Synchronous Approach to Industrial Process Control

Charles ANDRE Marie-Agnès PERALDI

March 14, 2000

Laboratoire Informatique, Signaux, Systèmes (I3S)
Université de Nice-Sophia Antipolis / CNRS
41 bd Napoléon III
F 06041 Nice cédex, France
e-mail: andre@cma.cma.fr peraldi@mimosa.unice.fr

Abstract

This paper attempts at introducing synchronous languages in the design of automatic and manufacturing systems. These systems have reactive propreties and they are submitted to real-time constraints. We advocate the use of the synchronous language: "**Esterel**". Through a simple example, we show how this language can express hierarchical structuration of systems, control, fault detection, and fault recovery.

Keywords: *Industrial Process Control, Flexible Structure, Computer Control, Real-Time Software, Synchronous languages, Esterel.*

1 Introduction

Automatic and manufacturing systems are generally complex to design. Decomposition into layers is a classical approach used in order to overcome complexity. Information flows through the system in both upward and downward directions. The *control* is conveyed by descending information, whereas *monitoring* is associated with ascending information. Layers are implemented by cooperating modules.

A module repeatly responds to stimuli from adjacent modules, by maintaining an ongoing relationship with its neighbours; this is typically the behaviour of a *reactive system* [1]. Moreover, reactions are subject to externally defined timing constraints. The lower the level is, the harsher the constraints are. Clearly, modules belong to the class of *real-time reactive systems*.

The main difficulty in the design of real-time reactive systems is to integrate the temporal requirements and to build reliable systems. The use of programming languages is essential to cope with these problems. Task-oriented approaches rely on real-time operating systems or on real-time languages (including ADA). For large applications, the debugging and the maintenance become problematic. To face these issues, new approaches have emerged. The *synchronous approach* seems to be a most promising one. A special issue of the Proceedings of the IEEE [2] has been devoted to synchronous programming. Among synchronous programming languages, **Esterel** [3] seems to be suitable for programming control and monitoring. **Esterel** is imperative and allows high-level event-oriented specifications. This language is syntactically simple, specially dedicated to control, using few basic constructions. Last but not least, it is based on a simple and sound mathematical semantics.

In the first section of this paper, we present the general requirements and constraints of the systems we are interested in. In a second part we briefly present the **Esterel** language. The third section illustrates, with a simple and progressive example, the advantage of using **Esterel** for expressing the expected behaviour of automatic systems, including abnormal behaviour resulting from faults.

2 Automatic and Manufacturing Systems

For convenience, the management and the control of automatic and manufacturing systems are generally decomposed into abstraction levels. From the planification level downto the local control level, the system is organized in layers: Planning, Scheduling, Real-time scheduling, Coordination, Local control. We are mainly interested in the three lower levels which are subject to the harshest temporal constraints.

Within this hierarchical framework, a level is composed of several *modules*. A module communicates with its senior module and its junior modules. Control flows downwards whereas status and faults are propagated upwards. Communication between a module and its peers has to be defered to a common ancestor module. A senior module may request services from one of its successors. In the simplest systems, a module has a fully deterministic behaviour. When it receives an order from its direct predecessor, it generates a fixed sequence of actions to its junior modules.

More complex systems such as *Flexible Manufacturing Systems* (FMS for short) require more intelligent modules. They have to take into account upgoing information. In this case modules act as filters. They are able to detect faultly behaviour, to inhibit some actions and even to recover from some errors.

Functionally, this kind of module can be decomposed into three blocks (Fig.1):

- The **control block** which is in charge of generating expected reactions in a timely way,
- The **detection block** that watches undesirable or critical situations,
- The **decision block** which deals with recovery problems. When one error is detected, this block checks whether the failure can be corrected by the module or not. If not, the information is propagated to the upper levels.

Figure 1: Module Decomposition

A similar structure has been proposed in [4] for a hierarchical and modular control and monitoring of FMS. The approach advocated in [5], in order to improve the dependability of production systems, relies on the same principle. Of course, it becomes problematic to express the expected behaviour of such modules. Classical approaches like **Petri Nets** or **Grafcet** (Function charts for Control Systems) are not sufficiently expressive. The use of a programming language seems to be more suitable, especially synchronous programming languages like **Esterel** which has been designed for real-time control.

3 Synchronous Languages

3.1 Synchrony

The synchronous approach is based on a relatively small variety of concepts. The main idea is to consider *ideal systems* that produce their output at the same instant they receive their inputs. This is the *synchrony hypothesis.* Synchronous systems compose very well and turn out to be more easily analysable than asynchronous ones. Of course at the implementation level the evolutions are asynchronous and take time. The synchronous hypothesis must be checked at this level [6]. The *temporal correctness* cannot make abstraction of the actual implementation.

3.2 The Esterel synchronous language

Esterel [3, 7] is an imperative language dedicated to the programming of reactive application. The language is based on a formal mathematical semantics. The compilation of an **Esterel**'s program yields a finite and deterministic automaton, semantically equivalent to the program. Thanks to dedicated tools available in the software environment of **Esterel** [8], the *logical correctness* of the program can be carried out by finite state machine analyses.

The programming unit of **Esterel** is the *module*. The declarative part specifies the interface between the module and its environment, whereas the body of the module defines its expected behaviour in an imperative style. The language is composed of few basic instructions:

- **Classical constructors** like `loop...end` or `if...then...else...end`.
- **Communication**: supported by instantaneous signal broadcasting. Signals are generated by the `emit` statement.
- **Temporal statements**: dealing with logical multiform time provided by input signals. Primitives like `present...then...else...end` or `await` test the presence of a signal in the current instant.
- **Exception and watchdog**: The `trap...end` statement stands for exception treatment. The exception raising is done by an `exit`. Timeout and watchdog programming are expressed by the `do...watching` statement.
- **Parallelism**: modules can be composed in parallel. The parallel operator is $||$.

4 Example of Esterel program

This example shows the advantages of using a synchronous language like **Esterel** for designing automatic application. Programs are written with extra comments patched over the actual program. These alterations allow to point out, in a visual and concise way, some of the features of the language.

4.1 The system to be controlled

Recall that this system is deliberately a very simple one. It is sufficient to illustrate the more common situations met during the design of a system. This system is the heart of a watering equipment. It is composed of a tank, a pump to empty the tank, a valve to fill in, two level sensors (Low Level and High Level) and command inputs.

4.2 Normal running mode

In this mode the system is assumed to be running without underflow or overflow. The expected behaviour is expressed as the parallel composition of two modules: the `Reglevel` module and the `Comsumption` module.

- The `Reglevel` module (see Fig.2): When active, `Reglevel` keeps the level in between the low and high levels. It operates according to a square cycle. When `StartReg` occurs the regulation is switched on. For simplicity, we assume that initially, the tank is in a "normal" state (neither too much nor too less liquid). The regulation operates upto the occurrence of the `StopReg` signal that stops the regulation. Note the concise and precise expression of mixing sequence, iteration and reactive statements. Such behaviours are common in industrial control processes.

- The `Comsumption` module acts in a similar way. The awaited signals are `StartComsume` and `StopComsume` and their respective effect is the starting or the stopping of the pump.

```
Program unit                [module] REGLEVEL :
                            input StartReg, StopReg, HL, LL;
                            output OpenVin, CloseWin;

Reactive statement          [await] StartReg;
                            emit CloseVin;
                            do
Infinite loop                   [loop]
                                    await LL;
Action on the environment           [emit] OpenVin;
                                    await [not HL];
                                    emit CloseVin
                                end loop
Preemptive                  [watching] StopReg
statement                   end module
```

Figure 2: Esterel module of a level regulation

Orders to start and stop the regulation or the consumption are signals imposed to the modules. The **Esterel** program that implements the "watering system" behaviour corresponds to the parallel execution of the two modules (Fig.3).

```
                            module WATERING :
Comments                    [%] Control without monitoring
                            input StartReg, StopReg, StartConsume, StopConsume;
                            output OpenVin, CloseWin, PumpOn, PumpOff;

Local communication         [signal] HL, LL in
                                run COMSUMPTION
                            ||
Insertion of module             [run] REGLEVEL
                            end signal
                            end module
```

Figure 3: Esterel Module of the Watering System

4.3 Faulty situation

Now, consider an abnormal situation. If the level in the tank is too low the comsumption should stop in order to avoid the destruction of the pump (safety property). It is typically the behaviour of system which needs a *fault detection* and a *recovery* treatment. In this case we add a `Comsumption_Monitor` module (Fig.4) which is in charge to detect the underflow. While the comsumption is active, the underflow exception is raised whenever the LL signal is present for three consecutive occurrences of H. Note the flexibility and the expressive power of the `trap ...handle` constuctor.

```
                            module COMSUMPTION_MONITOR :
                            input H, LL, ComsumptionOn;

Variable declaration        var cpt_MN: integer in
                            % Detection part of the monitoring
                            cpt_MN := 0;
                            loop
Begining of the               trap MONITOR in
exception block                 loop
                                  present LL then
                                    cpt_MN:= cpt_MN + 1;
                                    present ComsumptionOn then
                                      if cpt_Mn >= 3 then
Exception raising                       exit MONITOR;
                                      end if
                                    end present
                                  else
                                    cpt_MN:=0
                                  end present
                                each H;
Exception handling            handle MONITOR do
                                % The Recovery of the underflow should be here
                                % execution of the COMSUMPTION_DECISION module
                              end trap
                            end loop
                            end var
                            end module
```

Figure 4: Esterel module of the comsumption monitor

4.4 Error recovery

This section concerns the behaviour of the *decision block* described in Section 2. The `Comsumption_Decision` module (Fig.5) has the responsability of recovering from the underflow error. When the `Monitor` exception has been raised (Fig.4), the `Comsumption_Decision` module starts an `FixUnderflow` asynchronous task and broadcasts the `PumpException` signal to other modules.

Starting asynchronous task is possible in **Esterel** using the `exec` primitive. The asynchronous task can stop in two different cases:

- The repairing duration is less than 10 minutes: It is a normal ending, a `FixPumpDone` signal is emitted and the concurrent executions are aborted.
- The task duration goes through this delay: A timeout clause is executed. In this example, the timeout treatment consists in the emission of an alarm to the senior module.

```
                              module COMSUMPTION_DECISION:
                              input MINUTE, PumpAck;
                              output PumpException, PumpAlarm, FixPumpDone;
Task declaration              task Fix_Underflow()();

                                  [
                                      do
Asynchronous execution                    exec Fix_Underflow()()
                                      watching 10 MINUTE
Optional timeout clause               timeout
                                          emit PumpAlarm;
                                          await PumpAck
                                      end ;
                                      emit FixPumpDone
                                  ]
                              ||
                                  do
Continuous emission                   sustain PumpExec
                                  watching FixPumpDone
                              end module
```

Figure 5: Esterel module of the comsumption decision

4.5 Safe comsumption

A *safe comsumption* process results from the composition of the three modules described previously. Unlike asynchronous languages, this kind of composition does not involve an overhead at run-time. This is due to the algorithm used by the **Esterel** compiler to solve the communication equations. The resulting program is listed in (Fig.6). The `Safe_Consumption` module must be substituted for the `Consumption` module in Fig.3, in order to obtain a safe watering system.

```
module SAFE_COMSUMPTION:
input Start, Stop, StartComsume, StopComsume, H, LL, MINUTE, PumpAck;
output PumpOn, PumpOff, PumpAlarm, StartReg, StopReg;
task Fix_Underflow()();

signal PumpException, ComsumptionOn, FixPumpDonein
    await Start;
    emit StartReg;
    do
        loop
            await StartConsume;
            emit PumpOn;
            do
                sustain ComsumptionOn
            watching StopConsume;
            emit PumpOff
        end loop
    ||
        run COMSUMPTION_MONITOR
```

```
        watching Stop;
        emit StopReg
    end signal
    end module
```

Figure 6: Esterel module of a safe comsumption

5 Conclusion

Face to the ever-increasing complexity of industrial control systems, a hierarcical approach to the design is generally adapted. The lower layers of the hierarchy are *real-time reactive systems*. Their design has been the subject of this paper.

In order to improve the dependability of systems, layers are composed of modules endowed with some "intelligence". A module can act as a filter with respect to the control. Moreover, it is able to detect failures and even to fix them. The control, the fault detection and the fault recovery are cooperating activities. According to the Cartesian approach, these activities should be described separately and then composed. In this paper we have advocated the use of a synchronous language "**Esterel**" in the design of the lower layers of industrial control systems.

Languages are suitable for expressing expected behaviours. **Esterel** is well-equipped to specify reactive statements in a precise and concise way. As a synchronous language, **Esterel** composes very well. Contrary to asynchronous languages, the **Esterel** compiler generates an efficient code which is predictable and with low overhead at run-time. Last, but not least, the *logical correctness* of synchronous programs can be easily checked.

The example presented in this paper points out the power of the **Esterel** stuctures and statements to deal with communication, exception and concurrency. Initially the comsumption system has been implemented without monitoring. In a progressive way fault detection and fault decision treatments have been introduced.

The modules implemented in this example are very standard ones. They can be considered as instances of some generic modules. For instance, if we supplement the system with a regulation of the concentration in the vat, the regulation of the concentration is structurally similar to the regulation of the level. In that case, it is sufficient to rename signals when copying the regulation module. This observation leads us to developping a library of generic re-usable components.

References

[1] D. Harel, A. Pnueli, *On the Development of Reactive Systems*, in Logic and Models of Concurrent Systems, NATO ASI Series, Vol 13 (K.R Apt, ed.) Springer-Verlag, 1985, pp. 477-498.

[2] *Another Look at Real-Time Programming*, (A. Benveniste, G. Berry, ed.), Proceedings of the IEEE, Vol 79, n 9, Sept 91, pp1268-1336.

[3] F Boussinot, R De Simone, *The ESTEREL Language*, in [2], pp1293-1304.

[4] M. Combacau, M. Courvoisier, *A Hierarchical And Modular Structure For F.M.S. Control And Monitoring*, First International Conference on A.I., Simulation and Planning in High Autonomy Systems, Tucson, Arizona, March 1990.

[5] S. Elkhattabi D. Corbeel J.C. Gentina *Integration des aspects de sureté de fonctionnement dans la conception des systèmes de commandes d'ateliers fléxibles*, Congrès IFAC, Ottawa, Canada, Mai 1992.

[6] C. Andre, M.A. Peraldi *Hard Real-Time System Implementation on a Microcontroller* IFAC Workshop on real-time systems, Bruges, Belgium, June 1992.

[7] CISI Ingéniérie, *esterel v3: User's manuals*, Valbonne, France, 1991.

[8] R.de Simone, D. Vergamini, *Aboard AUTO*, Tecnical Report INRIA N111, October 1989.