From Modeling/Simulation with Scilab/Scicos to Optimized Distributed Embedded Real-Time Implementation with SynDEx

Yves Sorel

INRIA, Domaine de Voluceau Rocquencourt BP 105
78153 Le Chesnay Cedex, France,
yves.sorel@inria.fr,

Abstract. This paper presents the AAA methodology and its system-level CAD software SynDEx, used to optimize the implementation of distributed embedded real-time applications, and how coupling this software with the scientific software package and the dynamic systems modeler and simulator Scilab/Scicos, allows to provide a seamless flow which improves the design safety, and decreases the development cycle thanks to functional and timing simulations, as well as automatic code generation.

1 Introduction

Distributed embedded real-time applications are of crucial importance in domains such as avionic, automobile, telecommunication, mobile robotic, etc. Mainly based on digital electronics including software, this latter part of the application is rapidly growing. Because they have to meet constraints in terms of distributed resources as well as in terms of time, and in addition to be optimized because of embedding considerations, the design process of such applications is particularly complex. It is composed of two main steps.

The first one consists in specifying the functionality the application has to perform, usually, using a block diagram approach where the different functions are interconnected together. A function may be in turn described in terms of other interconnected functions leading to hierarchical specifications. In the more complex case the functionality is related to its physical environment which must be taken into account in order to control it. The parameters of the control laws, as well as the control laws themselves, are determinated through a simulation process involving the environment which must be carefully modeled. Usually, the environment is modeled as a continuous process whereas the control laws are modeled as discrete processes since they will be implemented onto digital electronics. An automatic control analysis allows to determine an upper bound for the delay between two successive input sample (periodicity, cadence) consumed by a control law from the environment, and an upper bound for the computation delay between an input sample and an output sample produced by a control law for the environment, which is the control produced in reaction to

this input (latency). If, when these bounds are not satisfied, control is no longer possible and leads to catastrophic consequences, these applications are called "strict or hard real-time" that we shall simply call "real-time" later on.

The second step takes place as soon as these parameters are set up, and consequently the control laws behave properly connected with its environment during the simulation process. It consists in implementing the discrete control laws onto digital electronic resources such as programmable components (processors) or non programmable components (ASIC[1] or FPGA[2] called later on "integrated circuit".) while satisfying real-time constraints that were determinated during the simulation process. The control laws interacting with the actual environment through sensors and actuators, such digital electronic system controls its environment by producing a reaction through actuator(s) processed from its internal state, and from the state of the environment consumed through sensor(s). It is in this sense that they are called "reactive systems" [1]. The control laws because of modularity considerations, or for performance reasons, are implemented onto several resources requiring a distributed heterogeneous architecture where hardware made of processors and integrated circuits, possibly of different types, are altogether interconnected through communications media, possibly of different types. We call such heterogeneous architectures "multicomponent" [2]. When digital electronic systems are embedded they must satisfy technological constraints, such as power consumption, weight, volume, memory, etc, leading to minimize hardware resources.

The complexity, not only of the functionalities that must be implemented, but also of the hardware architectures, and additionally the multiple constraints, imply to use methodologies when the development cycle time must be minimized from the high level specification until the successive prototypes which ultimately will become a commercial product. In order to avoid gaps between the different steps of the development cycle, we propose the AAA methodology based on a global mathematical framework. This allows to specify the application functionality as well as the hardware architecture with graph models, and the implementation of functionalities onto architectures in terms of graphs transformations. This approach has the benefit on the one hand to insure traceability and consistency between the different steps of the development cycle, and on the other hand to perform formal verifications and optimizations which decrease real-time tests. Also, it allows to perform automatic code generation.

All these benefits contribute to minimize the development cycle and provide a seamless flow which improves the design safety. The AAA methodology is supported by a system-level CAD software called SynDEx. Moreover, in order to allow functional simulation, the first step of the design process may be carried out by the scientific software package and the dynamic systems modeler and simulator software Scilab/Scicos. Its compiler is able to produce a source code compliant with the application functionality model of SynDEx. Therefore, by

[1] ASIC : Application Specific Integrated Circuit
[2] FPGA : Field Programmable Gate Array

coupling Scilab/Scicos and SynDEx the designer will use the best tool for both steps while insuring a consistent flow.

In section 2 we present the AAA methodology and its associated software SynDEx, then in section 3 we explain how coupling Scilab/Scicos with SynDEx provides a complete safe design flow, and finally in section 4 we give an example that illustrates this flow.

2 AAA/SynDEx

AAA/SynDEx provides a formal framework based on graph models and a system-level CAD software, on the one hand to specify the functionality of the application that we call "Algorithm", the distributed resources in terms of processors and/or specific integrated circuit, and communication media that we call "Architecture", and on the other hand to explore the design space solutions, i.e. the possible implementations of an algorithm onto an architecture. This is performed manually or automatically using optimization heuristics based on graph transformations. We call this optimization process "Adequation" leading to the name AAA of the proposed methodology. Exploration is mainly carried out through timing analyses and simulations whose results predict the real-time behavior of the application functions executed onto the different resources. This approach conforms to the typical hardware/software codesign. Finally, for the software part of the application, code is automatically generated as a dedicated real-time executive, or as a configuration file for a resident real-time operating system such as Osek, RTlinux, etc. For the hardware part of the application a synthetizable VHDL code is automatically generated.

The SynDEx software runs under Unix/Linux, Windows, and MacOS operating systems. It comes with a full documentation including a reference manual, a user manual, and a tutorial. It is downloadable free of charge under INRIA copyright at: www.syndex.org.

2.1 Algorithm

The algorithm model is an extension of the well known data-flow model from Dennis [3]. It is a directed acyclic hyper-graph (DAG) [4] that we call "conditioned factorized dependence graph", whose vertices are "operations" and hyper-edges are directed "data or control dependences" between operations. Hyper-edges are necessary in order to model data diffusion since a standard edge only relates a pair of operations. A data or control dependence models a data or control transfer between operations, and moreover imposes a "precedence dependence relation" between their executions. The set of the data and control dependences defines a partial order on the operations execution [5] called "potential operation-parallelism". Furthermore, each operation may be in turn described as a graph allowing a hierarchical specification of an algorithm. Therefore, a graph of operations is also an operation. Operations which are the leaves of the hierarchy are said "atomic" in the sense that it is not possible to distribute

each of them on more than one computation resource. The basic data-flow model was extended in three directions, firstly infinite (resp. finite) repetitions in order to take into account the reactive aspect of real-time systems (resp. "potential data-parallelism" similar to loop or iteration in imperative languages), secondly "state" when data dependence are necessary between repetitions introducing cycles which must be avoided by specific vertices called "delays" (similar to z^{-n} in automatic control), thirdly "conditioning" of an operation by a control dependence similar to conditional control structure in imperative languages. Delays combined with conditionings allow to specify FSM (Finite State Machine) necessary for specifying "mode changes", e.g. some control law is performed when the motor is the state "idle" whereas another one is performed when it is in the state "permanent". Repetition and conditioning are both based on hierarchy. Indeed, a repeated or "factorized graph of operations" is a hierarchical vertex specified with a "repetition factor" (factorization allows to display only one repetition). Similarly, a "conditioned graph of operations" is a hierarchical vertex containing several alternative operations, such that for each infinite repetition, only one of them is executed, depending on the value carried by the "conditioning input" of this hierarchical vertex. Finally, the proposed model is compliant with the synchronous language semantics [6], i.e. physical time is not taken into account. This means that it is assumed an operation produces its output events and consumes its inputs events simultaneously, and all the input events are simultaneously present. Thus, by transitivity of the partial order on the operations execution associated to the algorithm graph, an output event of the algorithm is obtained simultaneously with the arrival of the input event which trigger it. Each input or output carries an infinite sequence of events taking values, which is called a "signal". Here, the notion of event is general, i.e. signals may be periodic as well as aperiodic. The union of all the signals defines a "logical time" where physical time elapsing between events are not considered.

2.2 Architecture

The typical coarse-grain architecture models such as the PRAM (Parallel Random Access Machines) and the DRAM (Distributed Random Access Machines) [7] are not enough detailed for the optimizations we intend to perform. On the other hand the very fine grain RTL-like (Register Transfer Level) [8] models are too detailed. Thus, the model of multicomponent architecture is also a directed graph [9], whose vertices are of four types: "operator" (computation resource or sequencer of operations), "communicator" (communication resource or sequencer of communications, e.g. DMA), memory resource of type RAM (random access) or SAM (sequential access), "bus/mux/demux/(arbiter)" (choice resource or selection of data from or to a memory) possibly with arbiter (arbitration of memory accesses when the memory is shared by several operators), and whose edges are directed connections. Therefore, a processor is a graph composed of an operator, interconnected with memories (program and data) and communicators, through bus/mux/demux/(arbiter). It is similar for an integrated circuit but

without program memory. A "communication medium" is a linear graph composed of memories, communicators, bus/mux/demux/arbiters corresponding to a "route", i.e. a path in the architecture graph. Like for the algorithm model, the architecture model is hierarchical but specific rules must be carefully fulfilled, e.g. a hierarchical memory vertex may be specified with bus/mux/demux and memories (e.g. several banks), but not with operator. Thank to hierarchy, in order to simplify the specification a multiprocessor may be described as a bi-partite graph made of two types of vertices: processors and media. Although this model seems very basic, it is the result of several studies [10], in order to find the appropriate granularity allowing, on the one hand to provide accurate optimization results, and on the other hand to quickly obtain these results during the rapid prototyping phase. Data communications can be precisely modeled through shared memory or through message passing possibly using routes.

2.3 Adequation

Implementation An implementation of a given algorithm onto a given multi-component architecture corresponds to a distribution and a scheduling of, not only the algorithm operations onto the architecture operators, but also a distribution and a scheduling of the data transfers between operations [11].

The distribution consists in spatially allocating each operation of the algorithm graph to an operator of the architecture graph. The difference between programmable component and non programmable component lies in the fact that only a unique operation can be allocated to an integrated circuit whereas several operations can be allocated to a processor. This distribution leads to a partition of the operations set, in as much as sub-graphs that there are of operators. Then, for each operation two vertices called "alloc" for allocating program (resp. data) memory must be added, and each of them is allocated to a program (resp. data) RAM connected to the corresponding operator. Moreover, each "inter-operator" data transfer between two operations distributed onto two different operators, is distributed onto a route connecting these two operators. In order to actually perform this data transfer distribution, according to the element composing the route, as much as "communication operations" that there are of communicators, as much as "identity" vertices that there are of bus/mux/demux, and as much as "alloc" vertices for allocating data to communicate that there are of RAM and SAM, are created and inserted. Finally, communication operations, identity and alloc vertices are distributed onto the corresponding vertices of the architecture graph. All the alloc vertices, those for allocating data and program memories as well as those for allocating data to communicate, allow to determine the amount of memory necessary for each processor of the architecture.

The scheduling consists in transforming the partial order of the corresponding sub-graph of operations distributed onto an operator, in a total order. This "linearization of the partial order" is necessary because an operator is a sequential machine which executes sequentially the operations. This is a temporal allocation of the operations spatially allocated to an operator. Similarly, it also consists

in transforming the partial order of the corresponding sub-graph of communications operations distributed onto a communicator, in a total order. Actually, both schedulings amount to add new edges which are only precedence dependences rather than data or control dependences, to the initial algorithm graph. To summarize, an "implementation graph" is the result of the transformation of the algorithm graph (addition of new vertices and edges to the initial ones) according to the architecture graph.

Optimization It is necessary to choose among all the possible implementations a particular one for which the constraints are satisfied and possibly some criteria are optimized.

In the case of multiprocessor architecture the problem consisting in distributing and scheduling the algorithm onto the architecture such that the execution time of the algorithm is minimum, is known to be of NP-hard complexity [12]. This amounts to consider, in addition to precedences constraints specified through the algorithm graph model, one latency constraint between the first operation(s) (without predecessor) and the last operation(s) (without successor), equal to a unique periodicity constraint (cadence) for all the operations. We propose several heuristics based on the characterization of the operations (resp. communication operations) relatively to the operators (resp. communicators), e.g. execution durations of operations and data transfers, amount of memory, etc, in order to minimize the execution duration of the algorithm graph on the multiprocessor architecture, taking into account communications, possibly concurrent [11]. The characterization amounts to relate the logical time described by the interleaving of events with the physical time. "Greedy heuristics" are preferred because they are very fast [13] giving results in a time well suited to rapid prototyping of realistic industrial applications. Indeed, for this type of application the algorithm graph may have several thousands of vertices and the architecture graph may have several tens of vertices, leading to a very complex optimization problems that must be solved quickly as possible. We also extend these greedy heuristics to iterative versions [14] which are much slower, due to back-tracking, but give more precise results for the ultimate commercial product. We also propose a "simulated annealing" version of these heuristics which gives still more precise results but is actually very slow.

New applications in the automobile, avionic, or telecommunication domains, lead us to consider more complex constraints. In such applications it is not sufficient to consider the execution duration of the algorithm graph. It is also necessary to consider periodicity constraints for the operations, possibly different, and several latency constraints imposed possibly on any pair of operations (not only on input-output pair). Presently, there are only partial results for such situations in the multiprocessor case, and only few results in the monoprocessor case. Thus, we tackle this research area by interpreting the typical scheduling model given by Liu and Leyland [15] for the monoprocessor case, in our algorithm graph model. This leads to redefine the notion of task periodicity through the infinite repetition of any operation of an algorithm graph associated to an

integer defining the period of the repetition. Then, we extend the periodicity to an infinite repetition of an operations finitely repeated, thus generalizing the SDF (Synchronous Data-Flow) model [16] proposed in the software environment Ptolemy.

For simplicity reason, and because this is consistent with the application domains we are interested in that is to say strict real-time, we only consider that the real-time systems are non-preemptive, and that "strict periodicity" constraints are imposed on operations. In this case it is possible to we give a schedulability condition for graph of operations with precedence and periodicity constraints. We have formally defined the notion of latency which is more powerful [17], for the applications we are interested in, than the usual notion of "deadline" that does not allow to impose directly a timing constraint on a pair of operations, connected by at least one path, like it is necessary for "end-to-end constraints". Moreover, notice that when two deadlines are used to impose a latency this method gives an over-constrained solution compared to the latency method we propose. We give a schedulability condition for graph of operations with precedence and latency constraints. Then, by combining both previous results we give a schedulability condition for graph of operations with precedence, periodicity and latency constraints, using an important result which gives a relation between periodicity and latency. We also give an optimal scheduling algorithm in the sense that if there is a schedule the algorithm will find it [17].

Thanks to these results obtained in the monoprocessor case, we propose extensions of the heuristics for one latency constraint equal to a unique periodicity constraint, in order to solve the distribution and scheduling problem for graph of operations with precedence, periodicity and latency constraints in the multiprocessor case [18].

2.4 Code Generation

As soon as an implementation is chosen among all the possible ones, it is straightforward to automatically generate executable code through an ultimate graphs transformation leading to a distributed real-time executive.

For a processor the operator (resp. each communicator) has to execute the sequence of operations (resp. communication operations) allocated to it. For an integrated circuit the operator has to execute the unique operation allocated to it. Then, this graph is translated in an "executive graph" [9] where new vertices and edges are added in order to manage the infinite and finite loops, the conditionings, the inter-operator data dependences corresponding to "read" and "write" when the communication medium is a RAM, or to "send" and "receive" when the communication medium is a SAM. Specific vertices, called "pre" and "suc", which manage semaphores, are added to each read, write, send and receive vertices in order to synchronize the execution of operations and of communication operations when they must share, in mutual exclusion, the same sequencer as well as the same data.

These synchronizations insure that the real-time execution will satisfy the partial order specified in the algorithm. Executives generation is proved to be

dead-lock free [10] maintaining the properties, in terms of events ordering, shown thanks to the synchronous language semantics. This executive graph is directly transformed in a macro-code [9] which is independent of the processor. This macro-code is macro-processed with "executive kernels" libraries which are dependent of the processors and of the communication media, in order to produce as much as source codes that there are of processors. Each library is written in the best adapted language regarding the processors and the media, e.g. assembler or high level language like C. Finally, each source code produced by macro-processing is compiled in order to obtain distributed executable code satisfying the real-time constraints.

3 Scilab/Scicos Coupled with SynDEx

The algorithm graph of SynDEx, which corresponds to the specification of the application functionality, may be described using its graphical user interface as for the architecture graph, or may be imported into the software SynDEx as a file with extension .sdx according to a syntax defined in the document: www.syndex.org/v6/grammar.pdf. The software Scilab/Scicos[3] is a scientific package and a dynamic systems modeler and simulator whose compiler can produce a file of this type. Thus, a block diagram performed with Scilab/Scicos is translated in a .sdx file that can be directly imported in SynDEx, and then an adequation can be performed with any architecture specified also with SynDEx.

Although both models seem close syntactically speaking, in terms of semantic the block diagram of Scilab/Scicos is quite different of the conditioned factorized dependence graph of AAA/SynDEx. Indeed, contrary to the AAA/SynDEx model the Scilab/Scicos model is not purely data-flow. It mixes control, materialized by activation input signals on the top, and output signals on the bottom of a block, and data-flow materialized by input signals on the left, and output signals on the right of a block. Activation signals are produced by "if-then-else" or "select" blocks. They control the execution of blocks. Moreover, data signals are not purely data-flow in the sense that data are remanent, which is not the case in pure data-flow. So, a complex translation is necessary to transform control/data-flow block diagram of Scilab/Scicos in a conditioned factorized dependence graph of AAA/SynDEx where the control is included in the data-flow leading to a model which is purely data-flow. Actually, the control in the AAA/SynDEx model is a specific hierarchical conditioning vertex, only one of its sub-graphs will be executed depending on the value of its specific conditioning input. Also, the data are not remanent that is to say a value produced by a vertex of an algorithm graph is lost at the end of the execution of the algorithm graph which is infinitely repeated. It will be produced again during the next repetition or logical instant.

The translation is composed of two steps. First, each imbrication of "if-then-else" or "select" blocks which produces activation signals for the other blocks determines the number, and the hierarchy of conditioning vertices in

[3] www.scilab.org

AAA/SynDEx. Second, each time a data must be memorized because it is not used in a conditional branch, that is automatically done in a Scilab/Scicos block diagram because of remanence, it is necessary to use a delay vertex in AAA/SynDEx. This is complicated because when a data must be memorized it is necessary to follow the the control due to imbricated if-then-else blocks transformed in a hierarchy of conditioning vertices in AAA/SynDEx.

4 Example

Figure 1 shows the principles of Scilab/Scicos coupled with SynDEx. The presented example is a manual driving application with joystick for the CyCab. It is an intelligent and modular electric vehicle designed at INRIA Rocquencourt by the IMARA team[4] and industrialized by Robosoft[5]. On the top of the figure Scilab/Scicos allows to model the CyCab, including the process to control (driver, ground, wheels, suspensions, motor, etc) and the controller (combination of several control laws). This latter, depending on the position of the joystick, modifies the wheels direction, increases or decreases the speed of the motor, or activates the brakes. The complete model (process and controller) is simulated and the parameters of the controller are adjusted. Then, the block diagram representing the controller is discretized if this model was continuous, and compiled with the SynDEx option in order to produce a .sdx file. On the bottom of the figure SynDEx allows to import this file as an algorithm graph. Then, the distributed architecture of the CyCab, made of four microcontrollers MPC555 from Motorola and one embedded PC altogether interconnected through a CAN bus, must be specified with SynDEx. Now, it is possible to perform an adequation, and generate an executable code for each processor, using the executive kernels corresponding to the MPC555, the i80386, and the CAN bus. These codes are loaded an run inside the processors of the CyCab.

Figure 2 shows the graphical user interface of SynDEx used to implement the discrete controller onto the distributed architecture. On the left top of the figure a window depicts the algorithm graph, and because it is a hierarchical specification a smaller window (left bottom) depicts in turn a sub-graph of the algorithm graph. On the right top of the figure a window depicts the architecture graph (four MPC555 and an embedded PC connected with a CAN bus). Finally, on the left bottom of the figure a windows depicts the result of the adequation applied to the algorithm and the architecture graphs. This is a timing simulation window describing how the algorithm is distributed and scheduled onto the architecture by the optimization heuristics of SynDEx. It includes one column for each processor and communication medium, describing the distribution (spatial allocation) and the scheduling (temporal allocation) of operations onto processors, and of inter-processor communications onto media. Time flows from top to bottom, the height of each box is proportional to the execution duration of the corresponding operation (execution durations are given/measured by the user

[4] www-rocq.inria.fr/imara
[5] www.robosoft.fr

for each available pair of operation/processor or data/medium). Consequently, the height of this timing windows gives the execution time of the distributed application that the user can compare to the timing constraints. Usually, several adequations with different variants of the algorithm and the architecture (implementation exploration), are necessary to obtain the implementation satisfying the real-time constraints, then the user will ultimately generate the executable code.

5 Conclusion

We presented the AAA methodology and its associated software SynDEx providing a seamless design flow for the optimized implementation of distributed embedded real-time applications. This software coupled with the scientific software package and the dynamic systems modeler and simulator software Scilab/Scicos allows to use the best tool for both steps of the design process while insuring a consistent flow.

We have work in progress in order to extend the optimization techniques, which presently are mainly static, to more dynamic schemes to better support aperiodic event and dynamic creation of functions, and in order to provide automatic hardware/software partitioning in the codesign process.

References

1. Albert Benveniste and Grard Berry. The synchronous approach to reactive and real-time systems. *Proceedings of the IEEE*, 79(9):1270–1282, September 1991.
2. Y. Sorel. Real-time embedded image processing applications using the aaa methodology. In *Proceedings of IEEE International Conference on Image Processing, ICIP'96*, Lausanne, Switzerland, September 1996.
3. J.B. Dennis. First version of a dataflow procedure language. In *Lecture Notes in Computer Sci.*, volume 19, pages 362–376. Springer-Verlag, 1974.
4. R. Balakrisnan and K. Ranganathan. *A Textbook of Graph Theory*. Springer, 2000.
5. V. Pratt. Modeling concurrency with partial orders. *International Journal of Parallel Programming*, 15(1), 1986.
6. Nicolas Halbwachs. *Synchronous programming of reactive systems*. Kluwer Academic Publishers, Dordrecht Boston, 1993.
7. A.Y. Zomaya. *Parallel and distributed computing handbook*. McGraw-Hill, 1996.
8. C.A. Mead and L.A. Conway. *Introduction to VLSI systems*. Addison-Wesley, 1980.
9. T. Grandpierre and Y. Sorel. From algorithm and architecture specification to automatic generation of distributed real-time executives: a seamless flow of graphs transformations. In *Proceedings of First ACM and IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE'03*, Mont Saint-Michel, France, June 2003.
10. T. Grandpierre. *Modélisation d'architectures parallèles hétérogènes pour la génération automatique d'exécutifs distribués temps réel optimisés*. PhD thesis, Université de Paris Sud, Spécialité électronique, 30/11/2000.

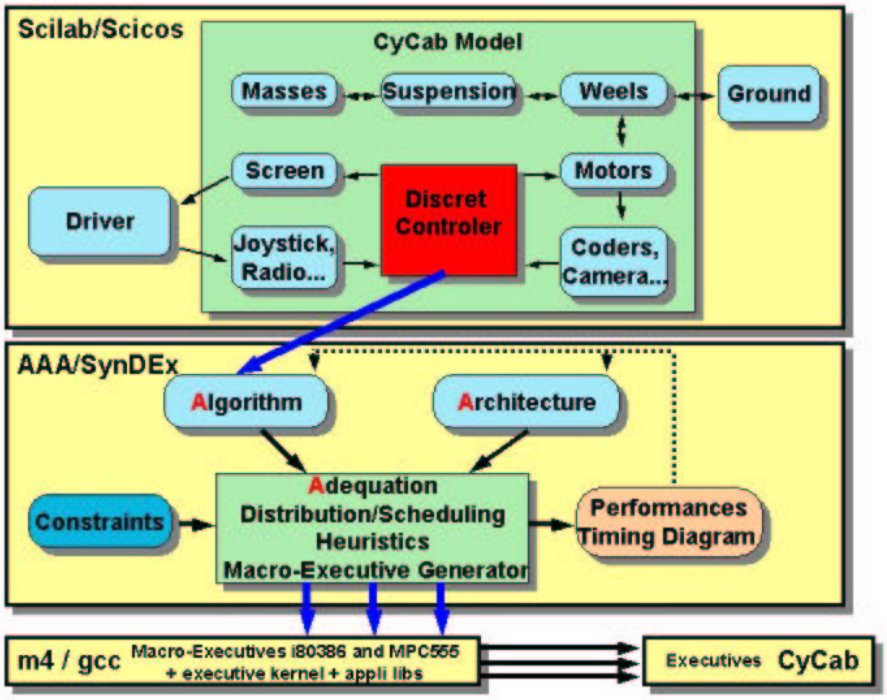

Fig. 1. Scilab/Scicos coupled with AAA/XSSynDEx

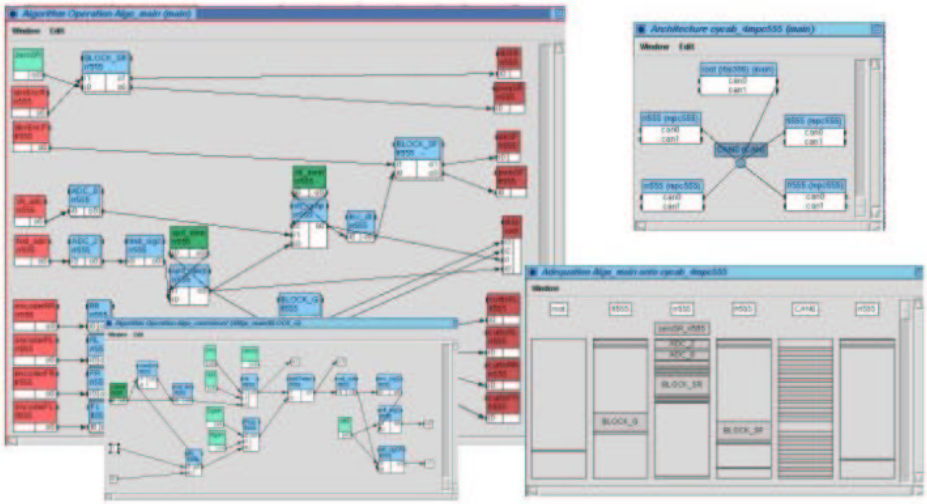

Fig. 2. SynDEx GUI used to design the manual driving application for the CyCab

11. T. Grandpierre, C. Lavarenne, and Y. Sorel. Optimized rapid prototyping for real-time embedded heterogeneous multiprocessors. In *Proceedings of 7th International Workshop on Hardware/Software Co-Design, CODES'99*, Rome, Italy, May 1999.
12. J. Leung and Whitehead J. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation(4)*, 1982.
13. Zhen Liu and Christophe Corroyer. Effectiveness of heuristics and simulated annealing for the scheduling of concurrent task. an empirical comparison. In *PARLE'93, 5th international PARLE conference, June 14-17*, pages 452–463, Munich, Germany, November 1993.
14. A. Vicard and Y. Sorel. Formalization and static optimization for parallel implementations. In *Proceedings of Workshop on Distributed and Parallel Systems, DAPSYS'98*, Budapest, Hungary, September 1998.
15. C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 1973.
16. E.A. Lee and Messerschmitt D.G. Synchronous data flow. *Proceedings of the IEEE*, 75, no. 9, 1987.
17. L. Cucu, R. Kocik, and Y. Sorel. Real-time scheduling for systems with precedence, periodicity and latency constraints. In *Proceedings of 10th Real-Time Systems Conference, RTS'02*, Paris, France, March 2002.
18. L. Cucu and Y. Sorel. Non-preemptive multiprocessor scheduling for strict periodic systems with precedence constraints. In *Proceedings of 23rd Annual Workshop of the UK Planning and Scheduling Special Interest Group, PlanSIG'04*, December 2004.