The Synchronous Languages 12 Years Later

Tawatchai Siripanya

Seminar in Programming Language(19666)
Advisor: Lilit Hakobyan
Supervisor: Prof. Dr. Elfriede Fehr
Institute of Computer Science
Freie Universität Berlin
Date: 26.02.13

Abstract

The synchronous programming language is a prominent language used in real-time embedded systems—typically in safety-critical embedded systems. It has been used in diverse projects in many well known companies such as Airbus, Scheider Electric, and Texas Instruments. Examples of the synchronous programming language include: Esterel, Lustre, and Signal. These languages have been developed more than 20 years [2]. The paper aimed to review the history of the synchronous programming languages, what have been achieved in the languages, what are their difficulties , and what majors problems remain. We have found that the synchronous languages are successful in the real-time embedded system industries. Their major problems are compilation problems—especially the Esterel complication and handling with arrays—it is still an open issue. The visual notations as part of the language requirements tend to move forward and highlight good potential features of the synchronous language.

Keywords: synchronous languages, Esterel, Lustre, Signal

Contents

1. Introduction

The synchronous languages have found their way in the industries as a technology to model, specify, validate, and to implement real-time embedded applications [2]. The languages are mostly used in safety-critical embedded systems and applied in many projects such as a nuclear power plant project (Schneider Electric), a part of the flight control software (Airbus A340-600), and the re-engineered track control system of the Hongkong subway (CS Transport) [2]. The examples of the embedded control system are for instance, flight control system, flight-by-wire avionics, and anti-skidding or anti-collision equipment on automobiles. The well known synchronous languages are such as Esterel [5], Lustre [6], and Signal [10].

Time has passed since the first synchronous language has been introduced in 1980s. It is interesting to review the history of the synchronous programming languages, what have been achieved in the languages, what are their difficulties , and what majors problems remain.

The first part of the paper describes the philosophy of the synchronous language. The second part discusses the improvement of the language, emphasizes the technologies to compile the synchronous language and gives insight of the difficulties, and the successes happened with the language. Finally, the last part concludes the paper.

2. Theoretical background

The three synchronous languages (Esterel, Lustre, and Signal) are build on a mathematical framework that allows us to prove the correctness of the systems. The mathematical framework uses the combination of synchrony and the deterministic concurrency. As described in [2], the three main requirements of the synchronous language are concurrency, simplicity,and synchrony. They can be described in the following:
(1) Concurrency—the language must support concurrency and rely on notations that express concurrency such as block diagrams, hierarchical automata, or some imperative type of syntax. (2) Simplicity—the language must have simplest formal model to easy to reason and easy to trace. Also, the parallel composition of two processes must have clean semantics. (3) Synchrony—the language must support the simple and frequently used "implementation model" as shown in the figure 1.

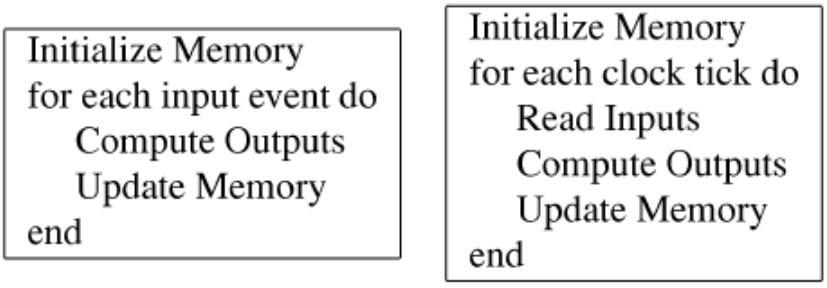

Figure 1: The figure illustrates the implementation model that is simple and frequently used (on the left (event driven), on the right (sample driven). Source: [2]

2.1. The philosophies of synchronous language

Many synchronous approaches have been developed to solve problems in safety-critical embedded systems. Their primary goal is to convince stakeholders of the system that the design and its implementation are correct [2]. We can divide these approaches into four categories include: (1) Microsteps, (2) Acyclic, (3) Unique fixpiont, and (4) Relation or Constraint.

1. Microsteps: We can define a reaction of the system to be a sequence of elementary microsteps. By doing so, we can insist that a reaction remains operational [2]. In the microsteps approach, the components of primitive systems are assumed to be sequences of elementary microsteps. This approach is used in Very High Speed Integrated Circuit Hardware (VHDL), Verilog modeling languages, Harel's Statecharts, and control system (to program programmable logic controllers) [2].
2. Acyclic: We can insist that a system behaves functionally, when the block diagrams of control systems contain no zero-delay loops [2]. This philosophy is used in Lustre.
3. Unique fixpoint: Each reaction of a system is assumed to be the solution of a fixpoint equation. The system always behave functionally when each reaction is a deterministic function of the form

$$\{state, input\} \longmapsto \{next\ state, output\} \tag{1}$$

However, to compile a program of this semantic becomes difficult [2]. This approach is used in Esterel.

4. Relation or Construct: Each reaction of a system is assumed to be a constraint. The reaction can have: zero solutions (no reaction form the program), one solution of the form (1), and multiple consistent solutions (nondeterministic behaviour) [2]. This approach is used in Signal.

3. Successes and Improvements

This section discusses the successes and improvements of the synchronous languages (Lustre, Esterel, and Signal). We begin to describe how the languages have been introduced to the market. Then, we introduce technologies to compile the languages and its improvements. Finally, we describe how to verify and test the synchronous languages, also introduce available verification tools.

3.1. How the languages have been commercialized

3.1.1. Lustre

In 1980s, two big industrial projects were launched in France. These projects were N4 series of nuclear power plants (Schneider Electric[1]) and the Airbus A320 (Airbus Industries[2]) [2]. At that time, there were no tools for highly safety-critical software available. The two companies have built their own tools that based on synchronous dataflow fashion. Schneider Electric has built SAGA and Airbus Industries has built SAO [2]. An ongoing research between Schneider Electric and Lustre cooperation was the reason why SAGA had to use Lustre [2]. Later, the company called Verilog has developed the commercial version of SAGA. It has provided integration capabilities for SAGA and SAO. This has introduced a so called Scade environment at the time. In 2001, Esterel Technologies has brought the Scade from Verilog [2].

3.1.2. Esterel

Dassault Aviation[3] has supported Esterel project from the beginning. The company has used Esterel for a landing gear system and a fuel management system of an air-plane [2]. In 1998, the French software company called

[1]http://www.schneider-electric.com/
[2]http://www.airbus.com/
[3]http://www.dassault-aviation.com/

"Simulog" has introduced Esterel in the market. In 1999, the Esterel Technologies has been found (It was prior a division of Simulog). Furthermore, Esterel has been used by Texas Instrument[4] to design a flow (some parts of C specification have been rewritten in Esterel) [2]. The most significant strength of Esterel is the feature to generate an automatic test suite that grantees state/transition coverage [2].

3.1.3. Signal

In 1980 CNET[5] has supported Signal for the development [2]. The primary goal of Signal was to be a development language for signal processing applications, which runs on digital signal processors (DSPs) [2]. Therefore, features like data-flow, graphical style, handling with arrays, and sliding windows have been introduced in the language. In 1998, Signal has been a joint trademark of CNET and INRIA [2]. In early nineties, Signal was licensed to a French company called "TNI [6]". In 1993, TNI has introduced an application called Sildex tool to the market. Many versions of Sildex have been distributed at the time. Sildex provides capabilities for users to work with data-flow diagrams, state diagrams, and applications from Matlab[7](e.g., Simulink)[2]. After that, TNI and Snecma[8] have cooperated. They have used Signal for Aircraft engine control applications.

3.2. Technologies developed for synchronous language

This section presents technologies that develop for the synchronous language. These include: Handling of arrays, Compiling, and Verification and Testing.

3.2.1. Handling of arrays

Arrays are powerful to structure programs and define parametrized regular networks in a dataflow language such as Lustre [2]. The "map" operation is an example how to apply one operation to the all elements in an array. However, the number of types provided in the synchronous languages are

[4]http://www.ti.com/

[5]Centre National d'Etudes des Télécommunications, the former national laboratory for research in telecommunications, now part of France Telecom(FTR&D)

[6]http://www.tni-valiosys.com

[7]http://www.mathworks.com/

[8]http://www.snecma.com

mostly limited to avoid manipulation of an array that causes an error. This error is called "run-time array-index-out-of-bounds errors"[2] —one might be avoided of. The concept of arrays has been first introduced in Lustre to describe circuits [2]. Then, the Scade tool and the Sildex tool have used this technology as well [2].

3.2.2. Compiling

Esterel firstly has three versions of compiler (V1,V2, and V3). The first version was based on literal interpretation [2]. Then, it has been built further to be based on automata using Brzozowski's algorithm [2]. The first three versions of the Esterel's compiler based on automata work good for compiling a small program. Many techniques have been introduced to optimize to handle larger programs. However, none of the techniques can compile concurrent programs that have longer than 1000 lines [2]. The followers of automata compilers are based on translating Esterel into digital logic[2]. This technique can minimize the size of the executable programs and used in the version four of Esterel (V4). However, it is incompatible with the prior compiler versions (e.g., V3). To avoid this problem the V5 has been built, but it is slower than automata-based 100 times because it is based on logic networks that poor match to imperative languages [2]. In addition, Weil et al. [13] introduces a resembling technique called "compiled-code discrete-event simulators". With this technique, the program is divided into segments. Each one becomes a separate C function and can be invoked by a centralized scheduler [2]. The idea is used bei Weil et al. and it is then called "SAXO-RT". SAXO-RT can compile all valid Esterel programs and it enables us to control the order of the execution within a cycle [2]. Nevertheless, none of the described techniques is considered satisfactory [2].

Signal: The complication of Signal is based on solving the abstraction of the program. The program is described by clock and causality calculus [2]. The first compilation has been introduced in 1988 and it has not changed since then.

To sum up, the compilation of synchronous language is difficult [2]. Traceability is the important requirement that designers of the safety-critical software have to keep in mind. It is one of certification constraints. Many compilations have been introduced and have made different tradeoffs between efficiency and traceability [2]. In [2] suggests to choose either of both. This idea is used in Scade/Lustre compiler that is DO178B (Software Considerations in Airborne Systems and Equipment Certification) certified.

3.2.3. Observers for Verification and Testing

The synchronous observers (see [7] and [8]) provide us to specify properties of programs (to describe non-deterministic behaviours) [2]. The well known technique is presented in [12]. With this technique, we first have to describe unwanted traces of the program and second make sure that these traces are not accepted by the automaton. In synchronous language, we can use observers to specify safety properties. Also, they can be used to observe variables or signals of interest. Two advantages of using observers include: (I) We can specify the safety properties within the program itself (written in the same language), and (II) observers can be executed (good for testing). Furthermore, it can be run during execution that enable us to perform auto test [2]. Many tools for verification are available (see [3], [4], [7], [9], and [11]).

3.3. Difficulties and Unsolved Problems

One difficulty in the synchronous programming language is compilation of Esterel (described in the last subsection). Because Esterel has semantics that include both control and data dependencies. Although many approaches have been introduced, none of them is considered to be the best one. Also, the compilation issue introduces a tradeoff between traceability (it is required for certification constraints) and efficiency. Second, arrays handling is an open issue that is still unsolved problems. A standard method to handle arrays during compilation is not existed. Third, the way the components of a system to communicate with each other depends on buses or serial lines of the deployed architectures. In the real world system such as in process industries, automobiles, or aircraft—the components cannot be considered synchronous. To this issue, the system designers have to find a solution for the system themselves—a standard solution is unavailable among synchronous programming languages. Finally, unlike the issues described above—say opportunities of these languages. Typically, applications in embedded system have longer life time than normal applications such as operating systems. Companies tend to use visual notations to model their applications first, then generate the codes of of the notations. These notations are easy to use for users and reusable. The modelling techniques used in synchronous languages are such as UML(Universal Modeling Language), Simulink/Stateflow a product from Mathlab [2] , and SyncCharts [1].

4. Conclusion

The synchronous languages are mostly used in the industries as a technology to model, specify, validate, and to implement real-time embedded applications. Because of its clean semantics built-in with a common mathematical framework, it is primary used in safety-critical embedded systems. Esterel is the oldest programming language of the synchronous languages which include: Lusterel, Signal, etc. All three languages have been used in real big projects and have been supported from well known companies—that was how people introduce new programming language into the market in the earlier years. All synchronous languages provide ability to model a system (e.g., UML, Simulink, and SyncCharts) and then code generation, because the requirements of the language from the beginning required notations such as block diagrams, hierarchical automata, and some imperative type of syntax. Like most of programming languages, the synchronous languages have difficulties and unsolved problems such as difficulties with the compilation (e.g., Esterel) and handling with arrays. Nevertheless, the languages provide easy to use virtual notations that attract users nowadays and potentially more popular in the future.

References

[1] André, C., Oct. 2004. Computing syncharts reactions. Electron. Notes Theor. Comput. Sci. 88, 3–19.
URL http://dx.doi.org/10.1016/j.entcs.2003.05.007

[2] Benveniste, A., Caspi, P., Edwards, S. A., Halbwachs, N., Guernic, P. L., Simone, R. D., 2003. The synchronous languages twelve years later. In: Proceedings of the IEEE. pp. 64–83.

[3] Borgne, M., Marchand, H., Rutten, r., Samaan, M., 1996. Formal verification of signal programs: Application to a power transformer station controller. In: Wirsing, M., Nivat, M. (Eds.), Algebraic Methodology and Software Technology. Vol. 1101 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 271–285.

[4] Bouali, A., 1997. Xeve : an esterel verification environment : (version v1_3). Tech. rep.
URL http://hal.inria.fr/inria-00069957

[5] de Simone, R., Ressouche, A., 1994. Compositional semantics of esterel and verification by compositional reductions. In: CAV. pp. 441–454.

[6] Halbwachs, N., Caspi, P., Raymond, P., Pilaud, D., 1991. The synchronous dataflow programming language lustre. In: Proceedings of the IEEE. pp. 1305–1320.

[7] Halbwachs, N., Lagnier, F., Raymond, P., 1993. Synchronous observers and the verification of reactive systems. In: AMAST. pp. 83–96.

[8] Halbwachs, N., Raymond, P., 1999. Validation of synchronous reactive systems: From formal verification to automatic testing. In: ASIAN. pp. 1–12.

[9] Jeannet, B., 2003. Dynamic partitioning in linear relation analysis: Application to the verification of reactive systems. Formal Methods in System Design 23, 5–37.

[10] Le Guernic, P., Gautier, T., Le Borgne, M., Le Maire, C., 1991. Programming Real-Time Applications with Signal. Proceedings of the IEEE 79 (9), 1321–1336.
URL http://hal.inria.fr/inria-00540460

[11] Marchand, H., Bournai, P., LeBorgne, M., Guernic, P. L., 2000. Synthesis of discrete-event controllers based on the signal environment. In: IN DISCRETE EVENT DYNAMIC SYSTEM: THEORY AND APPLICATIONS. pp. 325–346.

[12] Vardi, M. Y., Wolper, P., 1986. An automata-theoretic approach to automatic program verification (preliminary report). In: LICS. pp. 332–344.

[13] Weil, D., Bertin, V., Closse, E., Poize, M., Venier, P., Pulou, J., 2000. Efficient compilation of esterel for real-time embedded systems. In: Proceedings of the 2000 international conference on Compilers, architecture, and synthesis for embedded systems. CASES '00. ACM, New York, NY, USA, pp. 2–8.
URL http://doi.acm.org/10.1145/354880.354882