GRAFCET and Synchronous Languages

Charles André, Marie-Agnès Péraldi

Laboratoire Informatique, Signaux et Systèmes (I3S)
41 Boulevard Napoléon III
F 06041 Nice cedex
e-mail andre@cma.cma.fr

ABSTRACT. GRAFCET is a convenient model for expressing the behavior of control systems. Synchronous Languages are devoted to reactive system programming. In this paper, we compare and bring these two approaches together. Underlying concepts are the bases of our comparison. Synchronism appears to be the common denominator. Translations from GRAFCET into ESTEREL and LUSTRE are proposed. Such translations allow GRAFCET to use the sofware environment developed for synchronous languages. Joint use of GRAFCET and synchronous languages are suggested for the future.
KEY WORD. GRAFCET, function charts, synchronous languages.

1 Introduction

GRAFCET is a formal model suitable for the representation of the expected behavior of *control systems.* This model is widely spread over both the industrial and the teaching fields. GRAFCET is a French standard [AFN 82]. The international standard [IEC 87] is known as *"function charts for control systems"*. For simplicity, in what follows, we refer to both models as GRAFCET.

Synchronous languages are new coming programming languages. They are devoted to *reactive system* programming [HAR 85]. Recently, a survey of the synchronous approach has been published in the "proceedings of the IEEE" [I3E 91], the reader is urged to refer to this special issue for a deeper presentation. Imperative synchronous languages such as ESTEREL [BOU 91] are well-suited to "event-oriented" approach to reactive systems; equational languages such as LUSTRE [HAL 91] and SIGNAL [LEG 91] are more convenient for "data-flow" systems.

GRAFCET and synchronous languages have numerous common features:

• *application domains are overlapping:* industrial control systems are special reactive real-time systems (i.e reactive systems subject to temporal constraints). In fact, synchronous languages address a class of systems larger than GRAFCET does.

• *uses are comparable:* both formalisms can be used as a control language or as a behavioral model. On the one hand, a GRAFCET for a controlling system can be considered as a control program with respect to the controlled system. On the other hand, a program written in a synchronous language can be taken as a specification of the expected behavior of a system, independently of any implementation.

In this paper, after a brief introduction to an imperative synchronous language (ESTEREL) and an equational synchronous language (LUSTRE), GRAFCET is compared to those languages. We aim at finding common underlying concepts and possible discrepancies. In a second part, we explain how to "rewrite" a GRAFCET into a synchronous program. The mutual benefit of such translations to both GRAFCET and synchronous languages is reviewed in a third section. The main effect on GRAFCET is the availability of software environment initially developed for synchronous languages. As for synchronous languages, they are enriched with a new graphical formalism, very convenient for industrial automatic control systems. Finally, we will point out the future prospects of a joint use of GRAFCET and synchronous languages in research and in development, as well.

2 Formalisms

In this section, we briefly present GRAFCET and two synchronous languages.

2.1 *GRAFCET*

This model has been recalled in the introductory paper of the present issue [PRU 92]. More details are to be found in [IEC 87]. The evolution rules are the key points to our comparison, so, for convenience and according to the French use, we assign a number to each rule:

Initial Situation	[G1]
Clearing of a transition	[G2]
Evolution of active steps	[G3]
Simultaneous clearing of transitions	[G4]
Simultaneous activation and de-activation of a step	[G5]

An extra rule must be mentionned:

Clearing time of a transition or activation time of a step	[GD]

This duration is as short as one may wish, but it can never be zero.

The IEC standard defines GRAFCET as it was in the 80's. Some extensions have been proposed since then. Most of those extensions are described in [G7W 92]. In what follows, the reader is assumed to have some familiarity with the syntax of GRAFCET and its evolution rules.

2.2 *Synchronous Languages*

In this paper, we are not interested in the concrete syntax of synchronous languages. We consider them as a new approach to the issue of real-time systems'programming. As explained in an excellent introduction to the synchronous approach [BEN 91], synchronous languages are based on simple but rigourous mathematical grounds. Hence, synchronous languages are given a mathematical semantics, they can be compiled efficiently, and there exist techniques to prove the *logical correctness* of a synchronous program.

There are two programming styles: the imperative and the declarative styles.

• ESTEREL [BOU 91] is an *imperative language*. At first, it was introduced to cope with the issue of real time programming. Discarding many of the assumptions made by classical "real-time" asynchronous approaches, ESTEREL has brought a fresh way of thinking about reactive systems. The fundamental hypothesis are that reactions are synchronous with the stimuli and that communication is made by instantaneous broadcast. Those strong hypotheses allow nice properties such as a deterministic parallel composition. ESTEREL supports classical control structures, a parallel operator (||), a powerful exception mechanism, and the original reactive statements. ESTEREL is well-suited to the programming of event-driven systems.

• LUSTRE [HAL 91] is a *declarative language*. A LUSTRE program can be thought of as a network of operators. Operators are synchronous and computations are supposed to be instantaneous. When a real-time system is described by sets of equations, LUSTRE is preferable to ESTEREL. SIGNAL [LEG 91]: another (synchronous) data-flow language has been especially designed for real-time data processing.

3 Comparisons

In this section, we exhibit underlying concepts, so the comparisons are made at the semantics level, not at the syntactic one.

3.1 *Time*

Reactive systems have to maintain permanent interactions with their environment. For most applications, the reactions are temporally constrained, i.e the system must react within given temporal frames. These "real time" properties make reference to the *physical time*. To guarantee that a system is *temporally correct* is mainly a matter of implementation, so synchronous languages do not consider physical time, they use a logical multiform time, instead. In synchronous approaches, *time is basically global and discrete*. Occurrences are associated with *instants*, so that one must be able to say if "something" has occurred before, is occurring simultaneously, or will occur after "something else". Relationship between logical time and physical time is established by distinguished events whose occurrences are interpreted as indication of time passing. A tick of a physical clock is such an event. But any repetitive input signal can be taken as a generalized time basis (e.g litre, metre,…) so that the time is said to be *multiform.*

Evolutions rules of GRAFCET , especially G4, make implicit reference to a global time. What is not clear with the standard is whether the time is continuous or discrete. An attempt at giving a formal interpretation of the temporal behavior of GRAFCET is presented in this issue [FRA 92]. Clearly, a discrete time makes analysis simpler. Continuous time poses the problem of giving sound definitions to instant and simultaneity.

Generally, reactive systems are composed of several communicating subsystems. Any candidate to such system modelling has to express concurrency, synchronization and communication. Facilities for hierarchical description and reactive statements are desirable, too. Next subsections explain how GRAFCET and synchronous languages support these requirements.

3.2 *Synchronicity*

The *synchrony hypothesis* assumes that instants are not overlapping. Of course, many actions can be done at the same instant, but all are submitted to the same environment. ESTEREL, LUSTRE, and SIGNAL make a stronger assumption known as the *"strong synchrony hypothesis"*. According to this hypothesis, reactions are synchronous and instantaneous, so that stimuli and reactions are simultaneous. A consequence is that the computations associated with a reaction take no time at all. This is, of course, a drastic simplification of reality, but it gives rise to elegant and powerful semantics, and it leads to efficient composition rules. Thanks to the strong synchrony, the designer may concentrate his or her attention on the logics of the synchronization, instead of on mechanisms such as those provided with asynchronous approaches.

Because of rules G3 and G4, GRAFCET is a synchronous model. However, it is not strongly synchronous: the extra rule GD imposes a non-zero duration for a reaction. In ESTEREL causality is supposed to take no time, therefore it is legal to have a sequence of actions in a null-time. For GRAFCET a small amount of time is assumed between possible sequential clearings of transitions, even if they occur at the same "instant".

There exists another discrepancy between GRAFCET and synchronous languages. GRAFCET considers that two non-correlated events cannot occur at the same instant: they must be separated by a non-null interval of time. This hypothesis seems to be a way to express the asynchrony of the incoming events. In our opinion, this restriction is not a fundamental one, because the non-correlation of events is a matter of interpretation which is not directly relevant to the model itself. Note that this restriction is not even mentioned in the international standard.

3.3 *Concurrency*

ESTEREL has a parallel operator (||) that allows a restricted form of concurrency ("fork-join"-like structures). GRAFCET is more flexible, simultaneous divergences and convergences are not necessarily pairwise. As for data-flow languages, concurrency is implicit and is fine grain (concurrent excecution of operators).

3.4 *Hierarchy*

Hierarchical descriptions allow to deal with complex systems. Synchronous languages offer such facilities. The programming unit in ESTEREL is the *module*. A module can use other modules. In LUSTRE, the basic block is the *node* that can be composed of several nodes.

In GRAFCET, a sequence of steps which occur several times may be represented by a single step symbol (macro-step). This is only a syntactic commodity. The notion of *partial grafcet,* defined in extended GRAFCET [G7W 92], introduces a real hierarchy. The semantics of this extension is not well-established yet (see [LHO 92] in this issue).

3.5 *Interruptions*

This word is given the general meaning of any cause of alteration to the "normal" behavior. Interruptions are explicitely used in imperative approaches. ESTEREL is the richest. There exist two forms of abortion: *preemption* (prevent a block from being executed) and *branching after execution* (concurrent statements terminate what they have to do during the current instant and then abort). The *suspension* (temporary stopping) is not a primitive of the language but equivalent behavior can be obtained by programming. Moreover, ESTEREL is endowed with a powerful mechanism for exception handling. Of course, all the interruptions have a precise semantics, so that exceptions can be combined with concurrency without restriction.

Standard GRAFCET does not support interruptions, but its extension does. Activation (de-activation, resp.) of a step may activate (de-activate, resp.) a partial grafcet. It is even possible to save the current situation (i.e the set of active steps) of a partial grafcet and to restore it subsequently.

Data-flow languages have implicit suspensions by means of subclocks: some data are processed when and only when a subclock is true.

3.6 *Communication*

Interactions between subsystems generally involve information exchange (communication). Synchronous languages adhere to the instantaneous broadcasting discipline for communication. Because of this hypothesis, information is global: any subsystem has the same global vision. There is no distorsion due to asynchronous propagations with inherent delays. In ESTEREL, communication is done using *signals* that can be emitted, tested for presence, and that can convey a value. Note that ESTEREL allows private communication by a signal hiding mechanism.

The global vision is the rule for GRAFCET, too.

Combining "strong synchrony" with "instantaneous broadcast" is often refered to as the *"perfect synchrony hypothesis"*.

3.7 *Determinism*

Thanks to the perfect synchrony hypothesis, parallel composition is deterministic in synchronous languages, contrary to asynchronous ones. In fact, ESTEREL is a fully deterministic language, non-deterministic behaviors are rejected by the compiler. GRAFCET is deterministic, too.

Determinism is a most important property for real-time systems which must be *predictable*.

4 Rewriting GRAFCET into Synchronous Languages

As seen in the previous section, there are many similarities between GRAFCET and synchronous languages. Now, we propose translations from GRAFCET into synchronous languages. These translations have to respect the evolution rules of GRAFCET and must lend themselves to automatic generation. We suggest translations into an imperative language (ESTEREL) and an equational one (LUSTRE). The use of SIGNAL is described elsewhere [MAR 92]. Since translations are greatly language-dependent, we restrict our attention to the principles

4.1 *Sequential evolutions*

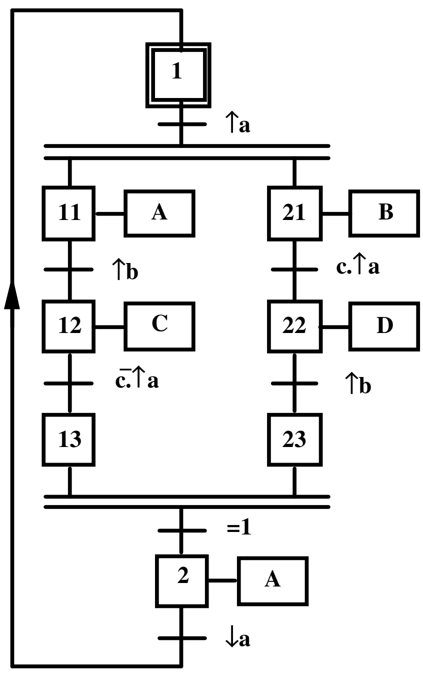

Figure 1: *A Well-Structured GRAFCET .*

In asmuchas GRAFCET represents sequences, ESTEREL seems to be more adequate target language. In fact, the translation is simple for "well-structured" grafcets with receptivities (transition conditions) including triggering events. Figure 1 illustrates such a case. ↑a (↓, resp.) denotes the triggering event "occurrence of a rising (a falling, resp.) edge" of the boolean-valued variable a. A possible translation in ESTEREL is:

```
     module Grafcet1:
     input Upa, Upb, Dna;
     output A, B, C, D;
     sensor c: boolean;
         loop
           await Upa;
           [
            [do
                 sustain A
              watching Upb;
              trap X12 in
                 loop
                   present Upa then
                      if ?c else exit X12 end if
                   end present;
                   emit C
                 each tick
              end trap]
            ||
            [% right hand branch,
              % omitted for simplicity
            ]
           ];
           do
              sustain A
           watching Dna
         end loop
end module
```

4.2 *Receptivity*

Sometimes in a grafcet, a transition condition may be true before the enabling of the transition. In this case the previous translation is not sufficient. Steps have to be considered as local states. ESTEREL can support this translation, but LUSTRE yields a simpler solution. We define a LUSTRE's node called "STEP" whose behavior is the one of actual steps.

```
node STEP (Init, Set, Reset: bool)
        returns(X, XS: bool);
```

```
-- X is true iff the step is active
-- XS is the next state of the step
   let
     X = Init -> pre(XS);
     XS = if Set then true else
                    if Reset then false else X;
   tel;
```

For the complete grafcet, activation and de-activation conditions for each step are expressed by equations. Here is an excerpt from the full translation:

```
node GRAFCET1 (a, b, c: bool)
         returns (A, B, C, D;bool);
  var  X1,XS1,X2,XS2,…,
         SetX1,ResetX1,SetX2,ResetX2,…:bool;
  let
    (X1,XS1) = STEP (true, SetX1, ResetX1);
    (X2,XS2) = STEP (false, SetX2, ResetX2);
    …
    SetX12 = X11 and (b and not pre(b)); -- X11.↑b
    ResetX12 = X12 and (not c) and (a and not pre(a));
    …
  tel;
```

4.3 *Instability*

Applying rules G1 through G5 changes a situation into another one. In this new situation some transitions may be enabled and the associated condition true. So, the situation is *unstable* and must change again. ESTEREL allows sequences of actions at the same instant, therefore changes in situation with finitely-many unstable transient situations can be simulated in ESTEREL. This is not a trivial exercise, see [MAR 92] for the difficulty in expressing unstability with SIGNAL.

5 Future research and development

Each synchronous language has its own industrial compiler. Because of the complementarity of the different languages, there is a trend towards developing common intermediate representations and tools [BEN 91]. After translation, GRAFCET may beneficit from this common environment. Let us mention some positive outcomes:

• *Verification:* theorem provers, model checkers, finite state machine analyzers are available. They can establish the logical correctness of a program and especially *safety properties*. A typical undesirable property for a grafcet is the existence of infinitly-many transient situations. This faulty behavior can be detected statically, by a synchronous langage compiler.

• *Implementation:* synchronous languages are compiled into intermediate codes (a finite state machine-based code for ESTEREL and LUSTRE, a decorated graph for SIGNAL). The intermediate code is then translated into general purpose languages or into specific microcontroller languages (see [AND 92]). The resulting code is safe and efficient.

• *Semantics:* the synchronous languages rely on rigourous mathematical semantics. Rules G1 to G5 are not sufficient to guarantee the unicity of interpretaion of a grafcet. The semantics of GRAFCET must be revisited. One way to cope with this problem is proposed in [LHO 92], the synchronous language approach may be an alternative way.

As yet, we have discussed about the translation of GRAFCET into a target synchronous language. Ongoing research tries to make direct compilation of GRAFCET into an intermediate code. Advantages are many-folds:

• greater efficiency in code generation,

• better use of specific characteristics of GRAFCET (particularly in industrial control specification),

• multiformalism approach (mixing GRAFCET and languages),

• unification of concepts.

The couple ESTEREL-GRAFCET might be most interesting for complex control system design. This mixed approach could combine graphical aspects of GRAFCET with powerful interruption and exception handling of ESTEREL.

6 References

[AFN 82] AFNOR, *Norme C03-190*, Paris, 1982.

[AND 92] C. André, M.A Péraldi, *Hard Real-Time System Implementation on a Micro-controller*, Proc IFAC Workshop on Real Time Programming WRTP'92, Bruges, 1992, pp185-189.

[BEN 91] A. Benveniste, G. Berry, The Synchronous Approach to Reactive and Real-Time Systems, in [I3E 91], pp1270-1282.

[BOU 91] F. Boussinot, R. De Simone, *The ESTEREL Language*, in [I3E 91], pp1293-1303.

[FRA 92] J.P Frachet, G. Colombari, *??????????????*, in this issue.

[G7W 92] ADEPA-AFCET, *Le GRAFCET*, Cépaduès Editions, 1992

[HAL 91] N. Halbwachs, P. Caspi, P. Raymond, D. Pilaud, *The Synchronous Data Flow Programming Language LUSTRE*, in [I3E 91], pp1304-1320.

[HAR 85] 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, pp477-498.

[I3E 91] *Another Look at real-Time Programming*, (A. Benveniste, G. Berry, ed.), Proceedings of the IEEE, vol 79, n 9, Sept 1991, pp1268-1336.

[IEC 87] International Electrotechnical Commission, *Preparation of function charts for control systems*, IEC 848, 1987.

[LEG 91] P. Le Guernic, T. Gautier, M. Le Borgne, C. Le Maire, *Programming Real_Time Applications with SIGNAL,* in [I3E 91], pp 1321-1336.

[LHO 92] P. Lhoste, H. Panetto, M. Roesch, *??????????????*, in this issue.

[MAR 92] L. Marcé, P. Le Parc, ----------------, Proc IFAC Workshop on Real Time Programming WRTP'92, Bruges, 1992, pp---------

[PRU 92] F. PRUNET, *??????????????????*, in this issue.