A methodology to design and prototype optimized embedded robotic systems

Rémy KOCIK, Yves SOREL
INRIA Rocquencourt - Domaine de Voluceau BP105
78153 Le Chesnay CEDEX - France
Tel: 33-(1) 39 63 51 75 - Fax: 33-(1) 39 63 57 86
email: remy.kocik@inria.fr, yves.sorel@inria.fr

ABSTRACT

We have designed a new electric robotic vehicle controlled by a distributed embedded computer system based on the CAN bus, providing features such as, secured manual driving or autonomous driving, and route planning. These functionalities involve control, image and signal processing algorithms executed under real-time constraints. In order to provide a safe software design and to reduce the development cycle time of such complex systems, we use a methodology called A^3 (Algorithm Architecture Adequation). It is based on graphs and partial order, to modelize the algorithm, the distributed architecture (microprocessors, specific integrated circuits, connected by a network) as well as the implementation of the algorithm on the architecture. The Adequation, based on heuristics, consists in finding the best implementation which satisfies the real-time constraints and minimizes the hardware resources. This allows to automatically generate dedicated executives with dead-lock free communication support.

CONTEXT

The microprocessor power increase as well as the electronic components miniaturization, allow computer based systems to be more and more powerful. Hugely using these techniques, robotics take naturally advantages of this evolution. Thus, the designers are encouraged to build more elaborate systems able to perform new complex tasks. The integration of these functionalities leads to increase software and hardware level of complexity.

Indeed, the first way to build high-performance systems is to better take into account the robot environment. It turns out that the systems involve sophisticated devices like camera, sonar, magnetic or ultrasonic sensors. In addition to these exteroceptive sensors, the robots must include proprioceptive ones in order to know their internal state and to ensure feedback in the actuator control loop. They can be as different as incremental optical encoders, accelerometers or strain gauges. To drive efficiently this different devices, the computer must also perform lot of low level processing to extract significant informations from the sensors and to carry out low level control on the actuators. These features involve a lot of computing (for example the processing of images from the camera) and/or a high input rate (incremental encoder position derivation or motor low control. . .).

On the other hand, to ensure a secure and autonomous behaviour to the robot, high level processing is necessary to implement control laws, states machine, data fusion, decision making, planning, neuronal networks, expert systems and data base. Finally the robot must communicate with the user through a graphical user interface.

These embedded robotic systems are reactive systems [5][12]. They must also cope with severe real-time constraints: the robot must react to external stimuli by triggering computations in order to generate output reactions within bounded duration (response time).

As well to satisfy these real-time constraints (small response time and high input rate), as to take into account the distributed nature of the resources (sensor/actuator, computation, memory) inherent to these robotic systems, high performance heterogeneous distributed computer based systems are needed. Specific components like ASICs, Digital Signal Processors and micro-controllers are used to perform efficiently signal and image processing and devices control, while CISC and RISC microprocessors perform high level tasks. Implementing the distributed application on a heterogeneous network while verifying all these constraints is difficult, and generally generate complex programs hard to test and debug. This explains the success of new dedicated robotics high level programming research software tools [10] [4].

They generally provide a graphical interface to help the user to specify and verify the application and are also able to simulate the beaviour of the robot. Afterwards, code can be generated automatically. An important effort has been done on these tools to make easier the specification and to guarantee that the generated code is conform. This generated code usually relies on traditionnal real-time multi-thread executives such as VXWORKS,OS9. . . running on VME centralized architectures. Thus, the user can quickly design safe complex applications but these tools don't allow to take into account distributed architectures and severe cost constraints met in some embedded applications.

To solve these problems we present here a new methodology and the associated tools. It allows high level verification, safe programming, automatic distribution and generation of optimized code while minimizing architecture resources with minimal development cycle.

A^3 METHODOLOGY

A^3 means Algorithm Architecture Adequation. The goal of the methodology is to find out an optimized implementation of an application algorithm on an architecture, while satisfying constraints [11]. ”Adequation” is a French word meaning an efficient matching. Note that it is different from the English word ”adequacy” which involves a sufficient matching. A^3 is based on graphs models to exhibit both the potential parallelism of the algorithm and the available parallelism of the multicomponent. The implementation is formalized in terms of graphs transformations.

Algorithm Specification And Verification

The classical notion of algorithm, given for example by Turing [13], defining a finite total order on the execution of a finite multi-set of operations, is here extended to partial order. However, this partial order is different from the one obtained by Hoare’s Communicating Sequential Processes approach [6]. The algorithm is modelized here by a conditioned data-flow graph, that is, a folded dependency graph presenting a pattern indefinitely repeated. This dependency graph describes data-dependency relations (edges) between operations (vertices). A data-dependency is actually a data transfer from a producer operation to a consumer operation. This involves a partial order on the execution of the operations, called potential parallelism. Potential means that this parallelism will be exploited only if parallel hardware resources are available. The execution of each graph pattern of the dependency graph is triggered when an input event, coming from the environment, is received by operations without predecessor. The output events are sent to the environment by operations without successor. Moreover, a dependency graph may be conditioned, that is, a part of this graph may not be executed. Specific conditioning vertices, with two input and one output, do not produce data when their conditioning input, which must be of boolean type, carries a false value. In this case, by transitivity all the dependent vertices will not be executed. Otherwise, when both input are present, but the conditioning one is true, the output takes the value of the other input. At this point, it is assumed that the result produced by an operation is simultaneous with the input which triggers it. This allows to make verifications on this specification in terms of input output events ordering. Note carefully here, that operations and data transfers durations are not taken into account. It turns out a logical time, which instants correspond to the input output events interleaving. This algorithm model has the semantics of the synchronous languages and more specifically, of the synchronous data-flow language SIGNAL [8].

Multicomponent Specifications

The implementation consists in distributing and scheduling the algorithm data-flow graph on the multicomponent taking into account real-time constraints. The operations and the data transfers durations are now considered, and therefore exploited to optimize the implementation, that is, to satisfy real-time constraints and to minimize the resources. In order to improve the usual approach based on a coarse model of the architecture like PRAM or DRAM [2], the multicomponent architecture is modelized more accurately. It is an hyper-graph which vertices are components, and which hyper-edges are communication media. A component may be in turn described in terms of an hyper-graph, leading to a hierarchical decomposition of the hardware architecture.

The smallest atomic component is a finite state machine. Two types of components are distinguished. An operator sequences operations on data, read from/written to communication media. Each operation of the algorithm graph is an indivisible sequence of computations. A transformator sequences data transfers between communication media (DMA, serial/parallel ...). Each transfer is an indivisible sequence of reads on one communication medium, interleaved with writes on another communication medium, in a ratio depending on the relative widths (number of parallel data wires) of the two communication media. A medium hyper-edge encompasses the wires used to support the communication, the finite state machine which arbitrates and synchronizes accesses to the wires, and some memory. Random access memory allows asynchronous communication, whereas sequential access memory (FIFO) implies synchronous communication. It is worthy to notice here, that it is very important during the optimization, to take into account carefully communications which support data transfers, especially if they are routed. Moreover, arbitrations must not be neglected because they introduce also delays.

The architecture hyper-graph is labeled by characteristics related to hardware considerations. Because, afterwards in this paper, only the reaction duration will be considered for the optimization, we focus here on the durations. However, other characteristics could be taken into account, such as, amount of necessary memory, power consumption .... Then, to each operator is associated the list of couples (operation, duration) it is able to perform. Similarly, to each transformator is associated the list of couples (data transfer, duration) it is able to perform. Furthermore, when several components share a resource (sequencer, memory ...) arbitration is necessary. Consequently, the durations associated to the operations assigned to the involved operators must be modified according to an interference matrix which modelizes a slowing down.

Adequation

The implementation is formalized in terms of transformations applied on the previously defined graphs. The distribution is a spatial allocation (assignation of several operations resp. data transfers, to an operator resp. transformator). It induce inter-operator communications, possibly routed when operators are not directly connected. The scheduling is a temporal allocation (ordering through the operator resp transformator sequencer, of the assigned operations resp. data transfers). Given an algorithm graph and a characterized architecture graph, the optimization consists in selecting among all the possible transformations, on the one hand the one which maintains the input output event ordering verified during the algorithm specification, and on the other hand the one which minimizes the reaction duration and the number of components. This problem cannot be solved optimally (NP complete), the selection is carried out by fast resources allocation heuristics based on list scheduling greedy algorithms [7] which give as better results as they rely on an accurate architecture model.

From a selected graphs transformation, it is possible to generate automatically an optimized distributed executive for the programmed part of the architecture. It is built from a library of architecture-dependent executive primitives composing the executive kernel. There is one executive kernel for each supported processor.

AN APPLICATION EXAMPLE: THE CyCab

Now we present how the A^3 methodology may help to the design and the realization of a new Semi-Autonomous Electrical Vehicle (CyCab) prototype. This system is a result of the Praxitèle project [9] based on a new public transport system concept: electrical vehicles in self-service. The aim of this project is to allow the user to borrow one of these individual vehicles, in specific stations designed to that effect. He is able to drive freely, but must give back the vehicle in one of these stations.

The CyCab has been designed to transport up to two persons in downtown areas, pedestrian malls, large industrial or amusement parks and airports, at a maximum of 30km/h speed.(Fig. 1)

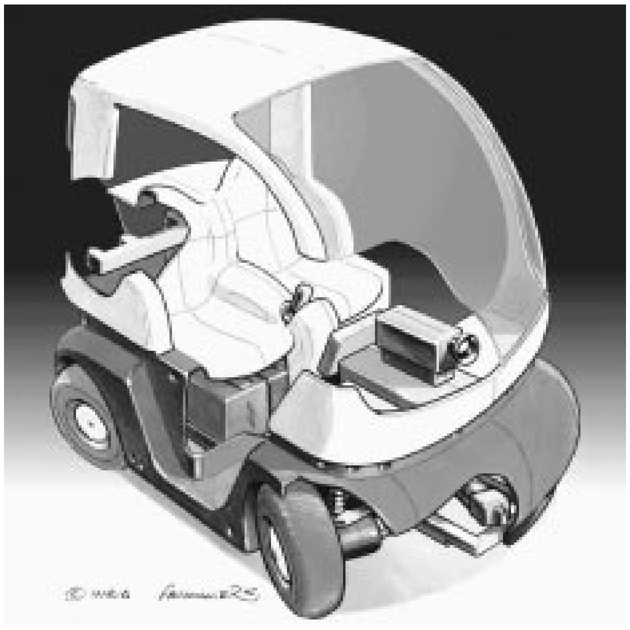


Figure 1: Semi Autonomous Electrical Vehicle: The CyCab

It offers two new functionalities, aided driving and autonomous driving. Aided driving rely on the use of a joystick and a finger touch screen. This joystick is connected to the computer, which controls the vehicle and then provides secure and easy driving: speed may be limited in curves and special areas. Everybody is able to drive this vehicle because it does not require specific skills from the driver. The finger touch screen allows the user to communicate with the system in order to get informations such as localization, vehicle autonomy, or for example the town museum list. To solve the problem of the vehicle fleet repartition through the stations, in order to satisfy customer demand, it is possible to form a ”car train” of empty vehicles with only one driver (in the front vehicle) [3]. Other autonomous driving modes, like radio-control or light markers guidance may be implemented.

This vehicle has been designed with mass production and public use constraints: low cost, low dimensions, robustness, easy maintenance. The whole design has been oriented in this way, from the mechanical to the computer based system.

Hardware Architecture

The mechanical is borrowed from a small electrical golf car frame, already produced in small series. The use of four identical wheel motor blocks, allows to reduce costs (by re-using identical parts) and volume (four small engines with small power controllers are easier to build and integrate than a big one with a high power controller). Consequently, the architecture is modular and the vehicle is easier to drive (the four wheels are propulsive and directive). The steering is made through an electrical jack mechanically linked to all the wheels.

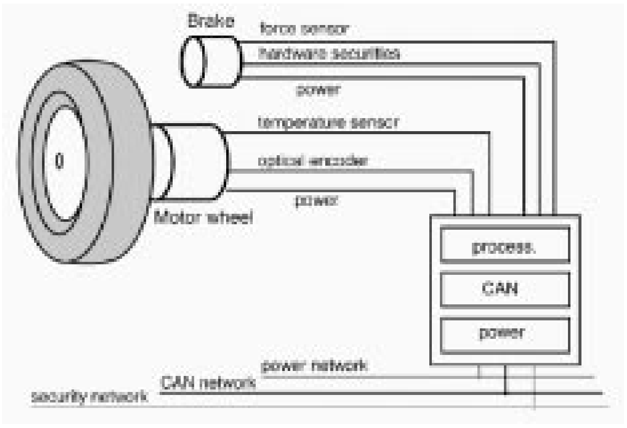


Figure 2: Node Architecture

Each wheel motor block has it’s own power amplifier, driven by a micro-controller(Fig 2). This intelligent node is constituted of three linked layers. The lowest one provides power to two motors. The second one allows the sensor acquisitions and the communications with the other nodes. The last one, made with a MC68332 drives the two others. The MC68332 is a micro-controller commercialized by Motorola. It is well suited for motor control. To generate PWM signals or to perform quadrature signals decoding, it integrates a programmable Time Process

Unit.

Each wheel node controls the drive engine and a brake motor, with all their associated sensors (optical encoder, y and brake torque measurement, temperature . . .). A fifth node is attached to the steering jack and the joystick.

The communications between the nodes are made through a CAN serial bus [1]. It has been designed specially for automotive applications and allows safe communications in disturbed environment, with a rate of 1Mbits/s. It carries messages up to 8 bytes length with 50 per cent control and arbitration bits overload.

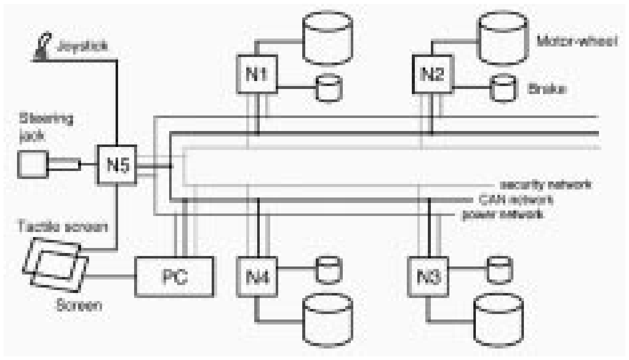


Figure 3: Network Architecture

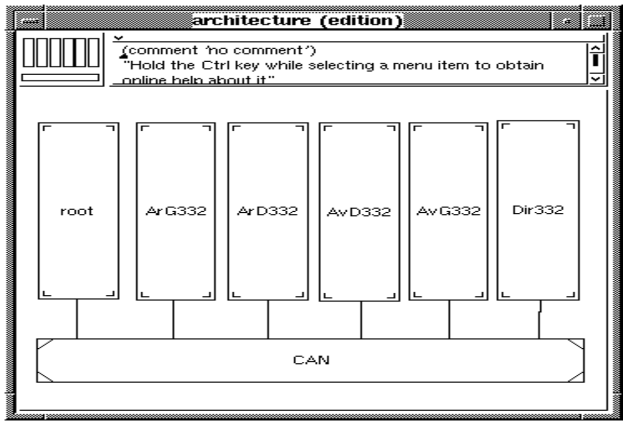
The network consists in 5 nodes and a PC board which drives the screen and the hard disk. (Fig. 3)

Implementation With SynDEx

We show here, on the “manual driving” mode implemented on the CyCab, the benefits of using the SynDEx[14] tool which support the A^3 methodology.

Using the graphical user interface of SynDEx, the first step consists in specifying the algorithm as a data-flow graph in order to exhibit the potential parallelism. Then the user describes the hardware architecture as an hypergraph. He may now ask for an Adequation, which carries out a distribution and a scheduling, displayed as a predicted diagram. At this time, the user may verify that his application satisfies the real-time constraints.

If the constraints are satisfied the user tends to optimize hardware resources by minimizing the components number, and by using cheaper components. On the contrary, if the application does not meet the constraints, he must redesign his algorithm, by choosing a smallest granularity allowing a better distribution. If this is not sufficient, he may add new components or use fastest ones. To help the user, SynDEx proposes an initial number of components, which is the ratio (maximum speedup) between the sum of all the operation durations and the critical path duration. The upper bound of this number is a good estimation of the minimum number of components to use. When the user is satisfied, he may ask SynDEx to generate a dead lock free executive for the real-time execution of the algorithm application on the multi-component architecture. A program, written in a macro-code independent of the microprocessor type, is generated for each component. The macro-code is built from the executive kernel which supports all the primitives provided by SynDEx: memory allocations, communications, tests, loops . . .

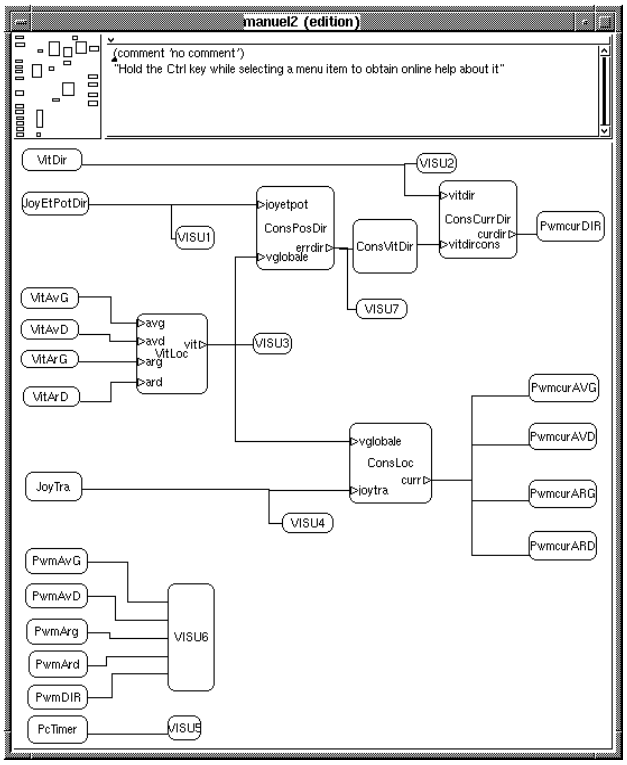


(comment 'no comment')
"Hold the Ctrl key while selecting a menu item to obtain online help about it"

Figure 4: Architecture Graph

The figure 4 shows a snapshot of the architecture graph of our application implemented using SynDEx . This actual hardware architecture, is composed of five MC68332 (`AvG332`, `AvD332`, `ArG332`, `ArD332`, `Dir332`) and one 486DXII66 (root) all linked through a CAN bus.

The figure 5 presents a snapshot of the data-flow graph of the “manual driving” application algorithm. The vertices are the operations to be executed on the data. The edges are data dependencies between operations. This graph is composed of 3 kinds of vertices: the input ones only produce data, the processing ones produce and consume data, and the output are the ones which only consume data.



(comment 'no comment')
"Hold the Ctrl key while selecting a menu item to obtain online help about it"

Figure 5: Algorithm Graph

Typically the input vertices are the sensors and the output vertices are the actuators. That is why they must be constrained to be assigned to the components attached to the sensors and actuators. The graph is cyclicly executed ; data flow from left to right. The execution time of one graph is the real-time loop period.

To figure out how to use SynDEx, we present the steering control part of our application. The figure 6 details how the control loop has been implemented in the algorithm graph.

The vertex `JoyEtPotDir` holds the wanted wheel angle (*joydir*) given by the lateral joystick position, and the steering jack position (*potdir*) given by a potentiometer. The other values necessary to the control, are the vehicle and the steering jack speed (resp. *v* and *vitdir*). The first one is computed by the vertex `VitLoc` which processes a mean of the 4 wheel speeds processed by `VitAvG`, `VitAvD`, `VitArG`, `VitArD`. The second one is given by `VitDir`.

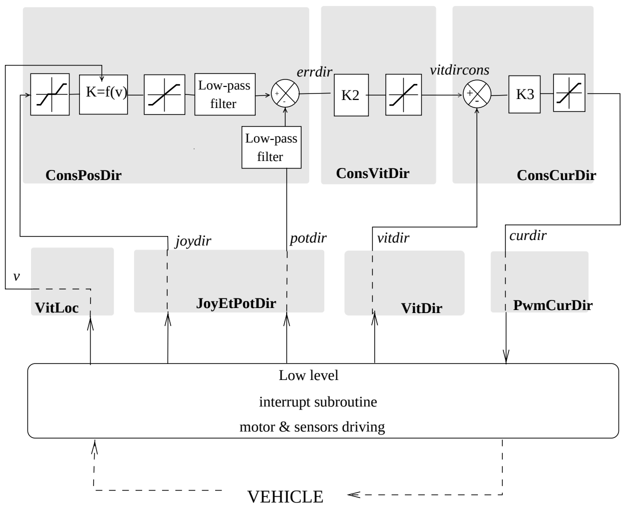


Figure 6: lateral control loop

The function named `ConsPosDir` processes the position error *errdir*. The *joydir* value is first saturated in order to bound the steering jack displacement. The value is then multiplied by a factor varying according to the vehicle speed, in order to provide a lateral acceleration boundary. Before to be subtracted, both the wanted and the actual position values are low-pass filtered to minimize the acquisition errors influence and to smooth the driving.

`ConsVitDir` provides a wanted jack speed (*vitdircons*) from the position error. It is a proportional filter saturated to the jack max speed displacement.

`ConsCurDir` processes the wanted current (*currdir* for `PwmCurDir` which performs the steering jack control current loop. *currdir* is the speed error (*vitdircons - vitdir*) multiplied by a constant.

Actually, `PwmCurDir`, `VitDir`, `JoyEtPotDir`, `VitAvG`, `VitAvD`, `VitArG`, `VitArD`, do not control directly the hardware. They send and receive data, to and from, a low level interruption subroutine which provides data acquisition, quadrature decoding, speed processing, current control loop and low level safety. The benefit is to allow low level processing to be executed at a fixed rate (for wheel speed processing) higher than the real-time loop one. In our application, the real-time period is 10ms when the low level control period is 1ms.

Note that in order to help the programmers, several vertices called ”VISU” allow to ”spy” the data, by displaying or saving them on the PC. This facility is very useful during the test and debug stage.

The figure 7 presents the predicted execution time given by SynDEx after the Adequation was completed. It shows only one execution of the graph. Each column represents the sequence of operations assigned to each processor, the Y-axis showing the time progress from top to bottom. The edges represent processor communications. More precisely, the origin of an edge is the date when the data is ready to be sent, and the ending of the edge is the date when this data is available on the destination processor.

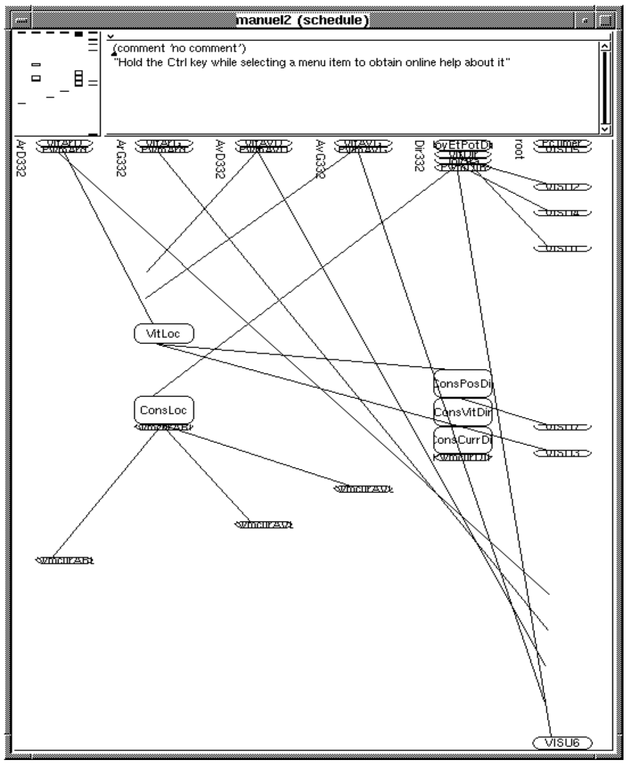

(comment 'no comment')
"Hold the Ctrl key while selecting a menu item to obtain online help about it"

Figure 7: Schedule

On each processor the optimized macro-code generated by SynDEx is expanded by the M4 (Gnu) macroprocessor to produce an assembly code. A makefile is also generated. It performs the compilation and the link, calling Gnu GCC, and loads executables on the processors through the CAN bus.

CONCLUSION

Like a lot of other research tools, the A^3 methodology provides designing facilities for robotic applications. Thus, the programmer may properly specify and verify the application. But contrary to these other tools, the user may here choose and size the hardware architecture, and optimize the application thanks to the SynDEx capabilities to predict the behaviour of an algorithm implemented on a specific architecture. Moreover, he may generate automatically dead lock free optimized distributed real-time executives. Then, the user is discharged of the low level communication tasks programming and debugging, and may focus on the algorithm specification and optimization. The automatically generated executives do not rely on expensive commercial executives and induce a minimal overhead. Thus, SynDEx can be used to handle minimal distributed heterogenous hardware architecture. This approach allows rapid optimized prototyping of robotic systems under industrial constraints.

References

[1] Michael Babb. New sensors have intelligence, will communicate. *Control Engineering*, pages 84–85, February 1994.

[2] M. Cosnard and A. Ferreira. On the real power of loosely coupled parallel architectures. *Parallel Processing Letters*, 1(2):103–112, 1991.

[3] Pascal Daviet and Michel Parent. Platooning technique for empty vehicles distribution in the praxitele project. In *Proceedings of the 4th IEEE Mediterranean Symposium on New Directions in Control and Automation*, Maleme, Krete, GREECE, June 1996.

[4] D. Gajski, F. Vahid, S. Narayan, and J. Gong. *Specification and Design of Embedded Systems*. Prentice-Hall, 1994.

[5] D. Harel and A. Pnueli. On the development of reactive systems. In Springer-Verlag, editor, *Logics and Models of Concurrent Systems*, volume 13 of *NATO ASI*, pages 477–498. New York, k. r. apt edition, 1985.

[6] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[7] C. Lavarenne and Y. Sorel. Performance optimization of multiprocessor real-time applications by graphs transformations. In *Proc. of the PARCO93 conference*, France, 1993.

[8] P. Leguernic, M. Leborgne, T. Gautier, and C. Lemaire. Programming real-time applications with signal. Research report, INRIA, June 1991. Research Report.

[9] M. Parent, E. Benejam-François, and N. Hafez. Praxitèle: a new public transport with self-service electric cars. In *ISATA Congress*, Florence, Italy, June 1996.

[10] Daniel Simon, Bernard Espiau, Eduardo Castillo, and Konstantinos Kapellos. Computer-aided design of a generic robot controller handling reactivity and real-time control issues. Research Report 1801, INRIA, November 1992.

[11] Yves Sorel. Massively parallel computing systems with real time constraints, the “algorithm architecture adequation”. In *Methodology Proc. of Massively Parallel Computing Systems Conference*, Italy, 1994.

[12] John A. Stankovic and Krithi Ramamritham. *Advances in Real-Time Systems*, chapter Introduction, pages 1–16.

[13] A.M. Turing. On computable numbers, with an application to the entscheindungs problem. In *Proc. London Math. Soc.*, 1936.

[14] SynDEx webpage at INRIA. `http://www-rocq.inria.fr/syndex/`.