Introducing the Synchronous Approach into a Real-Time Course
– Draft version –

C. André, M-A. Péraldi, J-P Rigault
Laboratoire Informatique, Signaux, Systèmes (I3S)
Université de Nice-Sophia Antipolis / CNRS

April 8, 1997

Abstract

This paper reports on the introduction of the synchronous approach in a full course in real-time system programming. This course is intended for both computer science and electrical engineering students.

The originality of the course is to combine classical real-time programming and a new coming synchronous programming. The content of the course is given and commented. A particular attention is paid to the synchronous part and the associated software tools and environment for teaching.

Keywords : Real-time, Education, synchrony.

1 INTRODUCTION

According to USA Today[1], in 1995, we were supposed to encounter 50 microcontrollers daily. This number is certainly far greater today. Micro-controllers are instances of “small” real-time systems. Telecommunication systems are large real-time applications. Even if *Real-Time Systems* (RTS) are so common, they are not sufficiently taught. One of the reasons is the inherent complexity of RTS. Knowledge in computer sciences, in automatic control, in electronics are generally necessary to deal with such systems. To make matters worse, developing a real time application requires some familiarity with the system to be monitored and/or controlled.

We have got the opportunity to teach RTS to students with different curricula: training as engineer in computer science[2], and classical studies in electrical engineering[3]. Both have a weak point: computer scientists are not especially proficient in control and electronics, whereas electrical engineering students lack programming experience. Moreover, this course addresses two different study levels: master’s degree and graduate school.

Taking account of our audience and the limited amount of time available for RTS teaching, we focus on a subclass of RTS. Our main goal is to explain how a computer-based system can be efficiently used in monitoring or controlling “technical processes”. The computer-based system may be a simple micro-controller, an industrial PC, or a set of processors cooperating through a real-time local area network (RTLAN).

[1] May 1, 1995, p 1B, quoted by Auslander[1].
[2] ESSI: École Supérieure en Sciences Informatiques
[3] Faculté des Sciences

As future engineers, our students must be given a classical background in the domain of real time systems. A large part of the course is devoted to the classical approach to real-time systems and to "standard" industrial practice.

They must also be kept aware of new trends and perspectives. Since we are active researchers in the field of *synchronous programming*[4], we have introduced a specific teaching, including recent research results. Software tools and environments, developed for research purpose, are widely used by the students. The *synchronous approach* is not a panacea for RTS design. It is presented as a clean and elegant way of dealing with reactive systems. When control-handling aspects are predominant, this approach allows faster and safer programming, and it can lead to very efficient solutions. Thus, most of our case studies are hardware or software controllers.

At the completion of the course, students should

- have a clear vision of RTS specificities and issues,
- have gain the skills needed to develop small real-time applications, and the potential to tackle larger RTS,
- be opened to research issues.

This paper reports our proposal for a real-time course that combines classical asynchronous and recent synchronous approaches. This course has been successfully integrated into engineer and master degree curricula for three years. The main lines of the presentation are:

- Survey of the contents of the real-time system course,
- Material of the synchronous programming teaching,
- The rationale for choosing the synchronous approach.

2 CONTENTS OF THE RTS COURSE

We only give the head of chapters and some keywords. Recall that we have adopted an engineer point of view. This choice is reflected by our textbook selection.

2.1 Table of contents

1. Formal models
   - Automata,
   - Transition systems,
   - Petri Nets
2. Features of RTS
   - Temporal constraints,
   - Concurrency,
   - Predictability,

[4]Project SPORTS (Synchronous Programming Of Real Time Systems), Lab. I3S, Sophia Antipolis (F). See the pages on the web: `http://wwwi3s.unice.fr/~map/sports.html`

- Dependability

3. Life cycle
   - Development cycle,
   - A Semi formal specification: SA/RT
4. Classical approaches to RTS programming
   - The many ways to manage tasks cooperation,
   - Real-Time Multasking Kernel,
   - High-level languages
5. Scheduling
   - periodic/aperiodic,
   - preemptive/non-preemptive,
   - static/dynamic
6. The **Synchronous Approach**
   - (see below.)
7. Distributed real-time systems,
   - Computer Integrated Manufacturing (CIM),
   - Fault-Tolerant architectures,
   - Real-time local area networks: field-busses

2.2 References

We encourage the students to access to the general literature on RTS. The real-time systems glossary and the reference library, compiled by Zalewski[5] are useful and convenient documents. For undergraduate students, we especially advise books describing an engineer approach to RTS. Their author(s) has (have) got a great experience in industrial control applications that they share with the reader. Just to mention some, we often refer to the books written by Halang & Sacha [2], Laplante [3], Bennett [4], Labrosse [5].

2.3 Organization

This RTS course is not a monolithic block. Several teachers are involved. Chapters are distributed among three different modules. Figure 1 shows the precedence in chapters and defines the scope of each modules. A module consists of lectures, tutorials, practical sessions and laboratory work.

[5] available by ftp: `cs.bu.edu:/IEEE-RTTC/public`

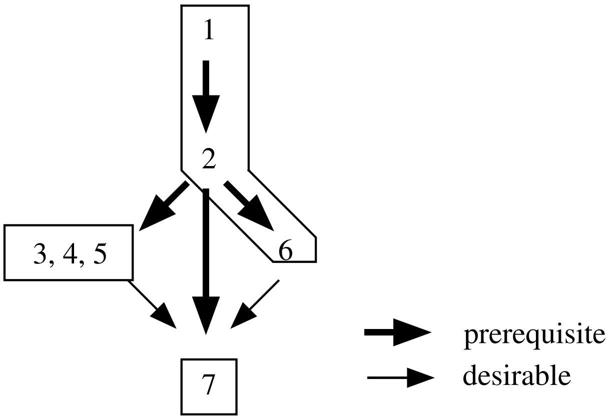

Figure 1: Precedence

3 SYNCHRONOUS PROGRAMMING COURSE

3.1 Plan

The sections of the synchronous programming chapter are:

1. Introductory example: a mouse driver
2. The synchronous paradigm
3. A declarative synchronous language: LUSTRE
4. An imperative synchronous language: ESTEREL
5. A graphical representation: SYNCCHARTS
6. Reactive objects
7. Formal proofs
8. Compilation of synchronous languages
9. Implementation – Execution machines

3.2 Comments

Section 1 introduces a simple *reactive system*. The goal is to detect either a single tick or a double tick on a mouse. Synchronous solutions to this problem are described in the special issue of the proceeding of the IEEE entitled "Another Look at Real-Time Programming" [6]. Both declarative and imperative solutions are given.

Section 2 explains and justifies the *"perfect synchrony hypothesis"* which allows to abstract away reaction times. It also stresses on the advantage of having concurrency, determinism, and preemption as orthogonal concepts.

LUSTRE [7] is a functional synchronous language described in **section 3**. By several points, this language is close to temporal logics and to sequential circuits. The former aspect is appealing for computer science students. The latter makes it easier to understand LUSTRE programming for electrical engineering students.

ESTEREL [8] (**section 4**) is the oldest, and in our opinion the most representative, synchronous language. Its programming style is imperative. The kernel language, restricted to pure signals, is rather simple, and its semantics can be mathematically characterized by a score of rewriting rules. The issue of "causality cycles", for long an obscure problem for the common esterel programmer, is now well-understood [9]. The different semantics of the language (behavioral, operational, electrical) are studied only in the graduate school course.

Engineers in the field of control and manufacturing systems are somewhat reluctant to synchronous languages. As a rule, they prefer *graphical approaches*. The GRAFCET [10], also known as Sequential Function Charts (SFC), is widely used in industrial programmable logic controllers. STATECHARTS [11], a hierarchical visual model which is part of the STATEMATE environment, allows design of complex reactive system and it takes advantages from its industrial support. In **section 5** we introduce these models, but we focus on SYNCCHARTS. SYNCCHARTS [12] is a new graphical synchronous model that inherits from STATECHARTS, but with a better semantics (fully compatible with the ESTEREL semantics). SYNCCHARTS also supports different kinds of preemption.

Reactive objects [13, 14] introduced in **section 6**, combine reactive programming and objects. Even if some methodologies like ROOM [15] advocates the use of objects in RTS design, this promising approach needs further research and is taught only to graduate students.

Section 7 is also highly linked to research. It appears that synchronous languages, which are founded on mathematical semantics, lend themselves to formal analysis. Safety properties, which are a major concern in RTS can be formally tested (see tools below). So is the bounded responsiveness, another form of safety properties. Temporal properties are, of course, also considered.

Section 8 is reserved to computer scientists.

Section 9 is the necessary come back to the real (asynchronous) world. We describe possible architectures of *execution machines* dedicated to synchronous program execution. Studied target machines are micro-controllers [16], PC, or workstations.

3.3 Tools and software environment

For research, we have developed packages that allow definition, rapid prototyping, property checking and implementation of synchronous programs. These tools, with other imported tools, are gathered in a development platform [17] used by the students.

- Input formats:
  - textual for ESTEREL and LUSTRE programs,
  - graphical for SFC and SYNCCHARTS
- Compilers:
  - ESTEREL and LUSTRE compilers,
  - SFC and SYNCCHARTS compilers
- Simulators:
  - XES an interactive simulator of ESTEREL program, with visualization of the execution at the source level,
  - SFC simulation
- Connection to proof tools:

  - testing safety properties of Pure ESTEREL programs and SFC with BAC(Boolean Automaton Checker) [18] and TEMPEST [19] a temporal logic formula checker,
  - automaton and transition system analysis with AUTO [20] and MEC [21],
  - timed automata with HyTech [22]
- Execution machine implementations with various code generators.

4 RATIONALE FOR TEACHING THE SYNCHRONOUS APPROACH

4.1 Point of view

Usually, students acquire programming skills through *transformational system programming.* They are convinced that programs with never-ending loop are surely incorrect; but they do not worry about using "while statements" or recursions.

Turning to *reactive system* programming calls into question many habits. To admit that a lately delivered exact result is incorrect, is almost a revolution.

In our opinion, trying to adapt transformational programming style to reactive programming is a mistake. It seems to be better to start up in reactive or real-time programming[6] with a fresh view, and later on, to see what can be reused from classical programming.

Synchronous languages were introduced to address the issue of *reactive system* programming. As mentioned above, they bring "another look at real-time programming". We have adopted this point of view in Chapter 6 of our RTS course. Figure 1 shows that this chapter could be taught concurrently with the classical approaches. In fact, it is introduced *before.*

Besides the special issue of the proceeding of the IEEE [6], we recommand the reading of the book of N. Halbwachs entitled *"Synchronous Programming of Reactive Systems"* [23].

4.2 Main benefits

Abstraction

The synchronous approach adopts an abstract, indeed even ideal, view of real systems[7]. *Signals* are supposed to be the unique support for exchanging information. Signals are *instantaneously broadcast*: this leads to a global perception of the system. *Reactions* are generated without delay, in synchrony with stimuli.

In programming terms, the consequence is that *(internal) control takes no time.* This strong hypothesis induces drastic simplifications and makes it possible to conciliate concurrency, determinism and preemptions. Note that *preemption* must not be taken in the restricted sense given in scheduling; it is a way to ensure coordination. *"Process preemption ... consists in denying the right to work to a process, either permanently (abortion) or temporarily (suspension)"* (G. Berry [24]).

Thanks to the "perfect synchrony hypothesis" synchronous languages have been given simple, clear and strict semantics. This formal background explains why formal methods applied so well to synchronous programs.

Another noteworthy specificity is that synchronous programs use a *multi-form (logical) time* with a clear notion of instant. Fundamentally, a program fully defines the order of events.

[6] We make a distinction between reactive and real-time. A digital filter must run within strict timing constraints and is therefore a kind of RTS, but it is not especially reactive since its behavior is not influenced by its environment. Conversely, applications like Human Machine Interfaces can be soft RTS but highly event-driven. In fact, controllers and most hard real-time applications are both reactive and real-time.

[7] See reference [12] for details.

Programming discipline

Due to the simplifications underlying the model, the reader may think that synchronous languages are too simple to tackle actual applications. In fact this is not the case. Other simple models like Moore or Mealy machines allow the design of rather complex controllers. So are synchronous languages that can cope with sophisticated reactive behaviors.

Within the framework of the synchronous approach, applications are described as a set of cooperating sub-systems. Each sub-system is characterized by triggering signals and the reactions they cause. The behavior of the whole system results from the instantaneous dialogues among sub-systems.

SyncCharts, which can be considered as a graphical version of Esterel, encourages the use of preemption for synchronization [12]. An "observer" (a sub-system) watches at another sub-system and triggers adequate reactions when special events occur. These kinds of behaviors are common in mechatronics and robotics.

Note that this approach avoid the use of communication or synchronization mechanisms such as those encountered in multitasking programming.

Application domains

We have just mentioned mechatronics and robotics. More generally, all systems with preponderant control-handling aspects can be elegantly and efficiently programmed with synchronous formalisms or languages.

Esterel is not restricted to this kind of application, it has also been used in protocol verification and Human-Machine Interfaces (HMI). Even if timing constraints are not stringent, the behavior of protocol and HMI may be highly reactive.

4.3 Limitations

Of course, the synchronous approach, because of its choices, has a limited scope.

First, synchronous languages have *restricted general data-processing capabilities*. Most of the data-processing must be supported by a host language (usually C). Esterel, for instance, knows only about booleans, integers and little about strings. All other data types are abstract. Esterel focuses on control.

The major objection is about *distributed applications*. The global perception is irrealistic, except for very special cases (synchronous LANs).

The *0-duration hypothesis* is not really a problem. Once compiled, a synchronous program has a deterministic behavior, so that we can check, a posteriori, whether or not the reactions take an amount of time negligible with respect to the time constant of the system to be controlled.

5 CONCLUSION

From an educational point of view, this experimental introduction of synchronous programming in RTS course, has been very instructive.

- Starting with a synchronous language for reactive systems programming has some positive effects. For students, the first contact with synchronous system programming is disconcerting because of this unusual style of programming. Indeed, they cannot establish links with the languages they know. On the one hand this is not a flaw: students are not inclined to mimic other languages. On the other hand, they adopt quickly this new style and they are able to express non-trivial reactive behavior by the end of the first practical session.

- Such a teaching is supported by a friendly and rich programming environment. The student may experiment different input formats, execute its program (interactive simulation with backwards mapping to source), generate code for various targets.

This course is also a good opportunity to initiate student to an open and active research field. First developed in Europe, the synchronous approach is now gaining audience. The introduction of the synchronous languages in the Ptolemy environment [25] is an evidence of this evolution.

References

[1] D.M Auslander. *Mechatronics: A design and Implementation Methodology for Real Time Control Software.* not yet published, available on the web, `dma@euler.me.Berkeley.edu`, Berkeley (CA), 1996.

[2] W.A. Halang and K.M. Sacha. *Real-Time Systems: Implementation of Industrial Computerised Process Automation.* World Scientific, Singapore, 1992.

[3] P.A. Laplante. *Real-Time Systems: Design and Analysis, an engineer's Handbook.* IEEE Press, 1993.

[4] S. Bennett. *Real-Time Computer Control – An introduction.* System and Control Engineering. Prentice Hall, UK, 1994.

[5] J.J. Labrosse. *Embedded System Building Block.* R&D Publications, Lawrence (KS), 1995.

[6] A. Benveniste and G. Berry. Another look at real-time programming. *Proceeding of the IEEE*, 79(9):1268–1336, September 1991.

[7] P. Caspi, N. Halbwachs, D. Pilaud, and P. Raymond. The synchronous data flow programming language LUSTRE. *Proceeding of the IEEE*, 79(9):1305–1320, September 1991.

[8] F. Boussinot and R. De Simone. The ESTEREL language. *Proceeding of the IEEE*, 79(9):1293–1304, September 1991.

[9] G. Berry. *The Constructive Semantics of pure Esterel.* not yet published, available on the web, `cma.cma.fr`, Sophia Antipolis (F), 1996.

[10] IEC, Genève (CH). *Preparation of Function Charts for control systems*, december 1988. International standard IEC 848.

[11] D. Harel. STATECHARTS: A visual formalism for complex systems. *Science of computer programming*, 8:231–274, 1987.

[12] C. André. Representation and analysis of reactive behaviors: A synchronous approach. In *Computational Engineering in Systems Applications (CESA)*, pages 19–29, Lille (F), July 1996. IEEE-SMC.

[13] F. Boussinot. *La programmation réactive. Application aux systèmes communicants.* Collection technique et scientifique des télécommunications. Masson, Raris, 1996.

[14] C. André, F. Boulanger, M.A. Péraldi, J.P. Rigault, and G. Vidal-Naquet. Objects and synchronous programming. *JESA*, ?(?):??–??, ? 1997.

[15] B. Selic, G. Gullekson, and P.T. Ward. *Real-Time Object-Oriented Modeling.* J. Wiley Publ., 1994.

[16] C. André and M-A. Péraldi. Predictability of a RTX2000-based implementation. *Real-Time System*, 10(3):223–244, May 1996.

[17] G Noubir and M-A Péraldi. Vérification de la sûreté d'exécution de grafcets. In *Real-Time & Embedded Systems (RTS&ES'96)*, pages 359–368, Paris (France), Janvier 1996. teknea.

[18] N. Halbwachs. BAC: A boolean automaton checker. Technical report, VERIMAG, Montbonnot (France), February 1994.

[19] Lalita J. Jagadeesan, Carlos Puchol, and James Von Olnhausen. Safety property verification of esterel programs and applications to telecommunication software. Liege, Belgium, July 1995. Conference on Computer Aided Verification (CAV'95).

[20] R. de Simone and D. Vergamini. Aboard AUTO. Technical Report 111, INRIA, October 1989.

[21] A. Arnold, D. Begay, and P. Crubillé. *Construction and Analysis of Transition Systems with* MEC, volume 3 of *AMAST Series in Computing.* World Scientific, Singapore, 1994.

[22] T. Henzinger, P.-H. Ho, and H. Wong-Toi. A user guide to hytech. Technical Report CSD-TR-95-1532, Department of Computer Science, Cornell University, 1995.

[23] N. Halbwachs. *Synchronous Programming of Reactive Systems.* Kluwer Academic Publishers, Amsterdam, 1993.

[24] G. Berry. Preemption in concurrent systems. *Proc FSTTCS, Lecture notes in Computer Science*, 761:72–93, 1992.

[25] W.-T. Chang, A. Kalavade, and E.A. Lee. Effective heterogenous design and co-simulation. pages ??–??, Lake Como (I), June 1995. NATO Advanced Study Institute Workshop on Hardware/Software codesign, NATO.