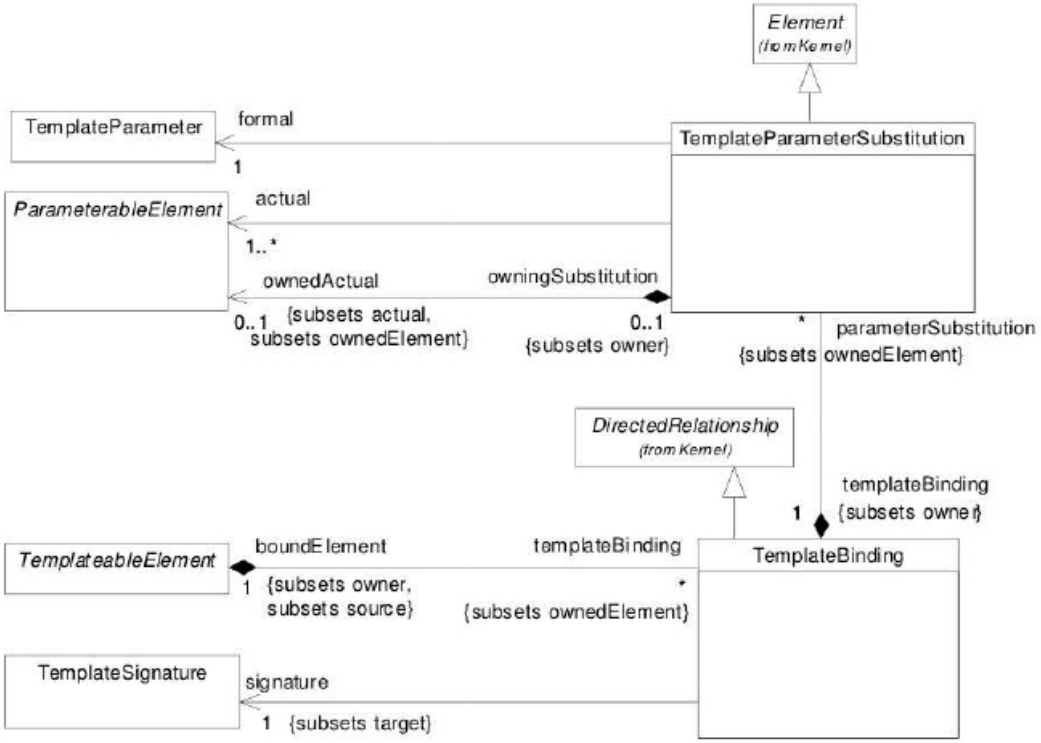

Figure 30: *Template Binding Metamodel*

The template package described in the UML 2 metamodel specifications [18] introduces four classes for its description: *TemplateSignature*, *TemplateableElement*, *it TemplateParameter* and *ParameterableElement* (Fig.29). The *TemplateBinding* and *TemplateParameterSubstitution* metaclasses are both used to bind templates to the bound elements (Fig.30).

UML 2 templates which are sub-classes of the abstract class TemplateableElement can be parameterized. *Classifier*, *Package* and *Operation* are templateable elements. The set of template parameters (TemplateParameter) of a template (TemplateableElement) are included in a signature (TemplateSignature) which corresponds to a small dashed rectangle superimposed on the symbol for the templateable element. A TemplateParameter corresponds to a formal template parameter and exposes an element owned by the template thanks to the ParameterableElement role. A template parameter is only meaningful in the specifications of a templateable element and not outside them. It should be observed that only parametrable elements (ParameterableElement) can be exposed as formal template parameters for a template or specified as actual parameters in case of a template binding. Such UML 2 elements are: *Classifier*, *PackageableElement*, *Operation*, *ConnectableElement*, *ValueSpecification* and *Property*. ConnectableElement is an abstract metaclass which represents a

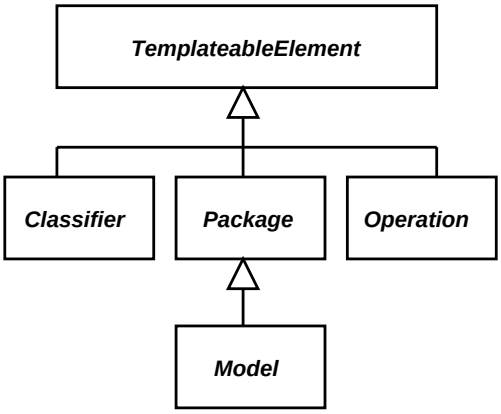

Figure 31: *TemplateableElement Metaclass*

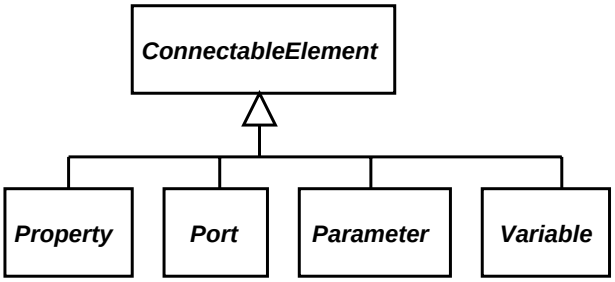

Figure 32: *ConnectableElement*

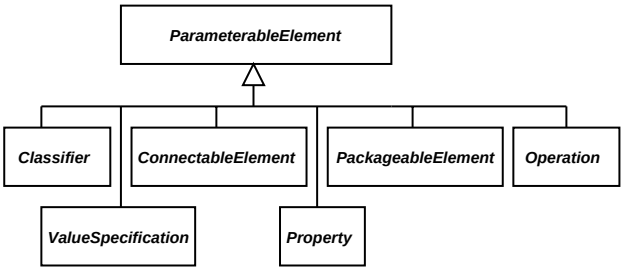

Figure 33: *ParameterableElement*

set of instances which play the role of a classifier. They are usually joined by attached connectors. ValueSpecification is an abstract metaclass and is used to define values in a model. It may reference an instance or may be an expression denoting an instance or instances when evaluated. Property is a structural feature, i.e. that it specifies the structure of instances of a classifier. It could represent an attribute for a classifier.

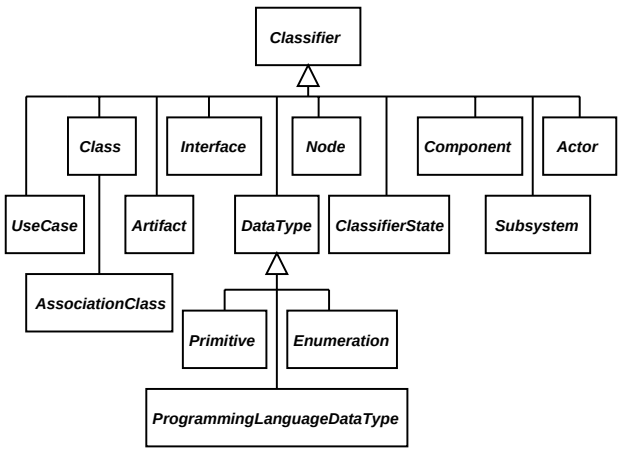

Figure 34: *Classifier Metaclass*

A template binding is best described as recopying the contents of a template element in the bound element while replacing the formal template parameters with actual parameters. It is a directed relationship labeled by the << *bind* >> stereotype from the bound element to the template. In the case where no actual parameter is specified in the binding for a formal parameter, the default element for that formal template parameter (if specified) is used. If the default value for a formal template parameter is not given, then the bounded element is also a templateable element.

A template cannot be used in the same manner as a non-template element of the same kind. A template element is used only to either create bound elements or as part of the specification of another template. A bound (non-template) element is an ordinary element and can be used in the same way as a non-bound (and non-template) element of the same kind. It can be used as the type of a typed element. In simplified terms, it means that a template cannot be instantiated and is an abstract form; but when a template is made concrete by means of the template binding relationship, the bound element (if not a template itself) can be instantiated. UML 2 also introduces the notion of partial binding. Since it is of no interest to us, we have not specified it.

4.3 Component Templates

So far in existing references, templates are only used to define class, collaboration or package templates. While the UML 2 specifications clearly state that all specializations of a classifier can be

used as template elements, there is a drastic lack of references relating to component templates. For the modeling of Multistage Interconnection Networks, we make use of Component Templates. We also associate the template parameter specified in the component template with ports of the template component to specify a multiplicity on ports of the component depending on the actual value of the template parameter when it is substituted in the template binding. Figure.35 shows the schematic of a Component Template.

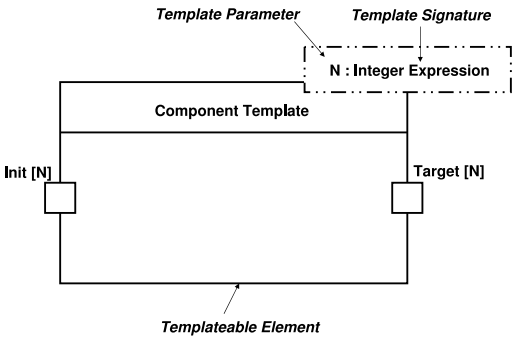

Figure 35: *Example of a Component Template*

Here is why this construction is possible: A port is defined as a *Structural Feature* of an *Encapsulated Classifier*. An encapsulated classifier extends a classifier and enables it to own ports as part of its specification. (It should be taken into account that only classes and components can contain the ports). We now focus only on components. Ports are structural features of a component (and help to define the specification of a component or its instances).

As a template parameter is used only in the namespace of a template element, it can be used anywhere in the specification of a template element. Thus it is possible to use the template parameter with ports of the component template resulting in the effect that when a template parameter is given an actual value for the parameter, the ports are given a specific multiplicity depending on the value of the actual parameter.

Thus by using component templates and template binding, we can create different component types with the necessary multiplicity on their ports. These component types can be instantiated in turn. We assume that we can create different component types from a component template having similar names after the template binding but with different values for the template parameter. The reason for this is described in the UML 2 standard. A bound element can have multiple bindings to the same template. The different bindings are evaluated in isolation and produce intermediate results, which are then merged to produce the final result.

4.4 Nested Templates

In the UML 2 specifications, nested templates are only defined expressively for packages. While a package template may contain another nested package template, nested classifier templates are

not precisely addressed in UML 2. While the concept of nested class(ifier) templates is present in programming languages like C++ [25], we find a lack of UML references in this context. We address this problem and propose a solution for using nested classifier templates. For this, we first recall anonymous bound classifiers as described in UML 2.

Figure.36 shows a class template *FArray*, with two template parameters ***T*** and ***k***. The second parameter ***k*** has a template signature and its default value is set to 10. The square box below the class template is an alternative to represent the template binding relationship. As it can be seen, the class template has not been renamed in the template binding and an anonymous bound class(ifier) is used which has the same name *FArray* as the class template. The template parameter ***T*** is substituted for the Point class. Since there is no substitution for ***k***, the default (10) will be used. This anonymous bound class is a type of a typed element and can be instantiated.

This methodology closely resembles the use of templates (or generics) in languages like C++. In C++, when a class template is specified, it is assigned a name for that template in the definition of the template; and a template is made concrete by using the same name as the template with an actual value for the template parameter along with its type declaration followed by the name of an instance. Thus a new class type (bound class) is created and the instance is of this class. The bound class is in fact an anonymous bound class as described in UML 2.

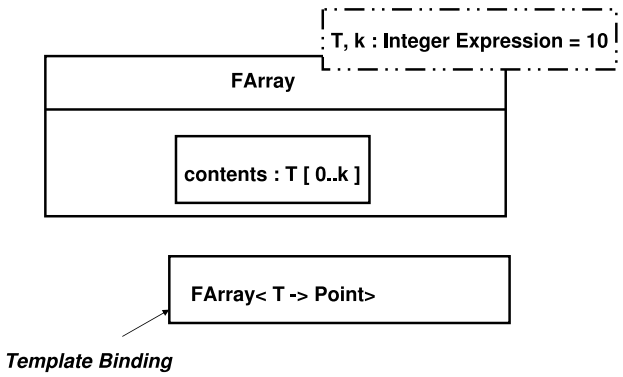

Figure 36: *Example of an Anonymous Bound Classifier*

We want to use this concept for the creation of nested component templates. However, since nested classifier templates are not precisely specified in the UML 2 standard, we propose a new semantic to address this problem.

In figure.37, we describe two template components A and B where A contains B as part of its specification. B is also shown separately to give a clear idea of the underlying semantics. The part b contained within A illustrates the new semantics. The notation $< N \rightarrow N >$ written with part b is to represent the template binding mechanism for nested templates where the formal template parameter N of B is substituted by the actual value of the template parameter N of A.

Using the template binding notation, a bound component having the same name as B is created with port multiplicities equal to the value of template parameter of A. It is evident that this bound

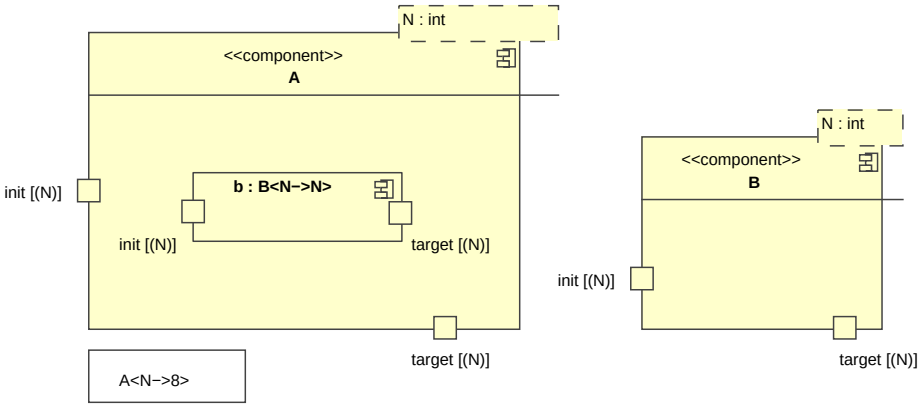

Figure 37: *Template Binding notation for Nested Templates*

component is an anonymous bound component and could have potential instances as evident from the part b. Using the template binding for A, a value of 8 is assigned to N which creates an anonymous bound component with the same name and concretize its internal structure. We also assume that a template can be used in the specifications of a non templateable element.

4.5 Nested Templates and Recursion

As described before, Delta networks are built using recursive composition. Hence for their modeling, we need to use generic components with recursive characteristics. For this reason, we specify the use of recursion using nested templates.

Figure.38 shows the schematics of nested recursive templates. As before, component template A contains the nested component template B as part of its specification. However for recursion to take place, template B itself contains template A as part of its own specification with the difference that in the binding relationship specified for A inside B, the actual value for the template parameter N of A is set equal to $\frac{N}{2}$ of B. The multiplicity on the ports is also changed due to the binding mechanism. Hence if N is given a value of 8 for example, due to binding, template A is concretized as an anonymous bound component and the part a acquires a multiplicity of $\frac{N}{2}$ on its ports. The difference between the anonymous bound component with template parameter N and the new anonymous bound component with template parameter $N = \frac{N}{2}$ is that they represent two different component types as explained earlier with different values of N; and each having their respective port multiplicities. These are intermediate results leading to a final result when the recursion is stopped.

It should be evident that this recursion is infinite in nature. We need a mechanism in order to avoid infinite recursion and to stop the recursion at a desired level. This is where OCL constraints [17] come into play. OCL (Object Constraint Language) is a language utilized for describing rules that apply to UML models. It provides object query expressions and constraints that cannot

be expressed by UML diagrams. By using the OCL constraints, we can control the recursion by manipulating the connectors present in our methodology. This approach has also been studied and used in the UML profile for SoC [15] for the creation of a generic shift register. This feature allows us to avoid the infinite recursion trap.

Consider the Fig.39 in which a shift register component *ShiftReg3* is shown with three register subcomponents. The input port *I* of the *ShiftReg3* component connects to the input port *I* of the first register *M0* while the output port *O* of *ShiftReg3* connects to the output port *O* of the last register *M2*. Connector *C0* and *C1* connect output ports of *M0* and *M1* to input ports of *M1* and *M2* respectively.

Fig.40 illustrates a generic *ShiftReg* template. By specifying an actual value for the template parameter *N*, we obtain *N* register sub components and *N-1* connectors which connect the output and input ports of these registers. A set of OCL constraints is also attached to the template which specify that the actual value of *N* must be greater than one. For a certain sequence of values of *N*, only those connectors exist which connect output ports of a register to input ports of another register. And for a different sequence of values of *N*, only those connectors exist which connect input ports of a register to output ports of another register. This is due to the fact that in UML for SoC, the ports are bidirectional in nature. Thus these OCL constraints help to specify existence of connectors depending on the values of *N*. By the same manner, the existence of a component part can be specified using OCL constraints.

In a latter subsection, we describe the OCL constraints utilized with the nested recursive templates example. While one template performs the required recursion, OCL constraints applied on the other template so that for a particular value of parameter *N* , the recursion stops by controlling the existence of certain connections.

We also apply Gaspard stereotypes to our templates. Examples of templates using stereotypes can be found in [15] and [36] .

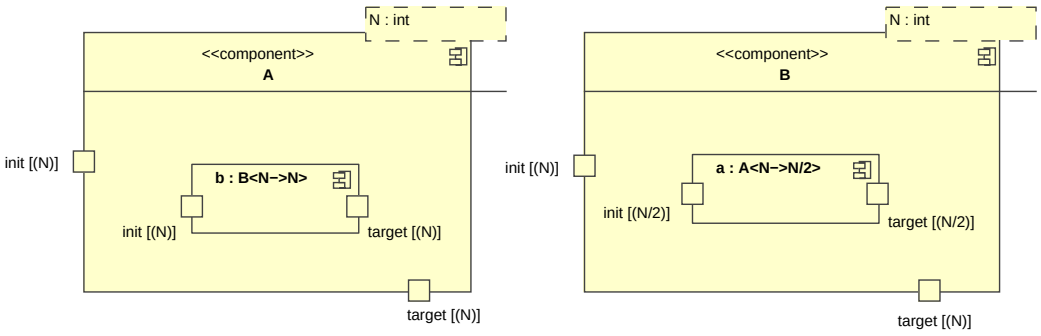

Figure 38: *Example of Nested Recursive Template components*

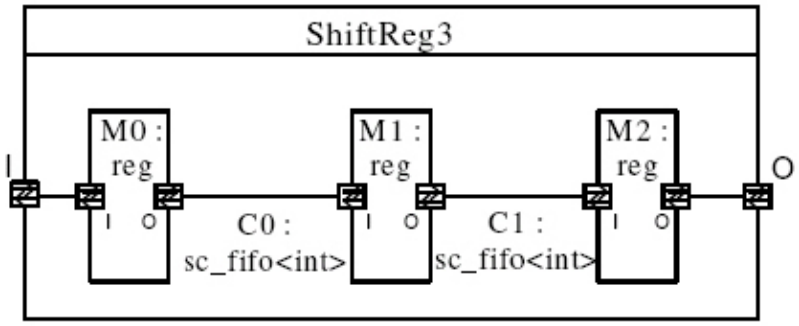

Figure 39: *A Shift register module in UML For SoC*

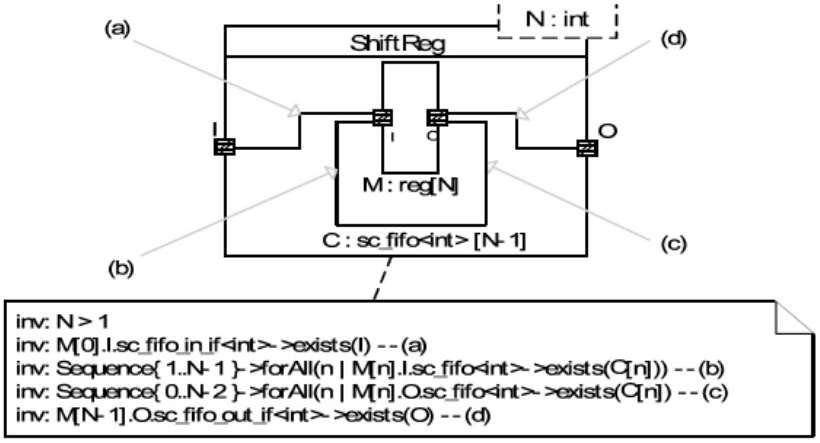

Figure 40: *A Shift register Template module in UML For SoC*

5 Modeling of Multistage Interconnection Networks

The second part of the report deals with the modeling of multistage interconnection networks while using the Gaspard 2 Profile for UML Real-Time and Embedded systems. As stated earlier, we focus mainly on the DSUB networks which are built using crossbars of size 2×2.

We thus describe certain types of delta networks described earlier in the report and present the modeling schematics of these networks.

5.1 Omega Networks

We first focus on the Omega networks as they are considered to be the most widely recognized type of Delta networks. An important aspect of Omega networks is that the interconnection links are the same between the stages of the switching elements except the interconnection links from the last switching stage to the destinations. The modeling of Omega network was first proposed in [12]. We expand that proposition with regards to the current Gaspard 2 UML Profile. We first analyze the perfect shuffle pattern of Omega networks while utilizing the mechanism of Array-OL as described earlier in the report. Fig.41 illustrates the modeling of a perfect shuffle connection pattern for size N which is utilized in Omega networks.

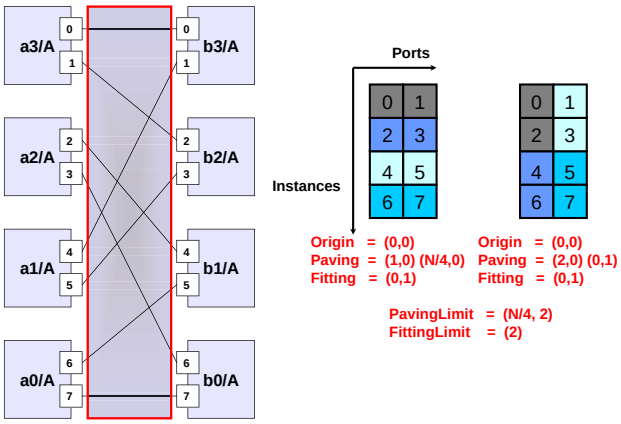

Figure 41: *Perfect Shuffle Pattern (Here N is taken as 8 in the example)*

We then use these values in the modeling of an Omega network. Fig.42 shows a typical Omega network of size 8×8.

As described before, A delta network of N inputs/outputs requires $\log_k(N)$ stages (where $k = 2$; the size of crossbars 2×2) and each stage has $\frac{N}{k}$ switches. For this Omega network, the number of stages is $\log_2(8) = 3$ stages, and each stage having $\frac{8}{2} = 4$ crossbars.

In order to model an Omega network, we break apart the network into several Gaspard Components. It should be noted that all components described now are template components except the Crossbar2 $\times$ 2 component. Also all components have the necessary *Communication* stereotype to indicate that the Omega network is basically a communication system.

The first component Crossbar2x2 represents a crossbar of size 2×2 as shown in figure.43. It consists of input port init(2) and output port target(2) where the multiplicities of the ports indicated the number of inputs and outputs for the Crossbar2x2.

Rising to the next higher level of hierarchy, we define the specifications of the template component Stage in figure.44. which represents one stage of the Omega Network. It has a template

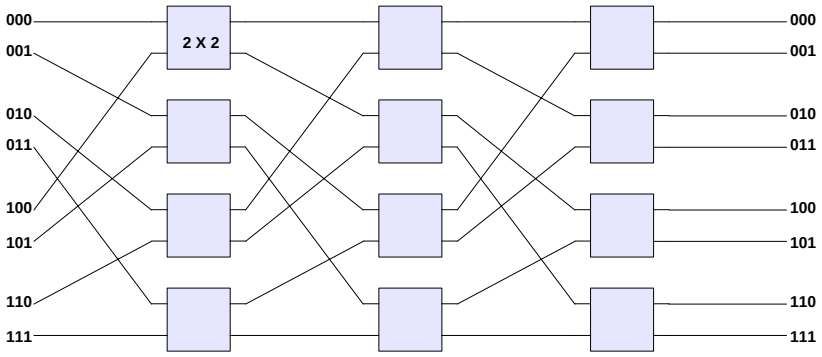

Figure 42: *An Omega (8,2) Network*

parameter N of integer type. It also has its respective init and target ports with multiplicity equal to N.

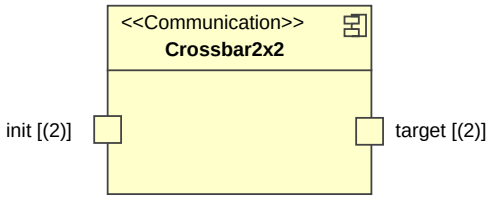

Figure 43: *Modeling of a* Crossbar2x2 *component*

The Stage component contains a part xbar of type Crossbar2x2 with multiplicity equal to $\frac{N}{2}$. This multiplicity represents the number of crossbars in one stage of an Omega network. Thus in the case when $N = 8$, the Stage component will contain 4 repetitions of a crossbar switch. The Stage component also contains two *Tiler* connectors, whose tagged values correspond to a straight connection from init(N) of the Stage component to the init(2) ports of the Crossbar2x2 and from target(2) ports of the Crossbar2x2 to the target(N) of the Stage component.

Rising to another hierarchical level, we define the template component Block in Fig.45 having a template parameter N of integer type with respective init and target ports. Block contains a part stg of type Stage. It should be noted that the template binding has been specified with the part stg with the notation $< N \rightarrow N >$ enabling it to have port multiplicities equal to as given to Block.

Block also contains two *Reshape* connectors. The first Reshape connector from init(N) of the Block component to init(N) of stg defines a perfect shuffle connection with its respective tagged values. The second Reshape connector from target(N) of stg to target(N) of Block component defines a direct connection or no permutation. It can also be described as an *identity permutation*.

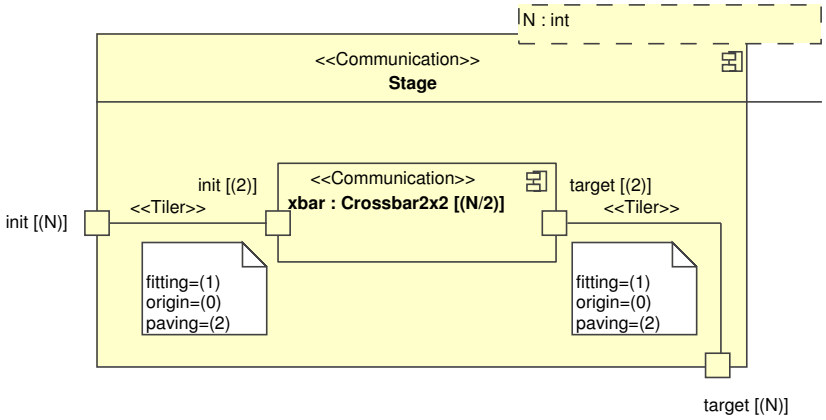

Figure 44: *Modeling of a* Stage *Template component*

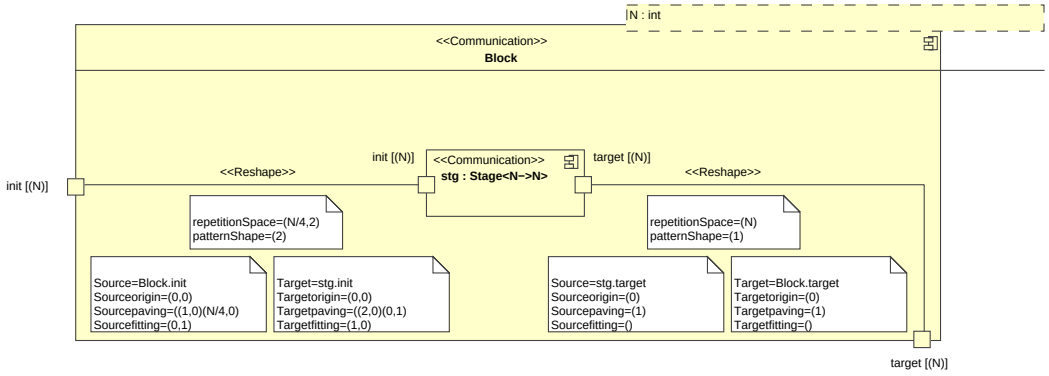

Figure 45: *Modeling of a* Block *Template component*

Mounting to the highest and final level of hierarchy, we describe the template component called OmegaNetwork in Fig.46. It contains part blck of the component type Block with multiplicity represented by the notation to $\log_2(N)$ with the binding notation $< N- > N >$ which corresponds to the repetition of part blck, $\log_2(N)$ times where N correponding to the value specified for OmegaNetwork. As with other template components, it also has its own template parameter N of integer type and its own init and target ports with multiplicities equal to N.

The connector with the *InterRepetitionLink* stereotype demonstrates the interconnection between the different repetitions of Block. This stereotype permits to specify that there exists a uniform dependence on the repetition space of blck , limited by the multiplicity. Also the tagged value

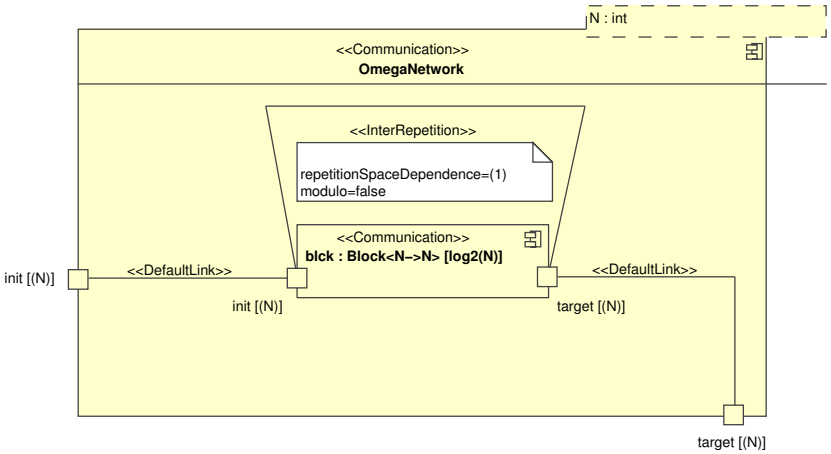

Figure 46: *Modeling of an* OmegaNetwork *Template component*

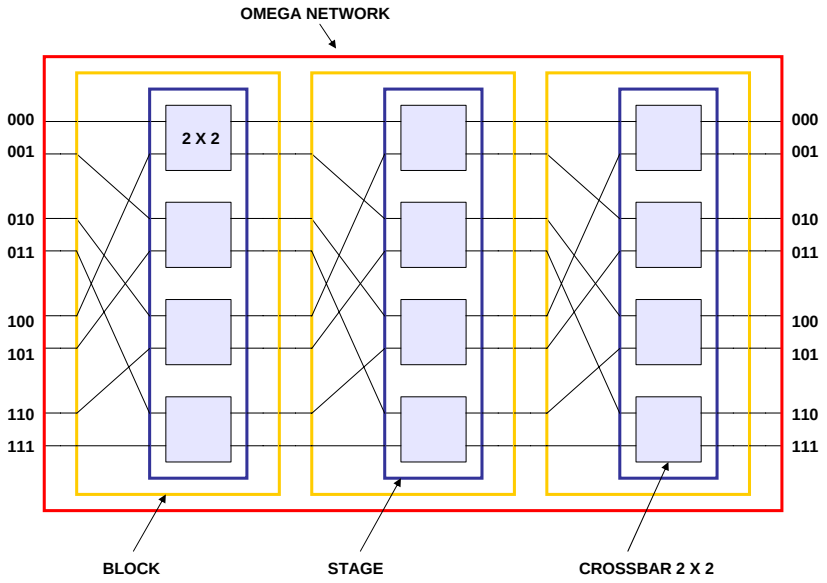

Figure 47: *Simplified overview of an Omega Network of size* 8×8

repetitionSpaceDependence is equal to (1). It should be noted that the tagged value *modulo* is equal to false, which signifies that the *target* ports of the last block are connected to nothing.

In order to completely specify the topology, two *DefaultLink* connectors are also utilized. The first is used to connect the init(N) ports of the OmegaNetwork to init(N) ports of the first repetition of the Block, while the latter is used to connect the target(N) ports of the last Block repetition to the target(N) ports of the OmegaNetwork.

In Summary, when the value of template parameter N of OmegaNetwork is given a value corresponding to the required size of the Omega network, The OmegaNetwork uses this value to generate the specifications of its internal structure. For a value, for e.g. $N = 8$, the template component OmegaNetwork takes this value and creates a concrete anonymous bound classifier with the same name and with multiplicity equal to 8 for its ports.

When OmegaNetwork is made concrete, its internal sub component structures are also made concrete. As it contains a nested template component Block with its own template parameter N, when an actual value for N is given, the nested template component Block is also concretized, but since a template cannot be instantiated, we use the *anonymous bound classifier* concept. Thus part blck represents a potential instance of the concretized form of the template component Block having a repetition $\log_2(N)$ as described by the template binding notation. So for $N = 8$, part blck has a repetition equal to 3, and a multiplicity equal to 8 for its ports.

We then take a look at the template component Block containing the template component Stage as part of its specifications. When an actual value for N is given, the nested template component Stage is also concretized, resulting in part stg with multiplicity equal to N for its ports as specified by the template binding notation $< N \rightarrow N >$. So for $N = 8$, part stg has a multiplicity equal to 8 for its ports.

Finally, we analyze the template component Stage which contains a part xbar of type Crossbar2x2 as its specification. When the template parameter N is assigned an actual value, i.e. $N = 8$; Stage is concretized and we obtain repetitions of xbar equal to $\frac{N}{2}$, i.e. $\frac{8}{2}$, resulting in 4 repetitions of crossbar 2×2 in a concretized Stage component.

Fig.47 shows a simplified overall view of the above mentioned explanation for an Omega network of size 8×8.

5.2 Butterfly Networks

We now focus on the modeling of Butterfly networks that form part of the Delta networks family. An important aspect of Butterfly networks is that the interconnection links are not the same between the stages of the switching elements except the connection pattern from the sources to the first stage and the connection pattern from the last stage to the destinations.

We first analyze the butterfly permutation pattern of Butterfly networks while utilizing the mechanism of Array-OL. Fig.48 represents the modeling of a butterfly connection pattern for size N.

In order to model a Butterfly network, we break the network into several Gaspard components. We make use of nested template components for this purpose. All components described here are template components except the Crossbar2x2 component. A bottom to top approach is adapted to explain the modeling. Also all components have a *communication* stereotype described in the Gaspard profile.

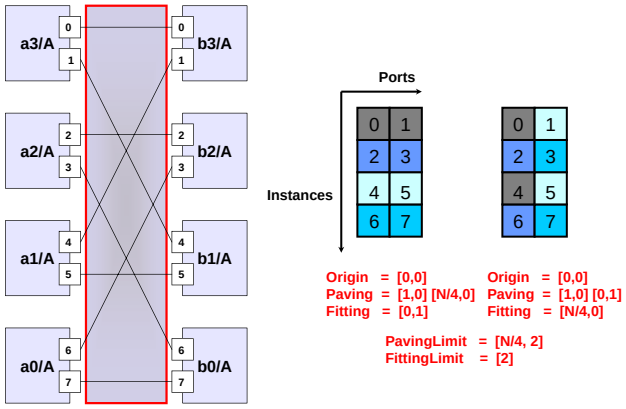

Figure 48: *Perfect Shuffle Pattern (Here N is taken as 8 in the example)*

The first component Crossbar2x2 represents a crossbar of size 2×2 as shown in figure.49. It consists of input port init(2) and output port target(2) where the multiplicities of the ports indicated the number of inputs and outputs for the Crossbar2x2.

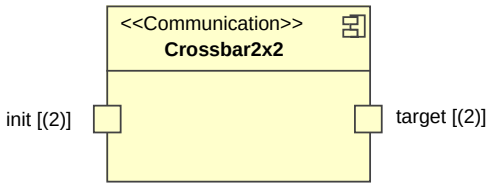

Figure 49: *Modeling of a* Crossbar2x2 *component*

Escalating to the next higher level of hierarchy, we define the specifications of a template component FirstStage in figure.50 which represents the first stage of a Butterfly network. It has a template parameter N of integer type. It also has its respective init and target ports with multiplicity equal to N.

The FirstStage component contains a part xbar of type Crossbar2x2 with multiplicity equal to $\frac{N}{2}$. This multiplicity represents the number of crossbars in the first stage of a Butterfly network. Thus in the case when $N = 8$, the FirstStage component will contain 4 repetitions of a crossbar switch. The FirstStage component also contains two *Tiler* connectors, whose tagged values correspond to a straight connection from init(N) of the FirstStage component to the init(2) ports of the Crossbar2x2 and from target(2) ports of the Crossbar2x2 to the target (N) of the FirstStage component.

Rising to the next hierarchical level as illustrated in figure.51, we define a template component NextStage having a template parameter N of integer type with respective init and target ports.

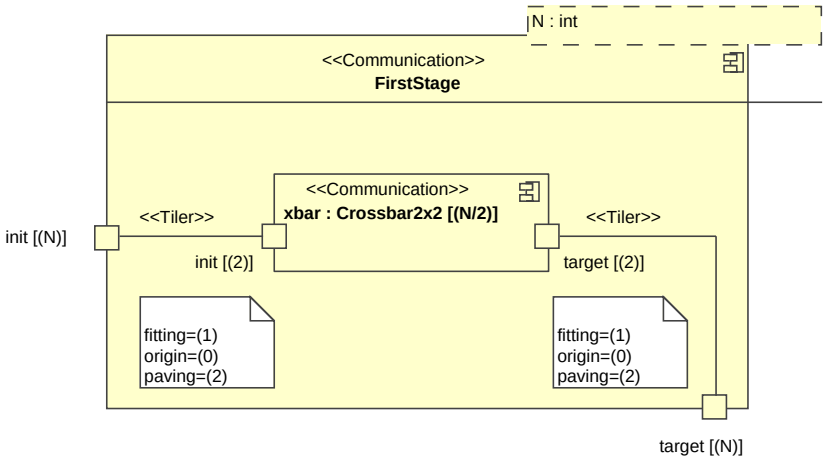

Figure 50: *Modeling of a* FirstStage *Template Component*

NextStage contains a part Recursive of type ButterflyBlock with multiplicity equal to 2. ButterflyBlock is described at a higher level of hierarchy. It should be taken into account that template binding has been specified with the part Recursive with the notation $< N \rightarrow \frac{N}{2} >$ enabling it to have port multiplicities equal to one half as given to NextStage.

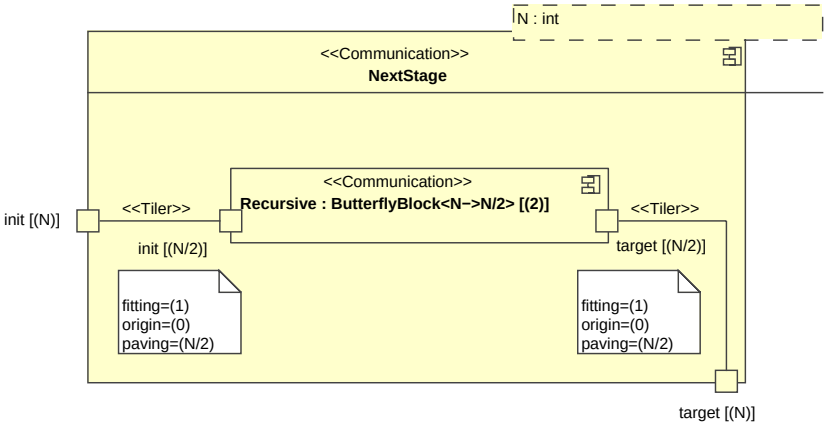

Figure 51: *Modeling of a* NextStage *Template Component*

The NextStage component also contains two *Tiler* connectors, whose tagged values correspond to a straight connection from the init(N) and target(N) ports of the NextStage to the init(N) and target(N) ports of one repetition of the ButterflyBlock component.

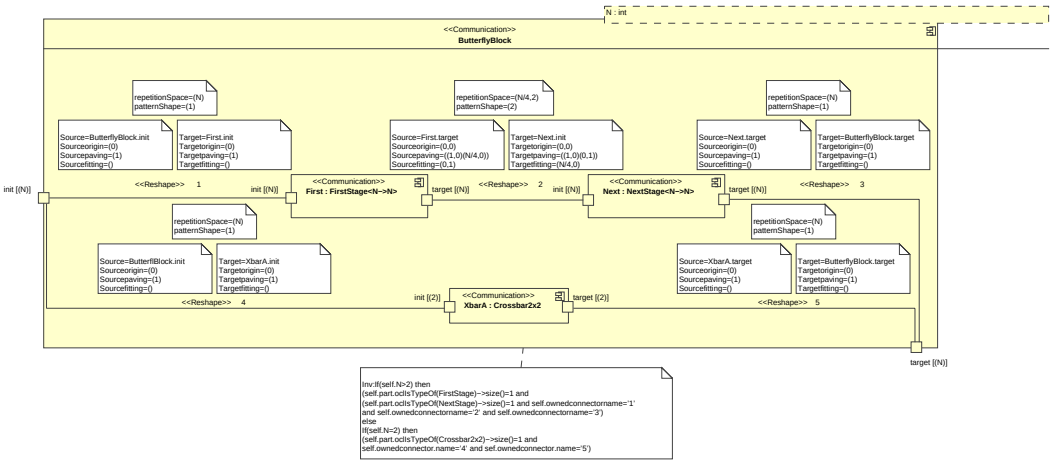

Figure 52: *Modeling of a* ButterflyBlock *Template Component*

Ascending to another hierarchical level in figure.52, we now define the template component ButterflyBlock containing a template parameter *N* of integer type and respective init and target ports. ButterflyBlock contains part First of the component type FirstStage, part Next of the component type NextStage and part XbarA of the component type Crossbar2x2.

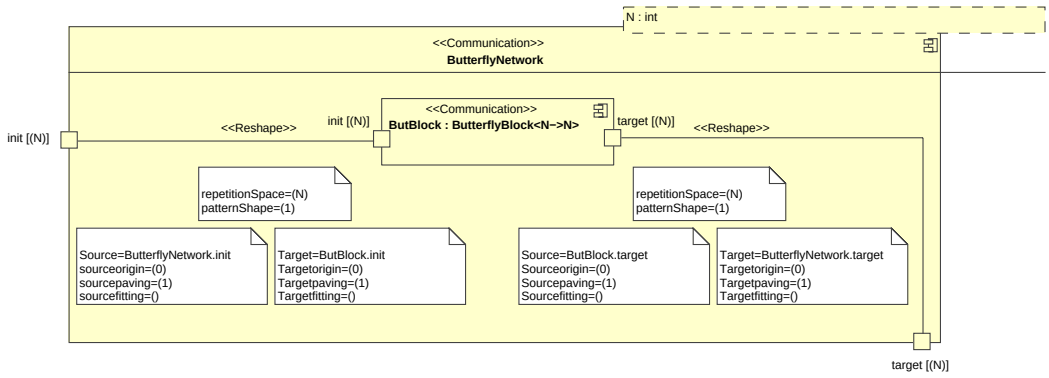

Figure 53: *Modeling of a* ButterflyNetwork *Template Component*

Both parts First and Next have the template binding notation assigned to them. For both, the notation is $< N \rightarrow N >$. In effect this means that when an actual value for formal template parameter N of ButterflyBlock is specified, the parts acquire the same value on their ports.

For ButterflyBlock, we also define five *Reshape* connectors and name them respectively. The naming is carried out to remove ambiguity when OCL constraints are applied. The first reshape connector (*Reshape 1*) is connected from init(N) ports of ButterflyBlock to the init(N) ports of part First, and defines a direct connection or no permutation. The second reshape connector (*Reshape 2*) is connected from the target(N) ports of part First to the init(N) ports of part Next. This reshape connector defines the Butterfly interconnection pattern for a value corresponding to N. The third reshape connector (*Reshape 3)* is connected from target(N) ports of part Next to the target(N) ports of the ButterflyBlock, and defines a direct connection or no permutation. The fourth reshape connector (*Reshape 4*) is connected from the init(N) ports of the ButterflyBlock to the init(2) ports of the part XbarA, and defines a direct connection. The fifth reshape connector (*Reshape 5*) is connected from the target(2) ports of the part XbarA to the target(N) ports of the ButterflyBlock, and defines a direct connection.

We also specify a set of OCL constraints for ButterflyBlock, which basically help to stop the recursion, as described earlier in the paper. The OCL constraints state that the actual value of the template parameter N should always be greater than or equal to 2. The Reshape connectors *1*, *2* and *3* along with parts First and Next, exist only for a value of $N \geq 2$. When the value of $N = 2$, only the Reshape connectors *4*, *5* and part XbarA exist resulting in the halting of the recursive process.

Finally in figure.53, we define the template component ButterflyNetwork having a template parameter N of integer type and its own respective init and target ports with multiplicities equal to N. It contains a part ButBlock of the component type ButterflyBlock having the binding notation set as $< N \rightarrow N >$. Thus when an actual value for N is specified for ButterflyNetwork, the part Butblock acquires this value on its ports as their multiplicity.

In order to completely specify the topology, ButterflyNetwork uses two reshape connectors. The first reshape connector connects the init(N) ports of the ButterflyNetwork to init(N) ports of part ButBlock. This reshape connector specifies the first interconnection pattern from the sources to the first stage of a Butterfly network. The second reshape connector connects the target(N) ports of part ButBlock to the target(N) ports of the ButterflyNetwork. This reshape connector specifies the last interconnection pattern from the last stage of a Butterfly network to the destinations.

In Summary, when the value of template parameter N of ButterflyNetwork is given a value corresponding to the desired size of the Butterfly network, it uses this value to generate the specifications of its internal structure. For a value, for e.g. $N = 8$, the template component ButterflyNetwork creates a concrete anonymous bound component with the same name and with multiplicity equal to 8 for its ports.

When the ButterflyNetwork is made concrete, its internal sub-component structures are also made concrete. As it contains a nested template component ButterflyBlock with a template parameter N as part of its specification; when an actual value for N of ButterflyNetwork is given, it replaces the formal template parameter of ButterflyBlock by the template binding notation and the nested template component ButterflyBlock is also concretized, but since a template cannot be instantiated, we use the *Anonymous Bound Classifier* concept. Thus part ButBlock represents the potential in-

stance of the concretized form of the template component ButterflyBlock and its port multiplicities are determined by the template binding notation with $< N \rightarrow N >$. So for $N = 8$, part ButBlock now has a multiplicity equal to 8 for its ports.

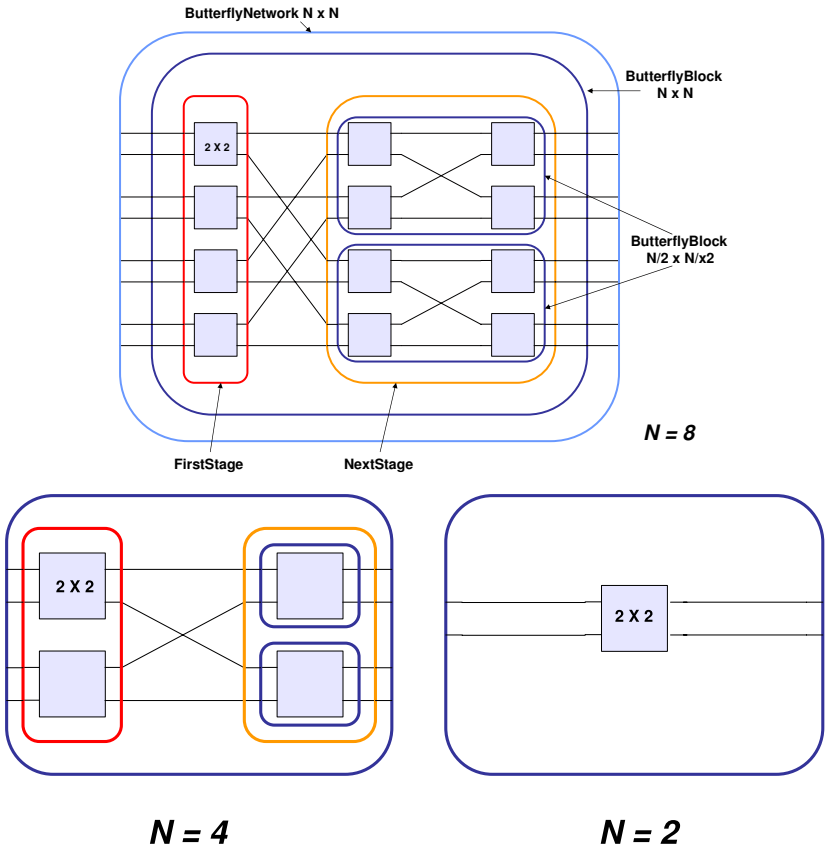

Figure 54: *Simplified Butterfly (8,2) network*

We then take a look at the template component ButterflyBlock containing two nested template components FirstStage and NextStage as well as part XbarA as part of its specification. When ButterflyBlock is concretized, its internal structure is also concretized, resulting in the concretization of the nested templates FirstStage and NextStage with the same mechanism as described above. Thus parts First and Next are potential instances of the concretized form of template components FirstStage and NextStage respectively and their port multiplicities have been substituted by an actual value, i.e. 8, during the template binding mechanism with $< N \rightarrow N >$. Since value of N is greater than or equal to 2, only *Reshape 1*, *2*, *3* along with parts First and Next are active respecting the OCL constraints.

We then analyze the template component FirstStage which contains a part xbar of type Crossbar2x2 as its specification. When the template parameter N is assigned an actual value, i.e. $N = 8$; FirstStage is concretized and we obtain repetitions of xbar equal to $\frac{N}{2}$, i.e. $\frac{8}{2}$, resulting in 4 repetitions of crossbar 2×2 in a concretized FirstStage component.

We create a finite recursion with the NextStage component. It contains a template component ButterflyBlock as part of its specifications as evident from part Recursive with multiplicity equal to 2. When template parameter N is assigned an actual value, i.e. $N = 8$; NextStage is concretized along with its internal structure and we obtain 2 repetitions of Recursive with port multiplicities equal to $N = \frac{N}{2}$ (i.e. for $N = 8$, $N \Rightarrow \frac{8}{2} = 4$) as defined by the template binding mechanism. Each repetition of the Recursive is the result of the concretization of the template component ButterflyBlock. Thus a recursion is created and we arrive again at the level of the template component ButterflyBlock but with a difference that the actual value of template parameter N of ButterflyBlock is now $\frac{N}{2}$, i.e. 4 as defined in the specification of NextStage. As before, only *Reshape 1*, *2*, *3* along with parts First and Next are active respecting the OCL constraints. The template parameter N of FirstStage binds with a value, i.e. 4 and produces 2 repetitions of xbar (corresponding to $\frac{N}{2}$). For NextStage, recursion again takes place and 2 repetitions of Recursive are produced with port multiplicities equal to $N = \frac{N}{2}$ (i.e. $N \Rightarrow \frac{4}{2} = 2$) as defined by the template binding mechanism. This is the final recursion and we arrive again at the level of the template component ButterflyBlock with actual value of template parameter N now equal to 2.

As per the OCL constraints, only Reshape *4*, *5*, and part XbarA are active respecting the OCL constraints which leads to the recursion being stopped by means of the OCL constraints. Figure.54 shows a simplified overall view of the above mentioned explanation for a Butterfly (8,2) network.

6 Conclusion

In this report, we have recalled the basic characteristics of Multistage Interconnection Networks and then have proposed a methodology to easily model Delta Networks. We have found a lack of references concerning component templates in relation with the existing UML 2 Standard. We thus have explored this concept and proposed the notion of nested component templates and the template binding notation for nested templates. A recursive approach is also used to facilitate the generation of certain type of Delta Networks. Now, all popular DSUB Networks constructed from crossbars of size 2×2 can be modeled using the given methodology. We can also extend this modeling methodology to include DSUB networks constructed from crossbars of different sizes based of the function of power of 2.

References

[1] A.C. Aljundi. *Une Méthodologie Multi-Critères Pour l'Évaluation de Performance Appliquée aux Architectures de Réseaux d'Interconnexion Multi-Étages*. PhD thesis, ustl, lifl, jul 2004. in-french.

[2] ARM. AMBA specifications. Technical report, ARM Limited, 1999.

[3] L.N. Bhuyan, R.R. Iyer, T. Askar, A.K. Nanda, and M. Kumar. Performance of Multistage Bus Network for a Distributed Shared Memory Multiprocessor. *IEEE Transactions on Parallel and Distributed Systems*, 8(1):82–95, Jan. 1997.

[4] OMG Architecture Board. Model Driven Architecture (MDA). Technical Report ormsc/2001-07-01, OMG, 2001.

[5] A. Borodin and J.E. Hopcroft. Routing, merging, and sorting on parallel models of computation. In *14th Annual ACM Symp. on Theory of Computing*, pages 338–344, 1982.

[6] P. Boulet, C. Dumoulin, and A. Honore. *From MDD concepts to experiments and illustrations*, chapter Model Driven Engineering for System-on-Chip Design. pub-iste, sept 2006.

[7] P. Budnick and D.J. Kuck. The organization and use of parallel memories. *IEEE Trans. Comput.*, C-20:1566–1569, Dec. 1971.

[8] J.R. Burke. Performance analysis of single stage interconnection networks. *IEEE Trans. on Comp.*, 40(3):357–365, Mar. 1991.

[9] C. Clos. A study of non-blocking switching networks. *Bell system tech. journal*, 32(2):406–424, Mar. 1953.

[10] M. Collier. A systematic analysis of equivalence in Multistage Networks. *Lightwave Technology*, 20(9), sep 2002.

[11] B. Cordan. An efficient bus architecture for system-on-chip design. *IEEE Custom Integrated Circuits Conference*, pages 623–626, May 1999.

[12] A. Cuccuru. *Modélisation unifiée des aspects répétitifs dans la conception conjointe logicielle/matérielle des systèmes sur puce à hautes performances*. PhD thesis, ustl, lifl, nov 2005. in-french.

[13] D. E. Culler, J. P. Singh, and A. Gupta. *Parallel Computer Architecture (A Hardware/Software Approch)*, chapter Interconnection Network Design. Morgan Kaufmann Publishers, 1999.

[14] M. Torgersen et al. Adding wildcards to the java programming language. In *2004 ACM symposium on Applied computing*, pages 1289–1296. ACM Press, sep 2004.

[15] A UML Profile for SOC Specification, April 2006.

[16] J. Gittoes, R. Barker, D. Lacan, T. Sinou, and C.C. Kion. *RS/6000 SMP Enterprise Servers Architecture and Implementation*. IBM Corporation, 2 edition, Dec. 1996.

[17] Object Management Group. OCL 2.0 Specification, Version 2.0. http://www.omg.org/docs/ptc/05-06-06.pdf, jun 2005.

[18] Object Management Group, editor. *UML 2 Superstructure (Available Specification)*. `http://www.omg.org/cgi-bin/doc?formal/05-07-04`, aug 2005.

[19] K. Hwang and F.A. Briggs. *Computer Architecture and Parallel Processing, 5th printing*. McGraw-Hill series in computer organization and architecture. McGraw-Hill International Editions, 1989.

[20] IBM. 32-bit processor local bus architecture specification, version 2.9. Technical report, IBM Corporation, 2001.

[21] A. Jantsch and H. Tenhunen. *Networks on Chip*. Kluwer Academic Publishers, 2003.

[22] M. T. Kechadi. *Un Modele de Fonctinnement Desordonne Pour les Systemes Multiprocesseurs Pipelines Vectoriels a Mémoire partagees (Definition, Modelisation et Proposition d'Architecture)*. PhD thesis, Universite des Sciences et Technologie de Lille, Laboratoire d'Informatique Fondamental de Lille, Mar. 1993.

[23] O. Labbani, J.L. Dekeyser, P. Boulet, and E. Rutten. UML 2 Profile for Modeling Controlled Data Parallel Applications. In *FDL'06: Forum on Specification and Design Languages*, Darmstadt, Germany, sep 2006.

[24] D. A. Lawrie. Access and alignment of data in an array processor. *IEEE Trans. Comput.*, C-24(12):1145–1155, Dec. 1975.

[25] MSDN. Visual C++ Language Reference:Nested Class Templates. `http://msdn2.microsoft.com/en-us/library/71dw8xzh.aspx`, 2006.

[26] S. Nussbaum and J.E. Smith. Statistical simulation of symmetric multiprocessor systems. In *35th Annual Simulation Symposium*, Apr. 2002.

[27] A. Pant. Ongoing analysis of NT SMP Clustering Products and Technologies. Technical report, National Center ofSupercomputing Applications (NCSA), Aug. 1997. http://archive.ncsa.uiuc.edu/People/apant/NTCluster/smp.html.

[28] J.H. Patel. Processor-memory Interconnections for Multiprocessors. In *Proc. 6th Annu. Symp. on Comput. Arch. Newyork*, pages 168–177, 1979.

[29] J.H. Patel. Performance of processor-memory interconnections for Multiprocessors. *IEEE. Trans. Comput.*, C-30(10):771–780, oct 1981.

[30] M.C. Pease. The indirect binary n-cube microprocessor array. *IEEE Trans. Comput.*, C-26:458–473, 1977.

[31] I. D. Scherson and A. S. Youssef. *Interconnection Networks for High-Performance Parallel Computers*. IEEE computer society press, 1994.

[32] H.J. Siegel. Analysis Techniques for SIMD Machine Interconnection Networks and the Effects of Processor Address Masks. *IEEE Trans. Comput.*, C-26(2):153–161, Feb. 1977.

[33] H.J. Siegel. Using the multistage cube network topology in parallel computers. 77(12):1932–1953, 1989.

[34] H.J. Siegel. *Interconnection Networks for Large scale Parallel Processing: Theory and case studies*. McGraw-Hill, 1990.

[35] P. G. Sobalvarro. Probabelistic Analysis of Multistage Interconnection Network Performance. Master's thesis, Electrical Engineering and Computer Science, MIT, Apr. 1992.

[36] D. Stein, S. Hanenberg, and R. Unland. Designing aspect oriented crosscutting in uml. *AOSD-UML Workshop at AOSD'02*, 2002.

[37] H.S. Stone. *High-performance computer architecture*. Addison-Wesley publishing company, 1987.

[38] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley Publishing Company, 1991.

[39] M. Tichy, B. Becker, and H. Giese. Component templates for dependable realtime systems. *In 2nd International Workshop on MDA With UML and Rule-based Object Manipulation*, sep 2004.

[40] D. Tutsch and M. Brenner. MINSimulate - A Multistage Interconnection Network Simulator. *17th European Simulation Multi-Conference : Foundations for Successful Modeling and Simulation(ESM'03)*, pages 211–216, 2003.

[41] France WEST Team LIFL, Lille. Graphical Array Specification for Parallel and Distributed Computing (GASPARD-2). http://www.lifl.fr/west/gaspard/, 2005.

[42] D. S. Wise. Compact layout of banyan/fft networks. In *CMU Conference, VLSI Systems and Computations*, pages 86–195, 1981.

Unité de recherche INRIA Futurs
Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399