

Introducing Control in the Gaspard2 Data-Parallel Metamodel: Synchronous Approach

Ouassila Labbani, Jean-Luc Dekeyser, Pierre Boulet and Éric Rutten

Laboratoire d'Informatique Fondamentale de Lille,
59655 Villeneuve d'Ascq Cedex, France
Web: <http://www.lifl.fr/west/>
E-mail: `Firstname.Lastname@lifl.fr`

Abstract. In this paper, we study the introduction of control into the Gaspard2 application UML metamodel by using the principles of synchronous reactive systems. This allows to take the change of running mode into account in the case of data parallel applications, and to study more general ways of mixing control and data parallel processing. Our study is applied to a particular context using two different models, exclusively dedicated to the process of computation or control. The computation part represents the Gaspard2 application metamodels based on the Array-OL language which is often used to specify the data dependencies and the potential parallelism in intensive applications treating multidimensional data. The control part is represented by an automaton structure based on the mode-automata concept which makes it possible to clearly identify the different modes of a task and the switching conditions between modes.

The proposed UML metamodel makes it possible to describe the control automata, the different running modes and the link between control and computation parts. It also allows to clearly separate the control and data parts, and to respect the concurrency, the parallelism, the determinism and the compositionality of the Gaspard2 models.

1 Introduction and Motivation

Computation intensive multidimensional data applications are more and more present in several application domains such as image and video processing or detection systems (radar, sonar, ...). The main characteristics of these applications is that they operate in real time conditions and are generally complex and critical. They are also multidimensional since they manipulate multidimensional data structured into arrays.

To study intensive signal processing applications, some computation models have been proposed to model and implement these systems. Among these models, we can find MDSDF (Multidimensional Synchronous Dataflow) [10] and Array-OL (Array Oriented Language) [11].

In our study, we are interested in modeling parallel applications using the *Array-OL* model and one of its development environments, *Gaspard2*¹. This model allows to easily program intensive signal processing applications. It is used to specify the *data parallelism* and the *data dependencies* between tasks. The Gaspard2 environment is a model driven system-on-chip co-design environment. The designs are based on a Y development model in which, from two UML² models describing the application and the hardware architecture, the application is mapped to the hardware architecture in an association model. This association model is then projected onto lower level descriptions such as simulation or synthesis models.

¹ <http://www.lifl.fr/west/gaspard>

² <http://www.uml.org>

Signal processing applications modeled in Array-OL can be considered as purely data flow based and only represent an intensive data processing without any reaction or control concepts. The goal of our work is to introduce, in the Gaspard2 application metamodel, the control concepts and the possibility to change running modes according to the execution context of the studied applications.

The introduction of control into data parallel applications requires the definition of a clear model and a rigorous semantics allowing to take various types of applications into account, mixing control and data parallel processing. This concept gives a *reactive* behavior to the studied parallel applications. In this case, the systems are not only describable by transformational relations, specifying outputs from inputs, but also by relations between outputs and inputs via their possible combinations in time. Consequently, the combination of descriptions including complex sequences of events, actions, conditions and information flow allows to synthesize the global behavior of a *reactive system* [1].

The complexity of reactive systems comes from the complex characteristics of the reactions to the different occurrences of discrete events [2]. This complexity can make it difficult to model the behavior of such systems and exposes them to errors. It becomes necessary to introduce rigorous design methods and formalisms to specify the behavior of these critical systems. Among these formalisms, the *synchronous approach* represents a significant contribution to this field [3]. It is based on the *synchrony hypothesis* which considers the execution of a reactive system as an infinite succession of instantaneous reactions.

In this paper, we use the concept of synchronous reactive systems to introduce the control parts in the Gaspard2 application metamodel. The basic idea is inspired by the principles of *mode-automata* [4] used in the case of synchronous reactive systems to clearly express the different running modes of an application and the conditions of switching between modes.

The metamodel that we propose must, on the one hand, clearly separate control and data flow parts [12], and on the other hand, respect concurrency, parallelism, determinism and compositionality. In this model, the computation part represents a set of parallel tasks (signal processing, image processing, ...) while the control part represents the switching conditions between the different running modes of the tasks according to control values. These values can be provided by the environment (pressing a button, temperature changes, ...) or by the computation part (result of a preceding computation, dependency between tasks, ...).

In the following sections, and after a brief presentation of the used concepts, we study the possibility to take into account the different changes of mode in a parallel application metamodel. This concept requires a good definition of the degree of granularity of applications or the clock signal for which the control values can be taken into account. Our work is based on the Gaspard2 application metamodel [17] and proposes a solution for the modeling of control automata, running modes and the link between the two in the case of a synchronous approach.

In the literature, few works have been proposed to introduce the control into a parallel computation field. For instance, in 1998, Smrandache studies application co-design using the Signal relational language and the Alpha functional language [14]. This approach uses the C language as a support of communication between the two levels of specification and does not define any specification model allowing the modeling of parallel applications with the control concepts. Another example can be found in Ptolemy [13] which proposes a multidimensional computation model (MDSDF) and an automata model (FSM). However, the combination of these two concepts has never been studied.

2 Context

In this section, we give a brief definition of the concepts used in our study. The context of this study can be classified into two main parts. The first part is related to intensive signal processing applications and presents the Array-OL language and its Gaspard2 environment in particular. The second part is about synchronous reactive systems and the mode-automata concept. For both parts, we also study the existing UML devices and concepts which can be used to model these kind of applications.

2.1 Intensive Signal Processing

Array-OL. Array-OL (Array Oriented Language) is a specification language allowing to express parallel applications by the way of data dependencies. This language has mainly been introduced to model intensive signal processing applications. It is based on a multidimensional model and makes it possible to express the whole potential parallelism of these applications (data or task parallelism).

In the Array-OL model, the different tasks are connected to each other using data dependencies. The expression of these dependencies initially allows to define a minimal partial order on the execution of these tasks. The compiler can then complete this partial order in an efficient parallel execution. When a data dependency is expressed between two tasks, it means that one of these two tasks needs whole or part of the data produced by the other task to be able to perform its computations. The compilation of Array-OL models has largely been studied by Soula, Dumont et al. [15, 16].

The Array-OL models can have hierarchical compositions on several description levels. In a hierarchical model, the data dependencies are mainly approximative until the lowest level where these dependencies are completely expressed. The description of an application in Array-OL uses two models. The *global model*, defines the sequence of the different parts of the application, in other words, the task parallelism, and the *local model*, specifies the elementary actions performed on the table elements and the existing data parallelism of the different tasks.

The **global model** is a simple directed acyclic graph where each node represents a task and each edge represents a multidimensional array. The number of incoming or outgoing arrays is not limited. These multidimensional arrays may have one infinite dimension that is generally used to represent time.

At the execution time of each task, the incoming arrays are consumed and the output arrays are produced. The number of produced or consumed arrays is equal to the number of inputs or outputs edges for each task. The graph relating to the global model thus represents a task graph and not a data flow graph. There is not implicit repetition of the task graph as in stream languages. The streams are explicit in the arrays (by the infinite dimension for example). The model is thus strictly single assignment at the array element level.

Using only the global model of the application, it is possible to schedule the execution of the different tasks. However, it is impossible to express the data parallelism present in our application. For this reason, the introduction of the local model becomes necessary.

The **local model** allows to express the data parallelism expressed by data parallel repetitions. In this model is a repetition constructor, where each repetition of the embedded task is independent. That repeated task is applied to a subset of the elements of each input array to produce data elements stored in each output array.

The size and the shape of the element set associated to an array is the same from a repetition to another. In the local model, each element set is called a *pattern*, and in order to express hi-

erarchical constructions, the patterns are themselves multidimensional arrays. The construction of the different patterns requires a set of information:

- O : the origin of the reference pattern
- D : the shape of the pattern (size of all the dimensions)
- P : a “paving” matrix allowing to describe how the patterns cover the array
- F : a “fitting” matrix describing how to fill the pattern with the array elements
- M : the shape of the array (size of all the dimensions)

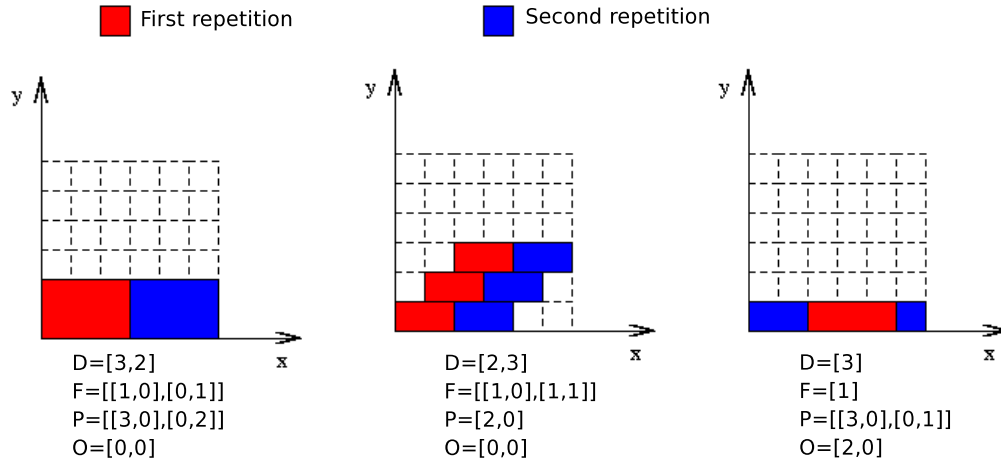


Fig. 1. Paving examples

The *paving matrix*, P , is composed of a set of paving vectors used to identify the origin of each array pattern, one for each repetition, as shown by figure 1. These pattern origins are defined by the set $\{\forall i \in \text{RepetitionSpace}, (O + P \times i) \bmod M\}$.

The *fitting matrix* is represented by a set of vectors where each vector is associated to a pattern dimension. The fitting vectors are used to identify the array elements of each pattern starting from the origin point as shown by figure 2. The pattern elements are defined by the set $\{\forall j, 0 \leq j < D, (O + P \times i + F \times j) \bmod M\}$.

The $\bmod M$ part of the above formula ensures that all the data elements of a pattern correspond to array elements. The modulo allows to handle cases such as toroidal physical spaces or the cyclic frequency dimensions obtained after an FFT or a DCT.

Gaspard2 Environment. Gaspard2³ is an under development model driven Integrated Development Environment for SoC (System on Chip) visual co-modeling. It extends the Array-OL language and allows modeling, simulation, testing and code generation of SoC applications and hardware architectures.

The Gaspard2 environment is mainly dedicated to the specification of signal processing applications. It is based on a *model oriented* methodology according to a Y design flow (figure 3).

³ <http://www.lifl.fr/west/gaspard/>

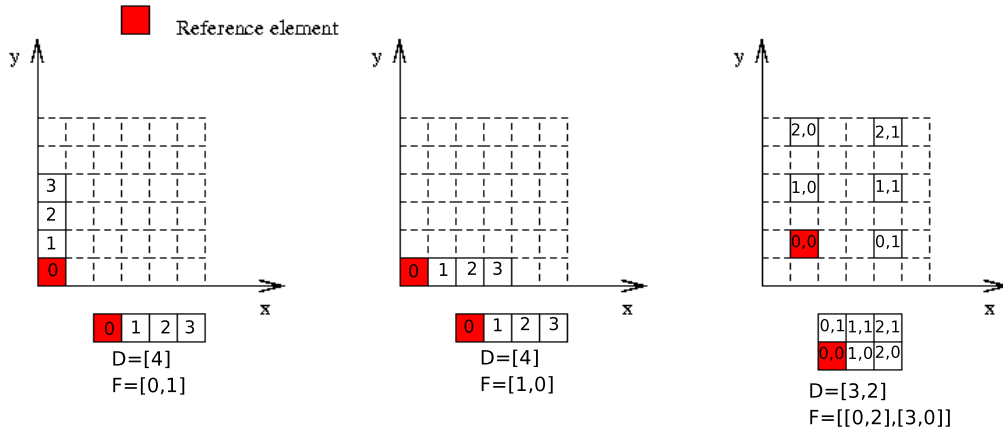


Fig. 2. Fitting examples

In this approach, the concepts and semantics of each design level are represented independently of the execution or simulation platforms.

The starting point in Gaspard2 consists in modeling the application, the architecture and the association by using a Gaspard2 UML2.0 profile [17]. These models are then imported in an Eclipse⁴ plug-in via model transformation to a specific metamodel using ModTransf⁵, and studied by applying mapping and scheduling algorithms and automatic SystemC⁶ code generation.

The model definitions of the Gaspard2 environment are based on a component oriented methodology. This methodology makes it possible to clearly separate the application and the hardware architecture and facilitate the re-use of existing software and hardware IPs. It also defines an association model that gives directives on the mapping of the application on a particular architecture.

UML Profile for Modeling Gaspard2 Components. In [18], Arnaud Cuccuru proposes an UML2.0 profile for modeling the various concepts of Gaspard2 models. This profile is defined around five packages: `component`, `factorization`, `application`, `hardwareArchitecture` and `association` as shown by figure 4.

In this profile, we can find the three main concepts of the Y model in Gaspard2: `application`, `hardwareArchitecture` and `association`. The `application` and the `hardwareArchitecture` packages share the same component definition introduced in the `component` package. The `association` package introduces concepts giving directives on the mapping of the application on a hardware architecture. The `factorization` package contains structural factorization mechanisms inspired by the Array-OL model. These mechanisms make it possible to express the multidimensional aspect and the relation between the pattern elements of inputs and outputs arrays of a task.

⁴ <http://www.eclipse.org>

⁵ <http://www.lifl.fr/west/modtransf>

⁶ <http://www.systemc.org>

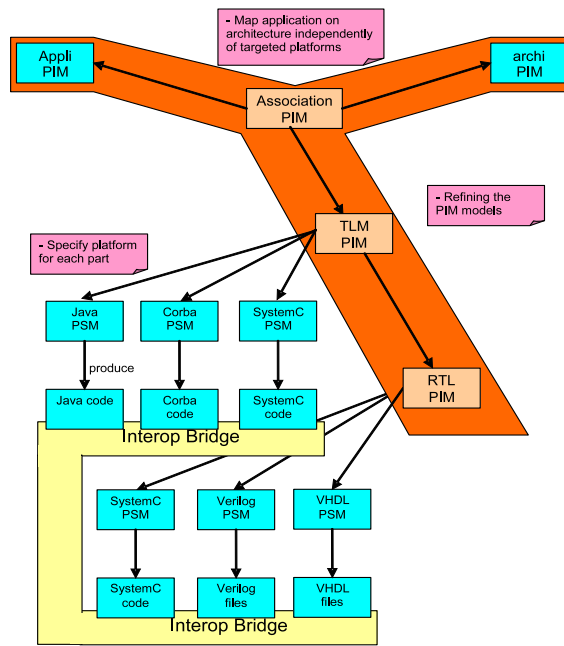


Fig. 3. Representation of the Y model relating to the Gaspard2 flow design

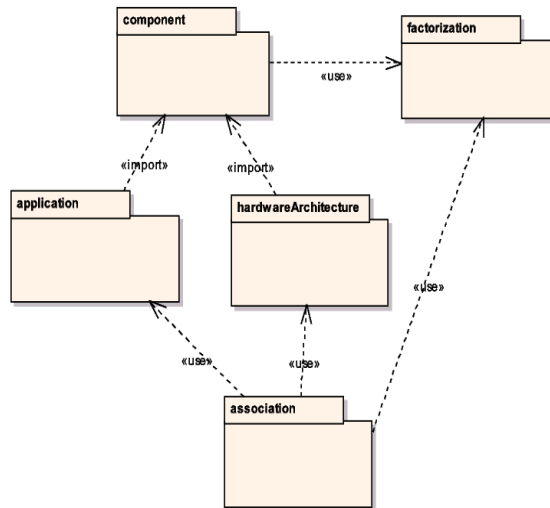


Fig. 4. Different packages of the Gaspard2 profile

In this paper, we are only interested in the application part. This part allows to model the data dependencies and the parallelism of applications based on the Array-OL model. In the application metamodel, the `AppComponent` concept refines the `Component` concept by adding an applicative connotation. The application components can be seen as a set of functions. These functions perform calculations on the input data coming from their external environment through input ports (provided ports in UML terminology) and produce results to their environment through output ports (required ports in UML terminology). These application components can be mainly described by using three types of components: `AppElementaryComponent`, `AppComponent` and `AppRepetitiveComponent`.

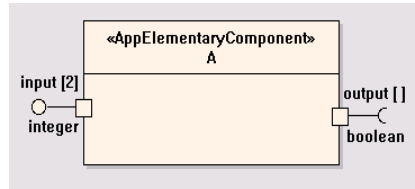


Fig. 5. Example of an *AppElementaryComponent* component

The `AppElementaryComponent` component represents a particular component which does not have any description of structure or behavior. Figure 5 represents a simple example of an `AppElementaryComponent`. This example defines a function taking as input a pattern of two integer elements and produce as result a pattern of only one boolean element.

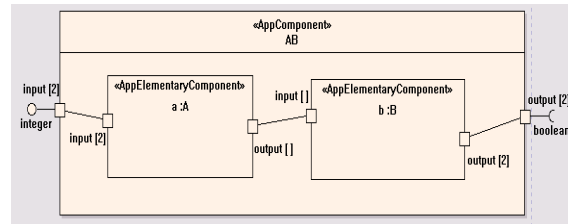


Fig. 6. Example of an *AppComponent* component

The `AppComponent` component is used to define compound components. For example, if the result of the elementary task *A* presented in figure 5 is used by another elementary task *B* which produces a pattern of two boolean elements, then the component `AppComponent` is used to group and represent the relation between the two tasks *A* and *B* as shown by figure 6.

The `AppRepetitiveComponent` component allows to describe the repetition of the different tasks modeled in this component. This repetition relates to the repetitive concept of the Array-OL model. Thus, by using a special connector (`RepetitiveConnector`), it is possible to give information on the origin, the paving and the fitting matrices for each input or output array. Figure 7 gives a simple example on the repetition of the *AB* task presented in figure 6. In this example, the model receives as input an array of two dimensions $8 \times *$ representing an infinity

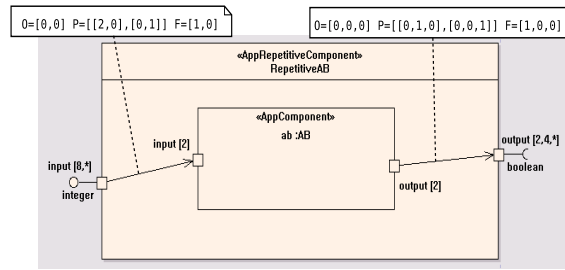


Fig. 7. Example of an *AppRepetitiveComponent* component

of 8 integer vectors, and produces as result an array of three dimensions $2 \times 4 \times *$ representing an infinity of 2×4 boolean arrays.

The Gaspard2 application metamodel allows to describe the data dependencies and the potential parallelism present in applications. However, this metamodel does not contain any representation of the control and the possibility of changing running modes according to the execution context of the studied applications.

In the following section, we study the introduction of control into the Gaspard2 application metamodel in order to take more general parallel applications mixing control and data processing into account. The introduction of control into a parallel application can be done by giving a reactive behavior which has been largely studied in the case of the synchronous reactive systems.

2.2 Synchronous Reactive Systems

Reactive Systems and Synchronous Approach. *Reactive Systems* are computer systems that react continuously to their environment, by producing results at each invocation [1]. These results depend on data provided by the environment, and on the internal state of the system.

Specification of software or hardware reactive systems behavior is complex. It can lead to important errors that are difficult to fix. Indeed, such systems are not only described by transformational relationships, specifying outputs from inputs, but also by the links between outputs and inputs via their possible combinations in one step. Modeling reactive systems is therefore a difficult activity.

In the beginning of the 80's, the family of synchronous languages and formalisms has been a very important contribution to the reactive system area [19]. Synchronous languages have been introduced to make programming reactive systems easier. They are based on the *synchrony hypothesis* that does not take reaction time in consideration. Each activity can then be dated on the discrete time scale. This hypothesis considers that each reaction is instantaneous and atomic.

The synchronous languages, like Lustre [6], Esterel [7] or Signal [8], are devoted to the design, programming and validation of reactive systems. They have a formal semantics and can be efficiently compiled into C code, for instance. Moreover, these formalisms make it possible to validate and verify formally the behavior of the system. In this field, we often speak about tools and approaches for simulation, verification and code generation for reactive systems specified in a synchronous language.

Mode-Automata. Mode-automata have been proposed in [4]. They introduce, in the domain-specific data-flow language Lustre for reactive systems, a new construct devoted to the expres-

sion of *running modes*. It corresponds to the fact that several definitions (equations) may exist for the same output, that should be used at distinct periods of time. This concept allows to decompose the specification of the system into several tasks called *modes* by assigning data operations to discrete states.

A mode-automaton is an input/output automaton. It has a finite number of states, that are called *modes*. At each moment, it is in one (and only one) mode. It may change its mode when an event occurs. For each mode, a transfer function determines the values of output flows from the values of input flows. Mode automata can be combined in order to design hierarchical models. They generalize both bounded Petri nets and block diagrams. The structure of mode-automata allows to clearly specify where the modes differ and the conditions for changing modes which makes it possible to better understand the behavior of the system.

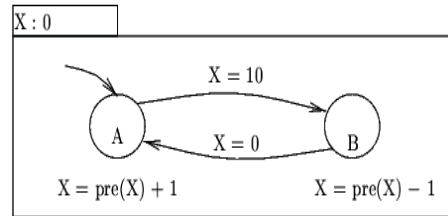


Fig. 8. Mode-automaton: simple example

Figure 8 represents a simple example of mode-automaton. It has two states, and equations attached to them. The transitions are labeled by conditions on X. The important point is that X and its memory are *global* to both states. The only thing that changes when the automaton switches its state is the transition function; the memory is preserved.

Synchronous Behavior and Automaton Structure UML Modeling. The UML modeling of the synchronous behavior and its concepts are an interesting research subject. For example, in [20], R. De Simone and C. André propose a UML subprofile, using a synchronous version of state and activity diagrams, to express the synchronous reactive behavior. Their proposition gives a synchronous solution to the UML state machine limitations by allowing the description of the absence and the simultaneous events.

In this paper, we limit our study to the discrete events of UML state machines. For this case, we propose to model the control automaton structure by a UML StateCharts model [9].

The UML StateCharts specify a set of concepts used for modeling discrete behavior through finite state-transition systems. They are an object-based variant of Harel StateCharts [5]. The semantics of the UML StateCharts are described in terms of the operations of the hypothetical machine that implements a state machine specification. In this state machine, states represent the existence conditions of the class they define, and the transitions are represented by directed arcs with named event triggers optionally followed by actions. Figure 9 gives a small example of an UML StateCharts.

The main characteristic of the UML StateCharts is that they have a *run-to-completion* semantics which imposes that no other event can be taken into account before the processing of the previous event is fully completed. This assumption simplifies the transition function since concurrency conflicts are avoided during the processing of events. Moreover, the absence of an

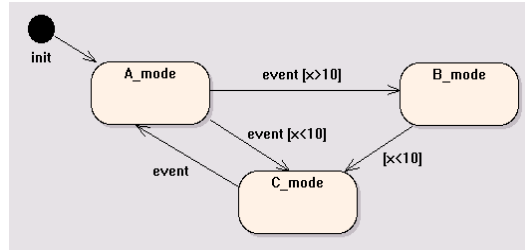


Fig. 9. Simple example of a StateChart

event instance cannot be taken into account in the UML StateCharts which makes it difficult to express highly reactive system behavior.

The only reason for which we have chosen to use the UML StateCharts model is the simplicity of this model which has a well defined semantics. However, it is always possible to model the control part by using a more sophisticated UML metamodel for the specification of the automaton structures as for example the UML subprofile introduced in [20].

3 Degrees of Granularity for the Control of Parallel Applications

The introduction of the control into data parallelism applications requires the definition of a *degree of granularity* for these applications. This concept allows to delimit the different execution cycles or *clock signals* in which it becomes possible to take the control values and then the various changes in the running modes into account.

According to the studied application and the selected semantics, several definitions of the degree of granularity are possible. In this section, we study a particular approach allowing to define the various moments at which it becomes possible to take the changes of modes into account. This approach, which we call *synchronous approach*, supposes that the data and control values are available at the same time and follow the same basic clock. In this context, the control model produces a *mode table* which is used by the application to determine the execution mode for the different repetitions. In other words, the mode table only represents an input data like all other input computation data.

The proposed model can have a first very simplistic impression. However, this approach imposes a good choice of the degree of granularity to be able to take the data values into account at the same time as the control values by respecting the semantics and the behavior of the application.

To define the degree of granularity in the Gaspard2 application metamodel, we need to modify the granularity of input and output patterns, and consequently, to modify paving and fitting matrices. This approach can also be seen as a *data oriented* approach since it depends on the input data and just takes the necessary set of data to perform a controllable computation.

For a better understanding of this concept, we consider the simple example of an Array-OL model represented by figure 10. In this model, the system takes as input a three dimensional array of $4 \times 4 \times *$ and returns as output a three dimensional array of $4 \times 2 \times *$. The executed task T is a parallel and repetitive one. For each repetition, an instance of the composed task AB processes an input pattern of two elements to produces an output pattern of one element.

In the following, we introduce a control module which makes it possible to change the running modes of the elementary task A . Since the mode table is produced by the control module

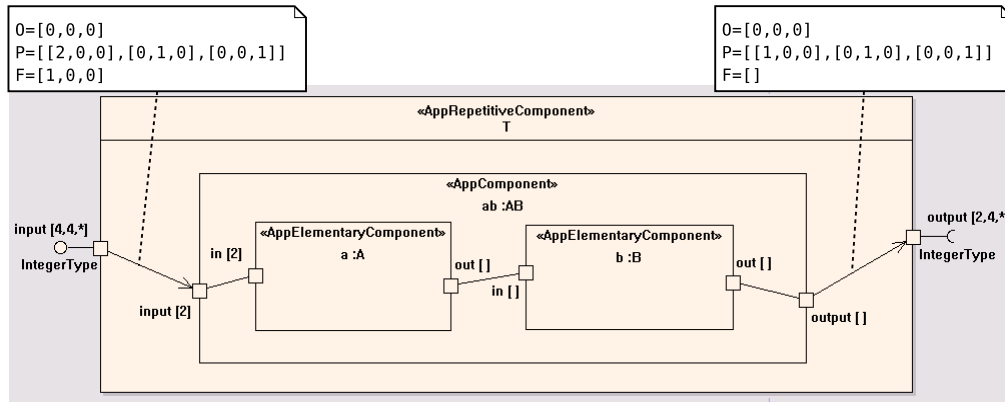


Fig. 10. Simple example of an Array-OL model

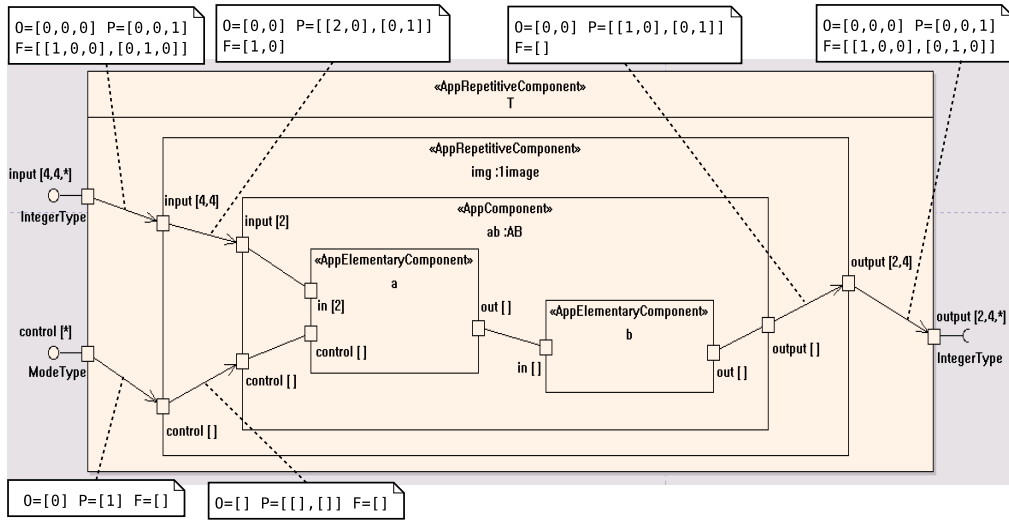
at the beginning of the application, the definition of the production rate of the mode values always depend on the behavior of the studied application.

The first case is the simplest case for which we consider that the change of mode refers to the whole output image (figure 11.(a)). In this model, the system takes as input an infinity of 4×4 images and a control array, and produces as output an infinity of 4×2 images. In this example, only one control value corresponds to each 4×4 input image. An instance of the repetitive task *image* is performed for each input image. It takes as input a pattern of two elements and a control value to produce as output a pattern of one element as explained by figure 11.(b). In this case, the same control value is used for all patterns of the same image. The degree of granularity chosen for this application thus corresponds to the calculation of a complete image.

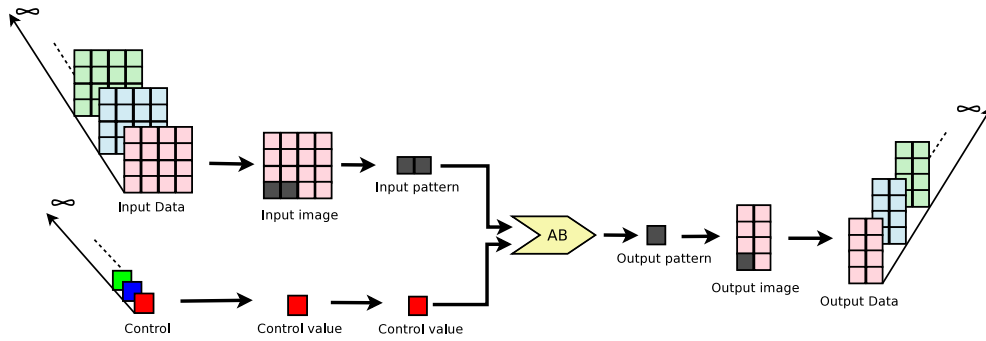
Another possible situation consists in authorizing different running modes for each point of the output image. To do that, we modify the input and output data flow to adapt them to the control flow as shown by figure 12.(a). In this case, each input pattern of two elements corresponds to one control value. The different patterns of the same image can have different control values as explained by figure 12.(b). The degree of granularity corresponds to the calculation of only one point of the output image. It is also possible to consider the change of modes for only one line of the output image, several lines, a column, several columns, and so on. . .

The definition of the degree of granularity for an application can also depend on the implementation or on the mapping of this application on a particular architecture. For example, if we know that our system is able to process two images in parallel, we can consider that the application consumes and produces an infinity of two images. In this particular case, we have considered the same control value for each of the output images. However, it is also possible to have different control values for each point of the output image. In this case, the modes table has a multidimensional structure.

The introduction of the degree of granularity concept into a parallel model allows to define the clock signal in which it becomes possible to take into account the various changes of modes in a parallel application. Our approach is a synchronous approach in which the mode tables are regarded as a simple input data, and the choice of the degree of granularity depends on the behavior of the studied application. Using this approach, users have a total freedom to define the degree of granularity for their applications according to the required behavior.

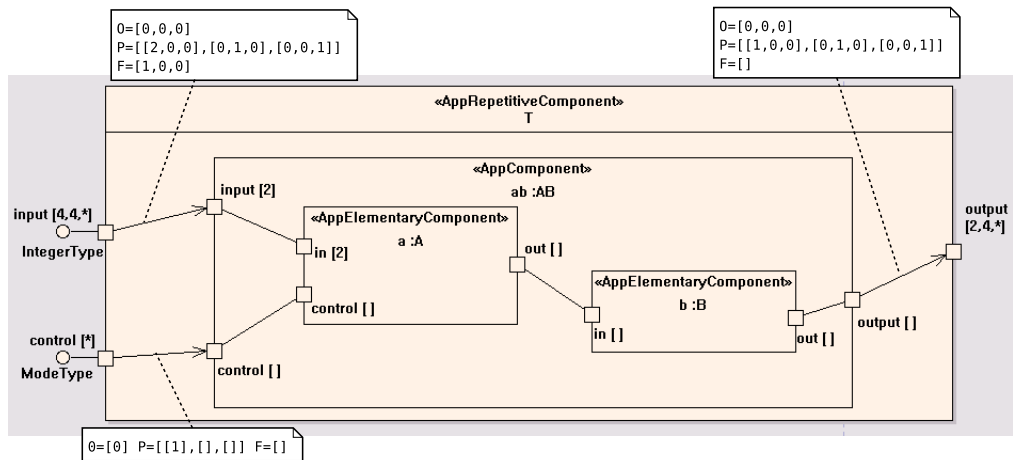


(a) UML model

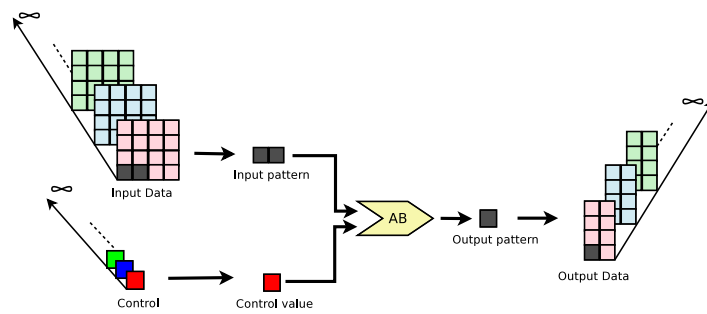


(b) Global view

Fig. 11. Example of a control introduction for the whole output image



(a) UML model



(b) Global view

Fig. 12. Example of a control introduction for one point of the output image

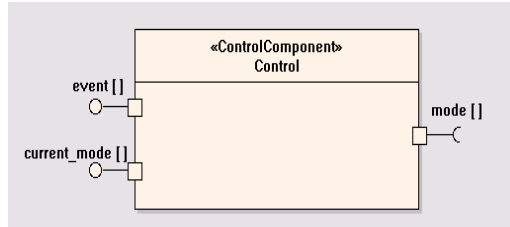


Fig. 13. Example of a *ControlComponent* component

4 Introducing Control in the Gaspard2 Data-Parallel Metamodel

In this section, we study the introduction of the control models into the Gaspard2 application metamodel. To do that, it is necessary to define a modeling concept for the control parts, the different running modes and the link between control and parallel processing.

4.1 Modeling of the Control Part

The control part represents an automaton structure based on the mode-automata concept. It allows to clearly specify the various running modes of the system and the switching conditions between modes. For modeling this part and introducing it into the Gaspard2 profile, we define a particular component stereotyped *ControlComponent*. This component produces a running mode table, possibly multidimensional, depending on the input events and its current mode (figure 13). To each *ControlComponent* component is associated the transition function of the control automaton. In other words, it *performs one step* of the automaton. In our UML metamodel, this can be represented by an *activity diagram* [9].

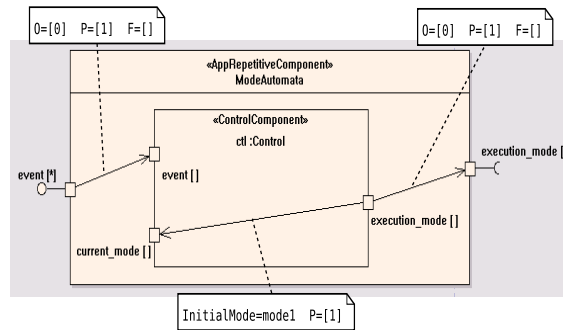


Fig. 14. Representation of the control repetition

Modeling the repetition of the control component, when the automaton *repeatedly performs steps*, consists in introducing a *dependency relation* between the various instances of the transition function. In this case, the mode-automata structure can be represented by a transition and a *control dependency* between the different instances as shown by figure 14. In this model,

an additional information `InitialMode` is introduced on the control dependency relation to specify the initial mode of the controlled task.

In the case of a more complex control automata, it is difficult to understand the control model if we represent the automaton structure by a `ControlComponent` component and a dependency relation, and its behavior by an activity diagram. For clarity reasons, it is preferable to represent the control part by an explicit automaton structure in terms of states and transitions.

For these reasons, we propose to introduce, in the application metamodel, a particular component stereotyped `AutomatonComponent`. This component receives as input one or more event tables and produces as output a mode table. To keep the general semantics of a reactive control automaton, the input and output tables of the `AutomatonComponent` are regarded as *data flows*. This hypothesis gives to this component a different semantics from that of the Gaspard2 applications.

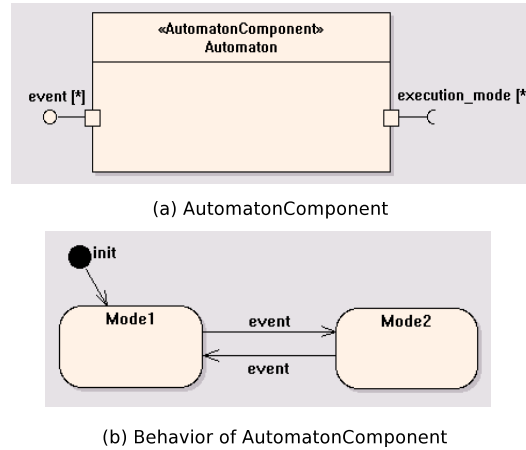


Fig. 15. Representation of the control automata by StateChart

In the Gaspard2 application metamodel, the dimensions of the arrays represent indifferently time or space. The order of execution is only constrained by the defines data dependencies. However, in the control automaton structure, the introduction of the control dependency between the different instances of the transition function imposes the introduction of the flow concept to the input and output arrays of an automaton component. This dependency relation makes it possible to memorize the preceding states of the automaton and then to respect the general semantics of a control automaton. In our metamodel, and when the control part is described by an automaton structure, we consider that the flow concept is implicitly described for the different input and output arrays of the automaton component. To model the behavior of the control automaton in our metamodel, we propose to use the UML StateCharts [9] structure as shown by figure 15.

4.2 Modeling of the different Running Modes

The controlled application, which can be replaced by different running modes, is represented by a particular component stereotyped `AppControlledComponent`. This component consists

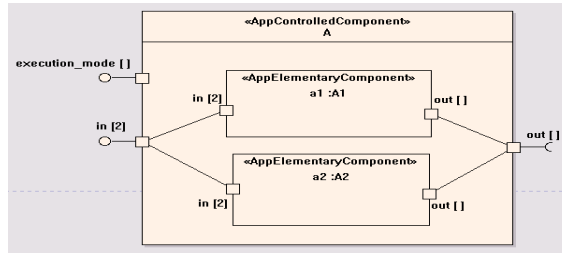


Fig. 16. Representation of the different running modes

of several running modes, each mode being represented by a *part* relating to the predefined Gaspard2 components. The different parts in the same `AppControlledComponent` component must have the same interface and are not connected between them. At each moment, one and only one part is activated at the same time according to the mode information available on the Mode port. Figure 16 represents the modeling of two running modes for the elementary task A presented in the example of figure 11.

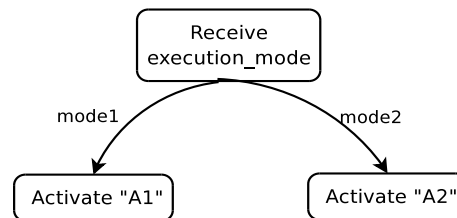


Fig. 17. Behavior of the `AppControlledComponent` component A

As shown by figure 16, the component `AppControlledComponent` has a particular port stereotyped `Mode`. This port makes it possible to specify the running mode to be activated, it is never connected and is only used by an activity diagram associated to the `AppControlledComponent` component to express its behavior (figure 17).

The expression of the repetitive factor around the component `AppControlledComponent` just consists in using the Gaspard2 predefined repetitive component stereotyped `AppRepetitiveComponent` as shown by figure 18.

4.3 Modeling the Link between the Control Part and the Different Running Modes

At each computation time, one and only one running mode is activated according to the information provided by the control part. This information can be the name of the mode to be activated or any other index allowing to distinguish the modes in a clear and single way. In our metamodel, we suppose that the control part provides to the computation part an information on the name of the mode to activate. According to this information, the computation part can activate or not the various modes of the system. The application of these concepts for the example of figure 12 is represented by the model of figure 19. In this example, the control and the controllable parts are represented in the same repetitive component. It is also possible to

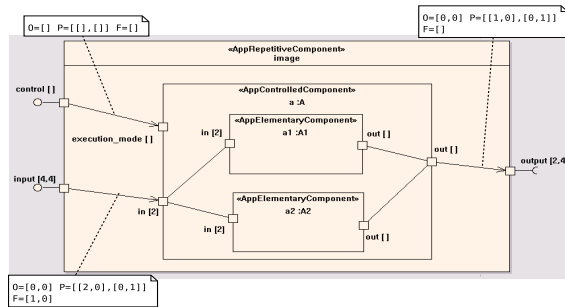


Fig. 18. Representation of the repetition of the *AppControlledComponent* component

separately represent the repetitive part of the control and that of the computation as it is shown by figure 20.

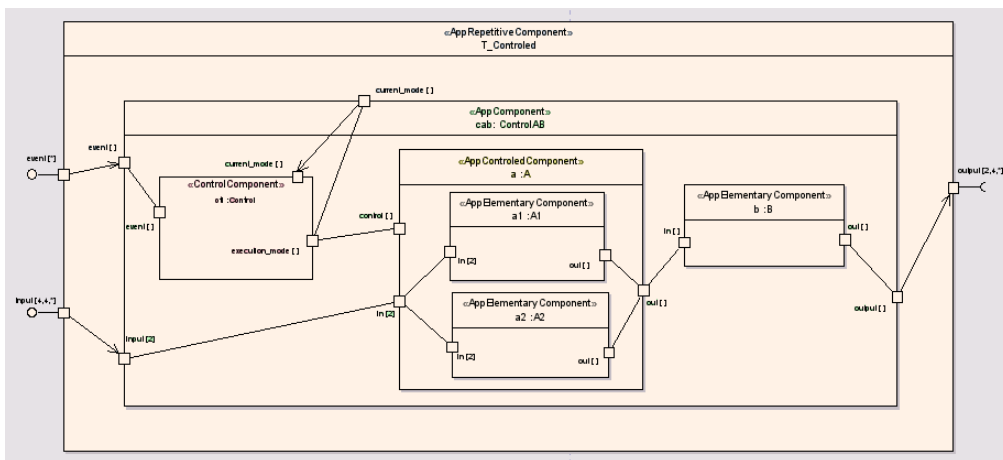


Fig. 19. Representation of the control part and the running modes in the same repetitive component

Figures 19 and 20 represent the case where the control part is modeled by a component of type *ControlComponent*. This representation completely respects the semantics of the Gaspard2 application metamodel. However, if we want to represent the control part by an automaton structure modeled by a StateChart, it is necessary to represent the two parts, the automaton and the repetitive computation, in a separated way as it is shown by figure 21.

The introduction of control into the Gaspard2 application metamodel supposes that the control values (mode table) must be present with the data values to launch the calculation model. The control part must also follow the arrival rate of the events since a control dependency is defined on the various repetitions of the control automaton. This approach imposes the introduction of the flow concept in the Gaspard2 application metamodel which can break the assumption of the *unified space-time specification* by imposing a partial order on the execution of the different parallel tasks. This unification is one of the main characteristic of the Gaspard2

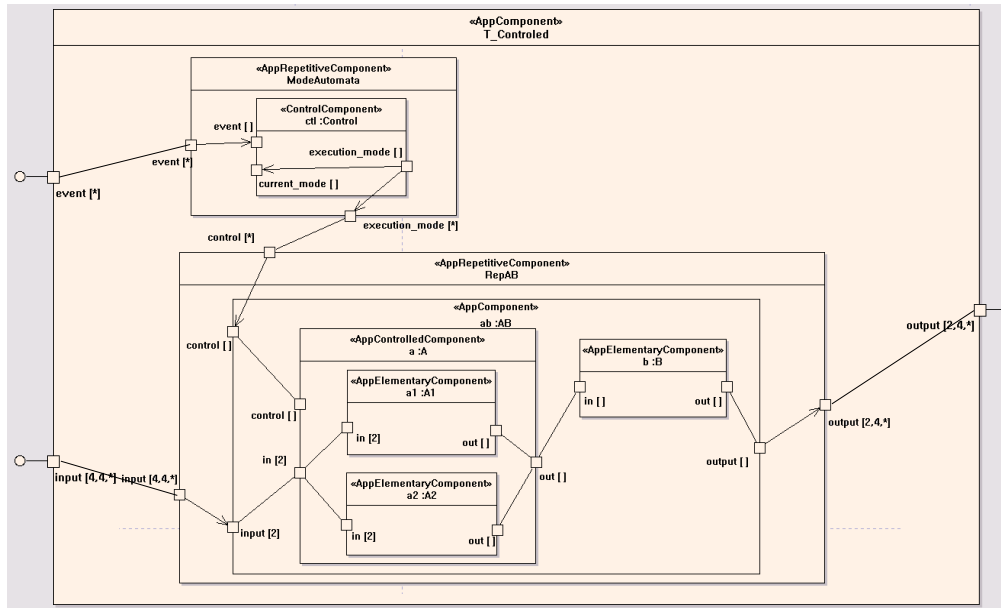


Fig. 20. Representation of the control part and the running modes in different repetitive components

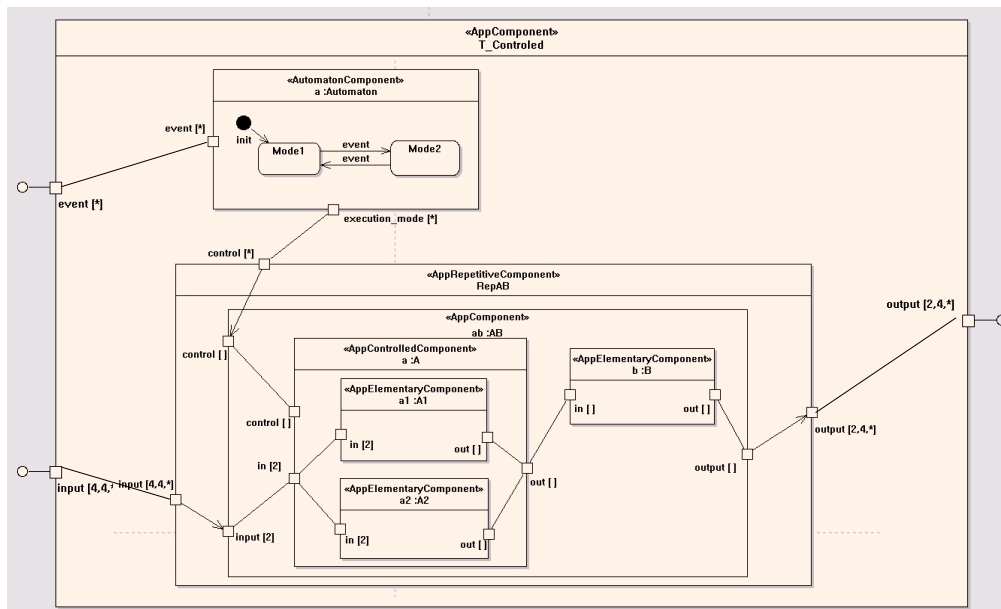


Fig. 21. Representation of the control part and the running modes using an automaton structure

metamodel and can be useful for modeling more general applications. However, the introduction of the `flow` concept into the metamodel can facilitate the understanding of the model and makes it more realistic since the input values, either control or data, are mainly generated by sensors and thus represent a control or a data flow structure. Moreover, the logical division of time between discrete instants allows to properly define mathematical models and operational semantics. Our approach also strictly respects the parallelism and concurrency of the model. It is deterministic, compositional and can easily be introduced into Gaspard2 application metamodel.

5 Conclusion and Future Work

In this paper, we have studied the introduction of the control concepts into the Gaspard2 application metamodel. Our idea is mainly inspired by the synchronous reactive systems domain and in particular by the concept of mode-automata. The proposed metamodel is based on a clear separation between control and data parallel parts. It respects the concurrency, the parallelism, the determinism and the compositionality.

We have shown that the introduction of control into a data parallel domain requires to define the different instances allowing to take the various changes of modes into account. To do that, we have proposed to introduce the notion of degree of granularity in the parallel applications and the control dependency relation between the different instances of the control automata. The studied approach is a synchronous one, which supposes that data and control values must be present to be able to execute a computation function.

The main goal of our work consists in proposing a UML solution for the modeling of control automata, the different running modes of an application and the link between the control and the data parallel parts. Our metamodel allows to study more general parallel systems mixing control and data processing, and gives users more freedom to express the behavior of their applications.

In future work, we will propose the introduction of the control concepts into architecture and association Gaspard2 metamodels allowing to take the configurability concept into account for the architecture models and a better use of the mapping and scheduling algorithms. We will study the relation between the parallel and hierarchical composition of the application model and the parallel and hierarchical automata, in particular for verification processes.

We also want to transform or compile our control/data parallel metamodel into a synchronous language (like Lustre). This would make it possible to take advantage of the various existing tools for simulation, verification and automatic code generation. Thus, it can be interesting to compare the analysis results and the generated codes obtained using the Gaspard2 environment and those obtained using the corresponding synchronous model.

References

1. David Harel and Amir Pnueli, *On the development of reactive systems*, In K. R. Apt, editor, Logics and Models of Concurrent Systems, **13**, NATO ASI Series, Springer-Verlag, pages 477-498, New York, 1985
2. David Harel, Hagi Lachover, Amnon Naamad, Amir Pnueli, Michal Politi, Rivi Sherman, Aharon Shull-Trauring, and Mark Trakhtenbrot, *STATEMATE: A working environment for the development of complex reactive systems*, IEEE Transactions on Software Engineering, **16 (4)**, pages 403-414, April, 1990

3. Nicolas Halbwachs, *Synchronous programming of reactive systems, a tutorial and commented bibliography*, In Tenth International Conference on Computer-Aided Verification, CAV'98, Vancouver (B.C.), LNCS 1427, Springer Verlag, June, 1998
4. Florence Maraninchi and Yann Rémond, *Mode-automata: About modes and states for reactive systems*, European Symposium On Programming, Springer Verlag, LNCS 1381, Lisbon, Portugal, March, 1998
5. David Harel, *StateCharts: A visual Formalism for Complex Systems*, Science of Computer Programming, **8**, pages 231-274, 1987
6. N. Halbwachs, P. Caspi, P. Raymond and D. Pilaud, *The synchronous data-flow programming language LUSTRE*, Proceedings of the IEEE, **79(9)**, pages 1305-1320, September, 1991
7. Gerard Berry and Georges Gonthier, *The Esterel Synchronous Programming Language: Design, Semantics, Implementation*, Science of Computer Programming, **19(2)**, pages 87-152, Amsterdam, November, 1992
8. P. Le Guernic, T. Gautier, M. Le Borgne and C. Le Maire, *Programming Real-Time applications with SIGNAL*, Another Look at Real-Time Programming, Proceedings of the IEEE, **79**, pages 1321-1336, September, 1991
9. Bruce Powel Douglass, *Real Time UML Third Edition, Advances in the UML for Real-Time Systems*, Addison Wesley, Object Technology Series, ISBN: 0-321-16076-2, February, 2004
10. Praveen K. Murthy and Edward A. Lee, *Multidimensional synchronous dataflow*, IEEE Transactions on Signal Processing, July, 2002
11. A. Demeure and Y. Del Gallo, *An array approach for signal processing design*, In Sophia-Antipolis conference on Micro-Electronics, SAME'98, France, October, 1998
12. Ouassila Labbani, Jean-Luc Dekeyser and Pierre Boulet, *Mode-automata based methodology for Scade*, In Springer, Hybrid Systems: Computation and Control, 8th International Workshop, LNCS series, pages 386-401, Zurich, Switzerland, March 2005
13. Edward A. Lee, *Overview of the Ptolemy Project*. Technical Memorandum UCB/ERL M01/11, University of California, Berkeley, March, 2001. Url: ptolemy.eecs.berkeley.edu/publications/papers/01/overview
14. Irina Madalina Smarandache, *Conception conjointe des applications régulières et irrégulières en utilisant les langages Signal et Alpha*, PhD Thesis, Université de Rennes 1, October, 1998
15. Julien Soula, *Principe de compilation d'un langage de traitement de signal*, PhD Thesis, Laboratoire d'Informatique Fondamentale de Lille, Université des Sciences et Technologies de Lille, December, 2001
16. Philippe Dumont and Pierre Boulet, *Another multidimensional synchronous dataflow: Simulating Array-OL in ptolemy II*, Research Report RR-5516, INRIA, March, 2005
17. Arnaud Cuccuru, Jean-Luc Dekeyser, Philippe Marquet and Pierre Boulet, *Towards UML 2 extensions for compact modeling of regular complex topologies - A partial answer to the MARTE RFP*, In MoDELS/UML 2005, ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems, Montego Bay, Jamaica, October, 2005
18. Arnaud Cuccuru, *Modélisation unifiée des aspects répétitifs dans la conception conjointe logicielle/matérielle des systèmes sur puce à haute performances*, PhD thesis, Laboratoire d'Informatique Fondamentale de Lille, Université des Sciences et Technologies de Lille, September, 2005
19. N. Halbwachs, *Synchronous programming of reactive systems*. Kluwer Academic Publishers, ISBN: 0792393112, 1993
20. Robert de Simone and Charles André, *Towards a "Synchronous Reactive" UML subprofile?*, Workshop on Specification and Validation of UML Models for Real Time and Embedded Systems (SVERTS), San Francisco (CA), October, 2003