Synchronous composition of discretized control actions: design, verification and implementation with ORCCAD

Daniel Simon, Roger Pissard-Gibollet, Konstantin Kapellos, Bernard Espiau

Synchronous Composition of Discretized Control Actions : Design, Verification and Implementation with ORCCAD

Daniel Simon, Roger Pissard-Gibollet, Konstantinos Kapellos and Bernard Espiau
INRIA Rhône-Alpes, ZIRST, 38330 MONTBONNOT ST MARTIN, FRANCE
e-mail: Daniel.Simon@inrialpes.fr

Abstract

Robotic systems are typical examples of hybrid systems where continuous time aspects, related to control laws, must be carefully merged with discrete-time aspects related to control switches and exception handling. These two aspects interact in real-time to ensure an efficient nominal behavior of the system together with safe and graceful degradation otherwise. In a mixed synchronous/asynchronous approach, ranging from user's requirements to run-time code, Orccad provides formalized control structures, the coordination of which is specified using the ESTEREL *synchronous language and formally verified using the* FC2TOOLS *package. CAD tools have been designed and integrated to help the users along the steps of programming, verification and implementation processes.*

1. Motivation

As particular examples of real-time embedded systems, robotic systems range from cooperative manipulators to autonomous vehicles. They have in common a continuously increasing complexity which makes more and more difficult the needed integration of issues raised by automatic control, sensor data processing and computer science areas. The goal of a control architecture is then to organize coherently all the involved subsystems so that the global system behaves in an efficient and reliable way to match the end-user's requirements.

Robotic systems belong to the class of hybrid reactive and real-time systems in which different methods and tools of programming and control are to be used. From the users and programmers' point of view, the specification of a robotic application must be modular, structured and accessible to users with different expertise, from the *control systems engineer* to the *end-user*. The ORCCAD[5] environment is aimed at providing such users with a set of coherent structures and tools to develop, validate and encode robotic applications in this framework[1].

As methods and tools to handle hybrid systems are not mature enough, the main interest of today users is in the specification of complex missions or applications in an easy and safe way. For that purpose it is necessary to define properly what are the activities he should handle and how to *compose* them in order to meet his requirements. Then adequate tools must be used to specify and implement these activities with respect to real-time and reliability constraints. The activity of the controller is twofold: the periodic calculation of control algorithms can be described by a data-flow graph. These tasks can be programmed using a classical general purpose language like C and handled at run time by a Real Time Operating System (RTOS). The control flow manages these calculation activities along the life of the application : in particular it is in charge of exception handling e.g. switching from nominal to degraded mode upon occurrence of a predefined event. This activity is relevant to discrete time reactive systems and thus can be specified and programmed with synchronous languages like ESTEREL [4].

This paper is organized as follows. In section 2the main control structures used in all our applications are shortly defined: some details are given about the basic defined entity, the *Robot Task* (RT), and describe how to compose them in order to construct the so-called *Robot Procedures* (RP). In the next section, design and verification issues of RTs using the ORCCAD toolbox are addressed. Section 4is devoted to the composition and verification of RTs into more complex RPs. Some details about the implementation of the controllers are given in section 5.The paper ends with a summary of lessons learned and further improvements.

[1] http://www.inrialpes.fr/iramr/pub/Orccad/

2. The ORCCAD methodology

Requirements: A complete robotic system includes a lot of various subsystems involving different fields of science and technology like automatic control, sensor data processing and computer science. The goal of the control architecture is to organize coherently all these subsystems so that the global system behaves in an efficient and reliable way to match the end-user's requirements.

Robotics primarily deals with physical devices like arms or vehicles. These devices are governed by the laws of physics and mechanics. Compared with pure computing systems they exhibit inertia and they cannot always be accurately modelled. Usually their behaviour can be described by differential equations where time is a continuous variable. Their state can be measured using sensors of various kind which themselves are not perfect. Control theory provides a large set of methods and algorithms to govern their basic behaviour through closed-loop control ensuring the respect of required performance and crucial properties like stability.

Robots of any type interact with their physical environment. Although this environment can be sensed by environment sensors like cameras or sonars, it is only partially known and can evolve through robot actions or external causes. Thus a robot will face different situations during the course of a mission and must react to perceived events by changing its behaviour according to corrective actions. These abrupt changes in the system's behaviour are relevant to the theory of Discrete Events Systems.

Besides the correctness of computations the efficiency and reliability of the system relies on many temporal constraints. The performance of control laws strongly depend on sampling rates and computing latencies. Corrective actions must be usually executed within a maximum delay to insure the mission success and the system's security.

Therefore robotic systems belongs to the class of hybrid reactive and real-time systems which need to use a lot of different methods and tools to be programmed and controlled. The ORCCAD environment is aimed to provide users with a set of coherent structures and tools to develop, validate and encode robotic applications in this framework.

The ORCCAD approach: In ORCCAD, two entities are defined in order to capture the aforementioned requirements. The *Robot-Task* (RT) models basic robotic actions where control aspects are predominants. For example, let us cite hybrid position/force control of a robot arm, visual servoing of a mobile robot following a wall or constant altitude survey of the sea floor by an underwater vehicle. The RT characterizes in a structured way continuous time closed loop control laws, along with their temporal features related to implementation and the management of associated events. These events are:

- preconditions, which can be associated with measurements and watchdogs
- exceptions of three types:
  - type 1 exceptions are locally processed in the RT, e.g. by tuning a parameter of the control law;
  - type 2 exceptions signal that the RT cannot run to completion and must be switched for a new one, chosen by the controller;
  - type 3 exceptions are fatal for the application and must, as far as possible, drive the system in a safe recovery state.
- postconditions are emitted when the RT successfully terminates.

For the mission designer this set of signals and associated behaviours represents the *external* view of the RT, hiding all specification and implementation details of the control laws (Figure 1). The characterization of the interface of a RT with its environment in a clear way, using typed input/output events, allows the composition of them in an easy way in order to construct more complex actions, the *Robot-Procedures* (RPs): briefly speaking, they specify in a structured way a logical and temporal arrangement of RTs in order to achieve an objective in a context dependent and reliable way, providing predefined corrective actions in the case of unsuccessful execution of RTs.

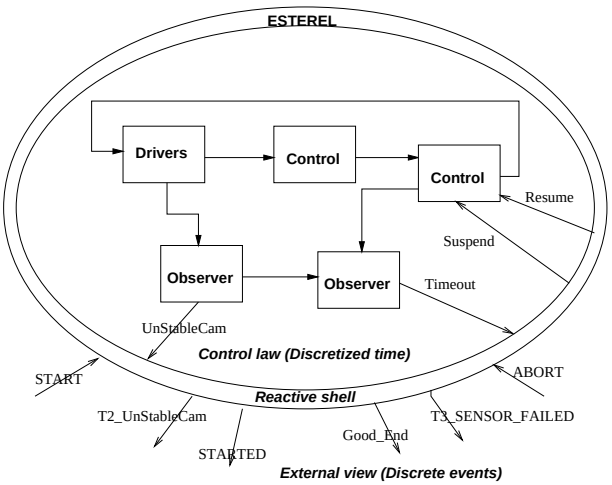


Figure 1: Encapsulation of the control law in a reactive shell

The *Robot-Procedure* (RP) paradigm is used to logically and hierarchically compose RTs and RPs in structures of increasing complexity. Usually, basic RPs are designed to fulfill a basic goal through several potential solutions, e.g, a mobile robot can follow a wall using predefined motion planning, visual servoing, or acoustic servoing according to sensory data availability and analysis. RPs design is hierarchical so that common structures and programming tools can be used from basic actions up to a full mission specification.

These well defined structures associated with synchronous composition, thanks to the use of the ESTEREL language [4], allows for the systematization and thus the automatization of the formal verification on the expected controller behaviour. This is also a key to design automatic code generators and partially automated verification. Formal definitions of RPs and RTs together with associated available formal verification methods may be found in [9].

Related work Remaining in the fields of robotics let us also cite [10] where data and control flows are gathered using the SIGNAL synchronous data-flow language. Using the same formalism for numerical calculations and tasking management allows for checking the temporal coherency of data in the whole application. However, if SIGNAL is well suited to specify signal processing algorithms, long computations like computing the explicit dynamics of a robot arm should be done by external functions written in a host language like C.

ControlShell [11] proposes an architecture quite similar to ORCCAD. At the level of control actions it allows for the construction of models of components such as PIDs or trajectory generators which can be gathered through a block-diagram oriented GUI. Event-driven aspects are handled by hand-made (and thus quite simple) Finite State Machines for which formal verification methods are not provided.

3. Specification of *Robot-Tasks*

The design of the robot controller begins with the design and test of basic actions when the necessary actions do not pre-exist in the Robot-task library. Let us now illustrate the design and validation process of a RT through an example of underwater robotics taken from [13], where the goal of the KeepStableCam RT consists of the stabilization of an underwater vehicle using visual servoing.

Modular Specification in Continuous Time In fact, it should be emphasized that the RT designer may select easily the adequate models and tuning parameters in ORCCAD, since they belong to some predefined classes in an object-oriented description of the control available through the graphical interface depicted by figure 2. The control algorithm is designed as a block-diagram, where elementary *algorithmic* modules are connected through input/output ports, e.g. `Servoing` and `Mapping` in Figure 2a. The value of parameters must be set to instantiate the design. It is also necessary to enter the localization of data processing codes (C files).

The particular *Physical-Resource* module (`ROV` here) provides a gateway towards the physical system through its Driver ports. A complete description of more complex RTs and of sensor-based tasks may be found in [13].

Time-constrained Specification The resulting specification defines the action from a continuous time point of view, i.e independently of time discretization and other implementation related aspects which are considered in a next design step. The passage from the above continuous time specification to a description taking into account implementation aspects is done by associating temporal properties to modules, i.e. sampling periods, duration of computations, communications and synchronizations between the processes.

Modules can be synchronized in several ways:

- explicitly periodic modules are connected to a system clock through a hidden input port ;
- an input port can be synchronized on an output port of another module, thus inheriting its period ;
- it can be synchronized on an external source, e.g. an interrupt coming from a vision system.

A Timed Event Graph model of the synchronization skeleton of the RT can be automatically built. Its analysis permits to conclude about dead-lock freedom and temporal coherency of the design [12]. Using the (max,+) algebra [3] is currently investigated to deeper analyze the temporal behaviour of the RT.

Specification of the reactive behaviour The logical behaviour of the RT is automatically encoded in ESTEREL through a graphical window (Figure 2c) (here we specify that the `UnStableCam` signal must be treated as a type 2 exception). Thanks to the strong typing of events and exceptions the code generator was proven to be correct and thus guarantees that crucial properties like safety and liveness are true [9]. Besides user's defined signals (pre and post-conditions, exceptions), the code generator builds a large ESTEREL file where hidden system signals are also declared. These

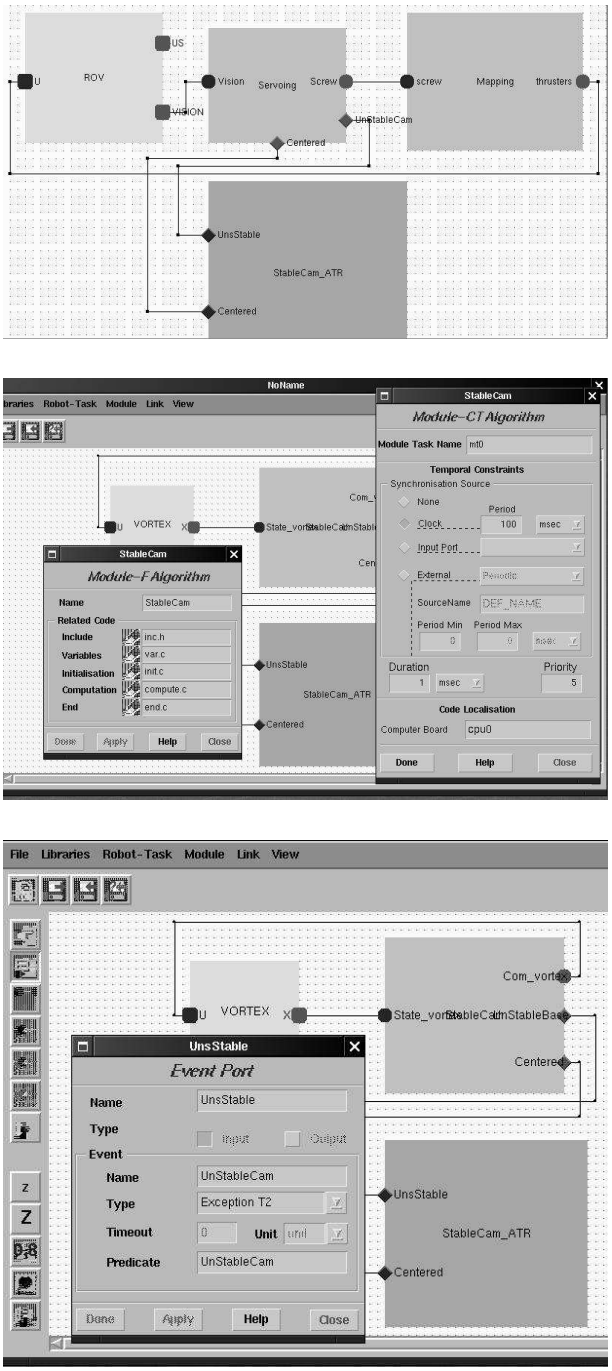


Figure 2: a)Block-diagram – b)Functional and temporal attributes of a Module – c) Specification of the reactive behaviour of a RT

signals will be used at run time to spawn, suspend or resume all the real-time threads necessary for the execution of the RT. Therefore, the control scientist in charge of the RT design has the advantage of synchronous programming without writing source code.

4. Design and Analysis of *Procedures*

After all necessary basic actions have been designed and validated, the user now wants to use them in more complex procedures to perform a useful underwater inspection mission [13]. Here is the detailed specification of the KeepStable procedure in order to enlighten the specification and analysis process proposed for basic actions composition.

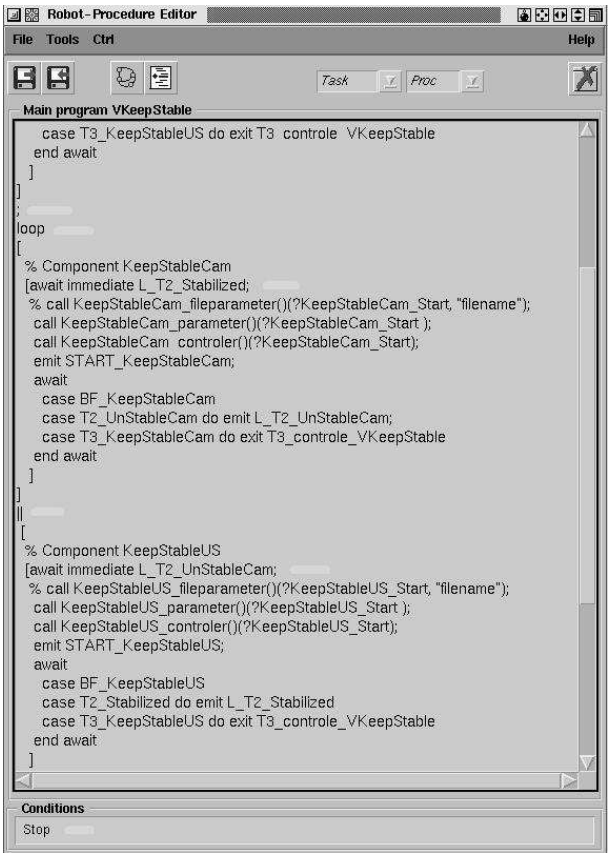
4.1. Procedure specification

The KeepStable procedure aims at stabilizing the underwater vehicle in spite of perturbations. The vehicle can be stabilized in two ways: it can be remotely locked on its target using either visual servoing or acoustic sensors. Here visual servoing (running the KeepStableCam RT) is considered as the nominal mode. However, if the camera loses the target, the controller must switch to sounders stabilization (degraded mode) running the KeepStableUS RT until being able to recover the vision tracking mode. This redundancy is useful to increase the safety and efficiency of the system and such a situation where several RTs are exceptions of each other is a typical structure in our procedures. Once again, the Esterel language is used to specify that actions are run in sequence or in parallel, or must preempt each other. Thanks to the structure of Orccad programs, the specification of this procedure can be written in two ways which both avoid the end-user to write himself the full source Esterel file:

- The Robot-Procedure Editor displays the external view of selected RTs and RPs. The user just has to add some statements to express, e.g., sequencing (;), parallelism (||), or escape mechanisms (trap-exit) to complete the specification (these additional statements are highlighted in Figure 3a). As the designer only access the external views, he cannot jeopardize the properties of the previously defined actions.
- Using the Maestro language [8] currently designed to target Esterel (Figure 3b). This language has been designed to be more "natural" for robotic practitioners. At compile time it expands in Esterel statements and also performs some preliminary verifications like liveness.

In this example the alternance between the two RTs is specified inside a parallel statement (PAR), in which each RT is guarded by a T2 exception emitted by the other RT. As these exceptions are mutually exclusive the two RTs cannot run simultaneously : anyway this property will be formally verified in the next section.

At compile time the control code of this procedure is translated into an automaton which output functions are automatically filled with calls to the underlying real-time operating system, thus alleviating the burden of the programmer while preserving formal verification capabilities.




```
do
  SEQ(do
        KeepStableUS
      until Stabilized
      ;
      loop
        PAR(when T2_Stabilized when T2_UnsStableCam
            do                 do
              KeepStableCam       KeepStableUS
            until UnStableCam   until Stabilized)
      end loop
       )
until Stop
```

Figure 3: Specification of a Procedure using: a) the GUI – b) the MaestRo language

Robotic applications are built incrementally by adding new RTs and RPs which behavior, e.g. synchronization, are encoded in the same way. The next example describes a RP used to synchronize the motions of the underwater vehicle and of its associated arm.

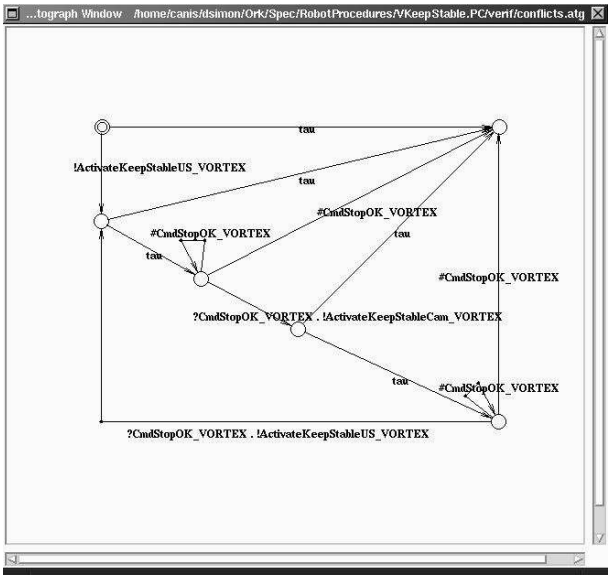
4.2. Logical Behavior Verification

First, the satisfaction of *crucial properties* can be checked. Concerning the *safety property* (any fatal exception must always be correctly handled) the process is as follows : knowing the user's specification defining the fatal exceptions and the associated processing, a criterion is automatically built to define an abstract action. The abstraction of the global procedure automaton with respect to this criterion is then computed. The absence of the "Error" action in the resulting automaton proves that the safety property is verified.

The *liveness property* (the RP always reaches its goal in a nominal execution) is proved in a similar way. The "Success" signal is emitted at the end of all successful achievements of a RP. The abstraction of the procedure automaton crossed with an adequate criterion built with this signal must be equivalent by bisimulation to a one state automaton with a single action, the "Success" one.

Conflicts detection : We are interested here in checking that during the RP evolution, there does not exist instants where two different RTs are competing for using a same resource of the system. Here, the physical resource controlled by the RTs (the underwater vehicle) is considered as well as the software resources used by the controllers (real-time tasks). For example, one wants to verify that the RTs KeepStableCamera and KeepStableUs never compete to apply different desired force inputs to the vehicle thrusters during all the RP evolution. The global automaton is reduced to the only interesting signals `Activate...` and `CmdStopOK...`. Thus by clicking on the conflicts button one can check that these two signals alternate during the RP life insuring that a control law can be started only after confirmation that the previous one is stopped and that the transition is as fast as possible to ensure the system's stability (Figure 4a).

Finally, the *conformity of the RP behaviour with respect to the requirements* is verified. For example, we want to certify that the vision-servoing and acoustic-servoing RTs can alternate for an arbitrary number of time during the life of the stabilization procedure. This property can be checked by observing



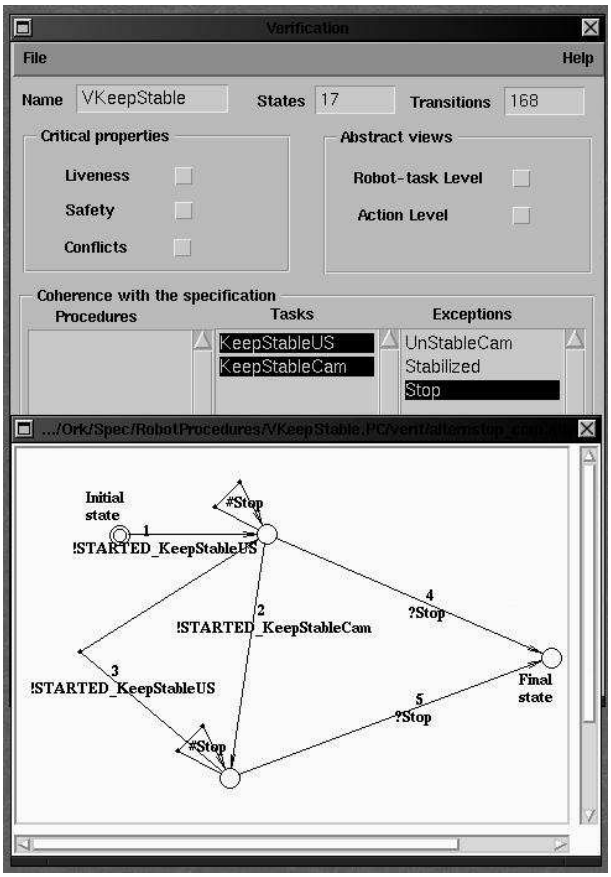


Figure 4: a)Checking for conflicts – b)An abstract view of KeepStable

the abstract view given by the ORCCAD graphical interface (figure 4b) : after minimization and abstraction through behavioral bisimulation ([6], the original automaton (which has 17 states and 168 transitions) becomes small enough to be visually analyzed. As expected, the procedure begins with the `KeepStableUS` RT (arc 1) : then, the two RTs altern in a loop, following arcs 2 and 3. The occurrence of the `Stop` signal is the only way to exit the procedure (arcs 4 and 5). The user just have to click on the name of signals which are relevant for the property to be checked to launch the verification process.

5. Implementation

5.1. Switching mechanism

Raising a type 2 exception or a "well-finished" signal request the embedding RP to stop the running RT and starting the next one, according to is own recovery program. Switching between two RTs must insure that:

- the control laws are not conflicting, i.e. the degrees of freedom of the system are all controlled, and only once.
- the configuration of the set of real-time tasks is valid before starting the new control law.
- the switching time must be minimum. Ideally, the delay during which the system is not controlled should be in the range of the sampling period.
- we must be able to check some formal properties about the switching mechanism.

The steps of the switching mechanism have been identified as follow:

- Reception of a type 2 exception or postcondition from the running RT, choice of the next one to start, starting of the transition phase
- Initialization of the new real-time tasks and instantiation of the communication ports. According to the load of the processor, it may be necessary to decrease the sampling period of the running RT (degraded mode allowing to keep the system under control)
- After fulfillment of the preconditions of the second RT, stopping the first control law and starting the second one. This delay must be minimized.
- Pause or destruction of the former real-time tasks

This mechanism is summarized in Figure 5 : the whole transition process usually takes 2 or 3 sampling periods.

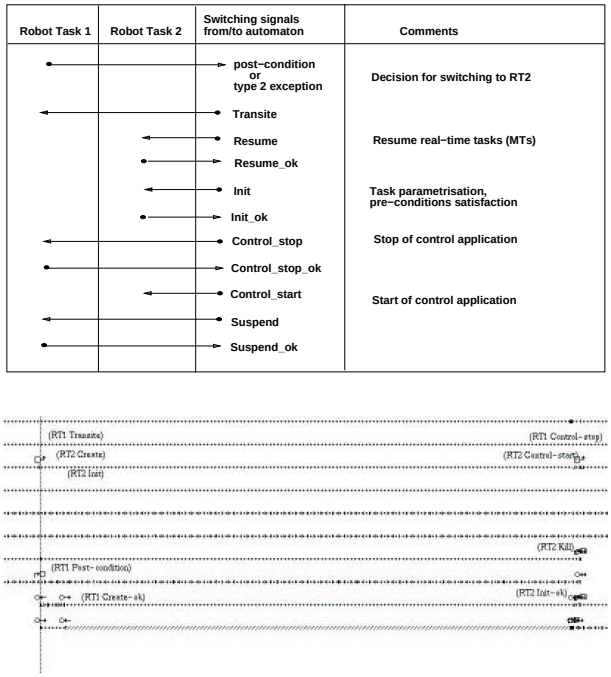


Figure 5: a) Outline of the switching mechanism b)Execution trace of the switching mechanism (3 sampling ticks)

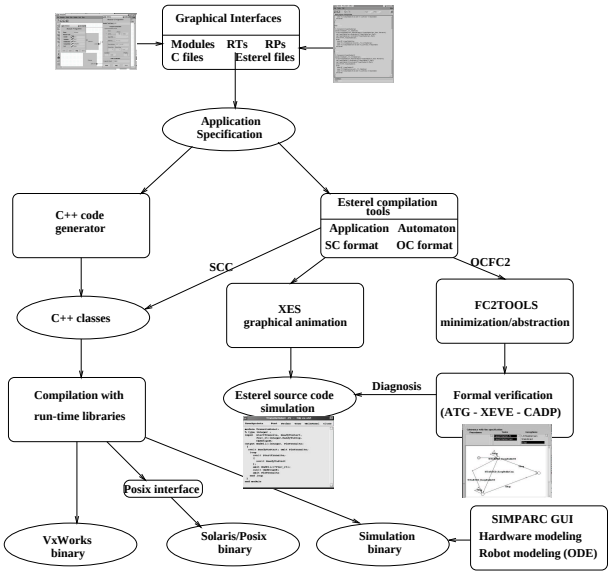


Figure 6: Code generation and associated tools

Figure 6 pictures the compilation process and integrated tools. The ORCCAD structures are instantiated as C++ classes provided by the kernel. For a single processor implementation, all ESTEREL files corresponding to the RTs and RPs logical behavior are gathered in parallel in a single file which is further compiled into various formats. The SC format (sorted boolean equations) is post-processed in C and encapsulated in a high priority real-time thread, while the FC2 format is the entry point for various model checkers. The application is finally compiled and linked with run-time libraries to produce down-loadable code. Currently ORCCAD targets VxWorks, Solaris and SIMPARC, an hybrid simulator able to simulate the closed-loop system including the temporal features of the controller ([1]).

5.2. Execution machine

At compile time, the ESTEREL source file is translated into an automaton encoded in C. Input and output functions are associated to allow the automaton to receive and emit signals. These functions are used to interface the synchronous reactive program with the asynchronous execution environment, i.e. the operating system. As the compiler knows nothing about the environment, the output functions are empty and must be filled by the user to make the program effective. Moreover, as ESTEREL is unable to do numerical computations, all calculations related to the execution of control algorithms are called as external procedures. Thus it is necessary to design an *execution machine* [2], the general structure of which is given by Figure 7a. Signals are collected to build the current event which is given to the automaton. Output actions calling extern calculation procedures must be carefully interfaced with the RTOS. Writing by hand this execution machine is highly tricky and error prone.

Thanks to the ORCCAD structures this machine can be automatically generated by the system (Figure 8). A main "system" task sets up the whole system and in particular generates the needed clocks used to trigger the periodic calculation modules. Real-time threads are made periodic by blocking their first input port on a semaphore which is released by clock ticks.

The automaton is the highest priority task: it is awakened by the occurrence of input signals related to pre-conditions, exceptions, and post-conditions. In reaction, it tells the RTOS what modules must be spawned, resumed or suspended. In particular, it carefully manages the smooth task switching mechanism described in Figure 5. Although this automaton is crucial for a safe and successful behavior of the application, it must be pointed out that it does quite nothing in term of computational activity : it spends most of time in active waiting for input events during the periodic execution of control algorithms managed by the RTOS. Typically, a transition of the automaton takes about 100 microseconds (on a MC68040 proces-

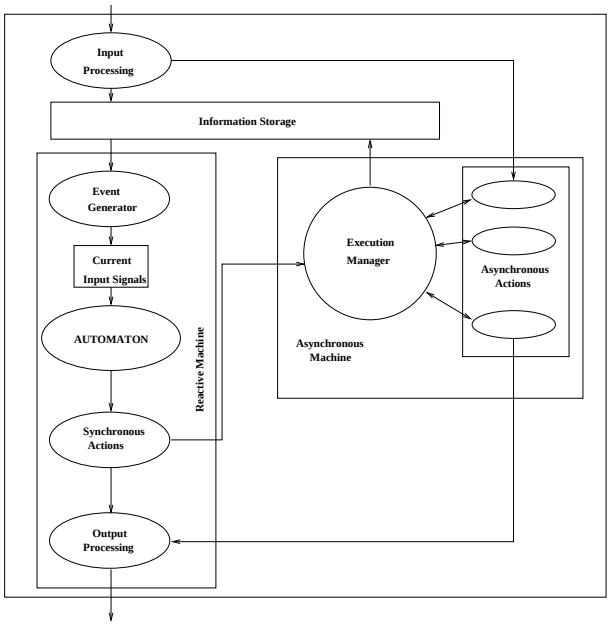


Figure 7: General structure of an execution machine for ESTEREL

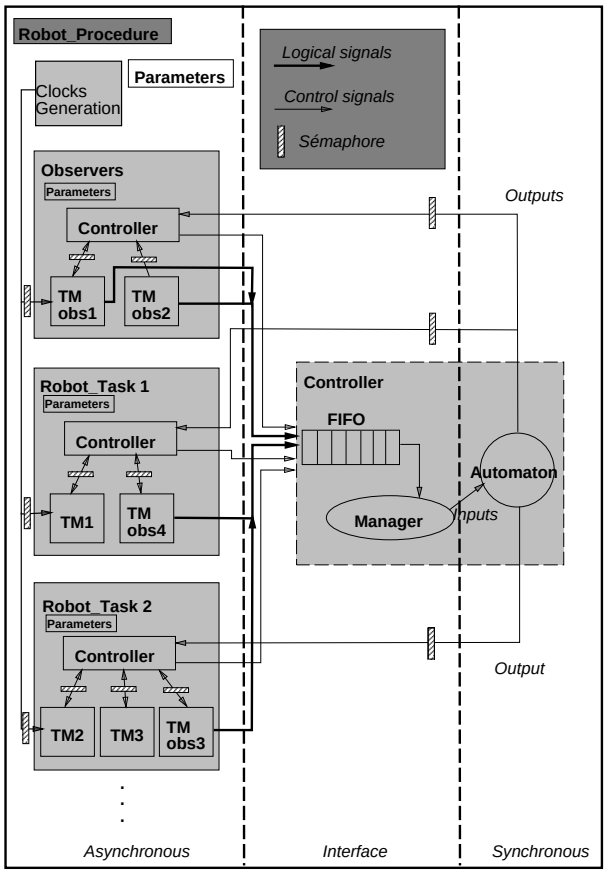


Figure 8: Implementation of the execution machine

sor) while the duration of robotic actions may range from seconds (e.g. manipulation tasks for a robot arm) to hours (e.g. mapping mission for an autonomous underwater vehicle).

6. Summary and work in progress

In this paper we have described the ORCCAD approach to formalize control structures in controllers for embedded systems such as robots. In particular the Robot-Task, where a discretized control law is encapsulated in a logical behaviour, can be considered as an hybrid structure constituting the frontier between continuous and discrete time aspects. Well formalizing these structures allows for formal verification of programmes and automatic down-loadable code generation.

In such control systems, the interleaving of events and activities in a real life mission justifies our hierarchical design and verification process, where we design complex actions from already validated ones lying at a lower level. The ESTEREL synchronous language was found to be efficient to specify and encode the coordination of actions, from the application level down to the low level logical management of real-time tasks. However, directly using the basic tools provided by synchronous design and programming remains difficult to use and leads to the design of more friendly interfaces:

- Writing synchronous programs requires some expertise and knowledge about this unusual kind of languages and easily leads to programming mistakes producing dead-locks or causality cycles. Starting from the specifications given through the ORCCAD GUI or by the MAESTRO language, the user is provided with control application oriented pre-defined structures. Thus automatic source code generation allows for taking advantage of synchronous programming with a moderate programming effort. This structuration is also a key for a partial automatization of the verification process.
- The ESTEREL compiler does not provide the interface functions with the environment. To avoid messy and error prone low level interfacing work, ORCCAD automatically generates the execution machine managing both the synchronous automaton and the asynchronous real-time threads. Thus the user may concentrate on application specification and validation rather than low level programming tricks.

References

[1] C. Astraudo, J.J. Borrelly, "Simulation of Multiprocessor Robot Controllers", *Proc. IEEE Int. Conf. on Robotics and Automation*, Nice, 1992.

[2] C. André, A. Ressouche and J.M. Tanzi: "Combining Special Purpose and General Purpose Languages in Real-Time Programming", *IEEE Workshop on Programming Languages for Real-Time Industrial Applications*, Madrid, Spain, december 1st, 1998.

[3] F. Baccelli, G. Gohen, G.J. Olsder, and J.-P. Quadrat. *Synchronization and Linearity.* Wiley, 1992.

[4] G. Berry "The Esterel v5 Language Primer ", available from http://www.inria.fr/meije/esterel/

[5] J.J. Borrelly, E. Coste-Manière, B. Espiau, K. Kapellos, R. Pissard-Gibollet, D. Simon and N. Turro "The ORCCAD architecture", *Int. Journal of Robotics Research*, vol. 17, no 4, pp 338–359, april 1998

[6] A. Bouali and R. de Simone: "Symbolic Bisimulation Minimisation", *Lecture Notes in Computer Science no 663, Computer Aided Verification*, pp 96–108, 1993.

[7] P. Caspi and A. Girault and D. Pilaud:"Automatic Distribution of Reactive Systems for Asynchronous Networks of Processors", *IEEE Transactions on Software Engineering*, vol. 25, no 3, may/june 1999.

[8] E. Coste-Manière and N. Turro "The MAESTRO Language and its Environment: Specification, Validation and Control of Robotic Missions", *10th IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp 836-841, Grenoble, 1997.

[9] K . Kapellos: *Environnement de programmation des applications robotiques réactives*, PhD dissertation, Ecole des Mines de Paris, Sophia Antipolis, France, november 1994.

[10] Marchand, E., E. Rutten, H. Marchand and F. Chaumette:" Specifying and verifying active vision-based robotic systems with the SIGNAL environment", *Int. Journal of Robotics Research*, vol 17, no 4, pp 418–432, april 1998.

[11] S. Schneider, V. Chen, G. Pardo-Castellote and H. Wang: "ControlShell: A Software Architecture for Complex Electromechanical Systems", *Int. Journal of Robotics Research*, vol. 17, no 4, pp 360–380, april 1998.

[12] D. Simon, E. Castillo and P. Freedman, "Design and Analysis of Synchronization for Real-Time Closed-Loop Control in Robotics", *IEEE Trans. on Control Systems Technology*, vol 6, no 4, july 1998, pp 445-461.

[13] D. Simon, K. Kapellos and B. Espiau: "Control Laws, Tasks and Procedures with ORCCAD: Application to the Control of an Underwater Arm", *Int. Journal of Systems Science*, vol. 17, no 10, pp 1081-1098, october 1998.