

Synchronous design of a transformer station controller with SIGNAL*

H. Marchand, E. Rutten
IRISA / INRIA - Rennes
F-35042 RENNES, France
e-mail: {hmarchan, rutten}@irisa.fr

Mazen Samaan
EDF/DER, EP, dept. CCC
6 quai Watier, 78401 CHATOU, France
e-mail: Mazen.Samaan@der.edf.fr

Abstract

This paper presents the specification and validation of the automatic circuit-breaking control system of an electric power transformer station. It handles the reaction to electrical defects on the high voltage lines. The purpose of this study is to construct a discrete event control system based on digital technology. To this end, we use the synchronous approach to reactive real-time systems, and in particular the data flow language SIGNAL, and its tools for specification, formal verification, simulation, and implementation. The hierarchical, state-based and preemptive controller is implemented with SIGNAL and its extension for preemptive tasks SIGNALGTi. A graphical simulator supports validation of the specification.

Keywords: power systems, discrete event systems, reactive systems, synchronous language.

1 Introduction

This paper presents the specification and simulation of the automatic circuit-breaking controller of an electric power transformer station. It is an experiment in the *synchronous approach* to the specification, implementation and formal verification of *reactive real time systems* [3]. We use the declarative language SIGNAL, applied to the design of the hierarchical, state-based, discrete event behavior of the controller of a power transformer station.

The context of the application is a power transformer station, such as the hundreds counted on the French national network operated by *Électricité de France* (EDF). Such a transformer station features an automatic control system handling the response to electric defects on the lines connected to it. The

control involves complex interactions between communicating automata, interruption and preemption behaviors, timers and timeouts, reactivity to external events, ... The functionality of the controller is to handle the interruption of power, the redirection of supply sources, and the re-establishment of power following an interruption. The objective of the controller is double: protecting the components of the transformer itself, and minimizing the effects of the defect in the distribution of power in terms of duration and size of the interrupted sub-network. The electric defects can be detected by sensors; the controller has to distinguish between several types of defects, and between transient and persistent ones.

We use the synchronous approach to reactive real-time systems, and particularly the data flow language SIGNAL. The synchronous languages are derived from theoretical and applied studies on discrete event systems with real time aspects, and on specification methodologies and programming environments for their development [3, 5]. They evolved into commercial products used in industrial settings [2]. Their aim is to support the design of safety critical applications, especially those involving signal processing and process control. The synchronous approach guarantees the determinism of the specified systems, and supports techniques for the detection of causality cycles and logical incoherences. A family of languages is based on this approach [5], featuring amongst other ESTEREL, LUSTRE, SIGNAL and also STATECHARTS. Among them, SIGNAL [6] is a data-flow language, with a declarative style: processes are systems of equations. The compiler transforms the specification into an optimized executable code (in C or FORTRAN). Verification of dynamical properties is based on polynomial dynamic systems over $\mathbb{Z}/3\mathbb{Z}$. SIGNALGTi is a recent extension that provides constructs for the specification of hierarchical preemptive tasks

*This work is supported by *Électricité de France* (EDF).

on intervals of time [11].

In this paper, we outline some results of a larger experiment on the specification and verification of the control system [8], by focusing on the specification of a particular aspect of its functionality.

2 The power transformer station

The transformer station on the network.

The French national electric network, operated by *Électricité de France* (EDF), counts a large number of transformer stations. Their purpose is to lower the voltage so that it can be distributed in urban centers to end-users [8], as illustrated in Figure 1. For each high voltage line, a transformer lowers the voltage. In the course of exploitation of this system, several kinds of electrical defects can occur, due to causes internal or external to the station. Three types of electrical defects are considered: phase (PH), homopolar (H), or wattmetric (W). In order to protect the device and the environment, several circuit breakers are placed in different parts of the station. These circuit breakers are alerted by sensors at different locations, and controlled by a local control system called *cell* (arrival cell, link cells, and departure cells) and by an operator in a remote control center.

Each circuit breaker controller defines a behavior beginning with the confirmation and identification of the type of the defect. If the defect is confirmed, the treatment consists in opening the circuit-breaker during a given delay, then closing it again, and after another delay, if the defect is still present, then repeating these operations for a certain number of cycles. The purpose of this is to treat transient defects; in case the defect is still present at the end of the cycle, the circuit-breaker is opened definitely, and the control is given to the remote operator.

One of the problems is to know which of the circuit breakers must be cut off. If the defect appears on the departure line, it is then possible to cut off the circuit breaker at departure level, or at link level, or at arrival level. Obviously, it is in the interest of users that the circuit be broken at departure level, and not at a higher level, so that the fewest users be deprived of power. This requires coordination between the different circuit breaker cells [12].

Functional description of a departure cell.

We will focus on one of the types of cell: the departure cell, because it features all the interesting aspects of the automatism behavior. Other cells have a behavior which is a subset of this one. It is decomposed

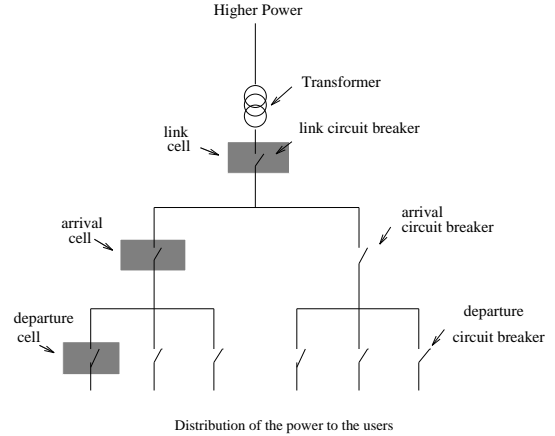


Figure 1: Topology of a power transformer station.

in a confirmation process, which also cares for identifying the type of the defect, followed by a treatment phase in an attempt at making the defect disappear.

The *confirmation phase* consists in taking the time to let transient defects cease naturally. For each of the defect types (PH, H, or W) a delay can assess its persistent presence. They are tested in sequence, until a defect is confirmed (i.e., present at the end of the corresponding delay). However, the sequence is interrupted as soon as the defect disappears, or one of the previously examined defect appears. This latter point involves a *preemption hierarchy*, as detailed in Section 4.

The *treatment phase* begins when the defect is confirmed. It alternates between breaking the circuit during varying delays, and closing it again to check whether the defect has disappeared. Each circuit break begins with emitting the command of opening of the circuit breaker, followed by the reception of the `Open` event, upon which the delay is fired. Upon completion of the delay, the closing of the circuit breaker is required, and confirmed by the reception of `Closed`. Once the circuit is re-established, either the defect has disappeared, and the cell goes into its normal state, or it is still present: then, after a .5s delay, the treatment phase goes into the next cycle (requesting opening, ...), or if it was the last cycle, the circuit breaker is definitely cut off, and its management is left to a remote control operator. This phase involves a *series of values* of the delay in the successive cycles, which consist in a repetition of the same task on a *series of activation intervals*.

In the following, Section 3 gives a brief overview of `SIGNAL`, and of its extension `SIGNALGTi`, introducing features that answer the need for series of values, and series of activation intervals found in the application.

Then Section 4 describes the detailed specification of the confirmation phase, and just a brief account of the rest of the controller for the sake of brevity.

3 The synchronous language SIGNAL

The SIGNAL equational language.

SIGNAL [6] is built around a minimal kernel of operators. It manipulates *signals* X , which denote unbounded series of typed values $(x_t)_{t \in T}$, indexed by time t in a time domain T . An associated clock determines the set of instants at which values are present. A particular type of signals called **event** is characterized only by its presence, and always has the value *true* (hence, its negation by **not** is always *false*). The clock of a signal X is obtained by applying the operator **event** X . The constructs of the language can be used in an equational style to specify the relations between signals *i.e.*, between their values and between their clocks. Systems of equations on signals are built using a composition construct. Data flow applications are activities executed over a set of instants in time. At each instant, input data is acquired from the execution environment; output values are produced according to the system of equations considered as a network of operations. Section 4 gives examples of their use.

Kernel of the SIGNAL language.

It is based on four operations, defining primitive processes or equations, and a composition operation to build more elaborate processes in the form of systems of equations.

Functions are instantaneous transformations on the data. For example, the definition of a signal Y_t by the function $f: \forall t, Y_t = f(X_{1t}, X_{2t}, \dots, X_{nt})$ is written in SIGNAL: $Y := f\{X1, X2, \dots, Xn\}$. The signals $Y, X1, \dots, Xn$ are required to have the same clock.

Selection of a signal X according to a boolean condition C is: $Y := X$ **when** C . If C is present and *true*, then Y has the presence and value of X . The clock of Y is the *intersection* of (*i.e.*, *included in*) that of X and that of C at the value *true*.

Deterministic merge noted: $Z := X$ **default** Y has the value of X when it is present, or otherwise that of Y if it is present and X is not. Its clock is the *union* of (*i.e.*, *includes* or *contains*) that of X and that of Y .

Delay gives access to past values of a signal. *E.g.*, the equation $ZX_t = X_{t-1}$, with initial value V_0 defines a *dynamic process*. It is encoded by: $ZX := X\$1$

with initialization $ZX \text{ init } V_0$. X and ZX have equal clocks.

Composition of processes is noted “ l ” (for processes P_1 and P_2 , with parenthesizing: $(l P_1 l P_2 l)$). It consists in the composition of the systems of equations; it is associative and commutative. It can be interpreted as parallelism between processes; instantaneous communication between them is carried by the broadcasting of signals.

Derived features and design environment.

Derived processes have been defined on the base of the primitive operators, providing programming comfort. *E.g.*, the instruction **synchro** $\{X, Y\}$ specifies that signals X and Y are synchronous (*i.e.*, have equal clocks); **when** B gives the clock of *true*-valued occurrences of B ; X **cell** B memorizes values of X and also outputs them when B is true; the expression $C := \# E$ is a counter of the occurrences of event E . Arrays of signals and of processes have been introduced as well. Hierarchy, modularity and re-use of the definition of processes are supported by the possibility of defining process models that can be invoked by instantiation.

The design environment features a block-diagram graphical interface, a formal verification tool based on the equational model of SIGNAL, and a compiler that establishes a hierarchy of inclusion of logical clocks (representing the temporal characteristics of discrete events), checks for the consistency of the inter-dependencies, and automatically generates optimized executable code ready to be embedded in environments for simulation, test, prototyping or the actual system.

Task preemption in SIGNALGTi.

A recent extension to SIGNAL handles tasks executing on time intervals and their sequencing and preemption [11]. Data flow and sequencing aspects are both encompassed in the same language framework, thus relying on the same model for their execution and analysis.

A *time interval* I has the value **inside** between the next occurrence of event B and the following occurrence of event E , and **outside** otherwise. It is written: $I :=]B, E[$ **init** I_0 with initial value I_0 (**inside** or **outside**). The operator **compl** I defines the complement of an interval I (**inside** when I is **outside** and reciprocally). The opening and closing occurrences of the bounding events are given by **open** I and **close** I . Occurrences of a signal X inside I are selected by X **in** I , and those outside by X **out** I (*e.g.*, **open** I is B **out** I , and **close** I is E **in** I).

Tasks consist in associating a process with an interval on which it is executed. Inside the task interval, the task process is active *i.e.*, present and executing, and out of it, it is absent and its internal state is u-

```

(| Defect := Ext_Defect default Def_Conf default (not (when not (PH or H or W)))
| I_PH := ] First_Defect, when Defect ]
| (| End_Delay_PH := when ((#Time) = Delay_PH)
  | Def_Conf := (when PH when End_Delay_PH) default (when PH in I_H) default Def_Conf_H
  | I_H := ] (when (not PH) when End_Delay_PH), Def_Conf_H ]
  | (| End_Delay_H := when ((#Time) = Delay_H)
    | Def_Conf_H := (when H when End_Delay_H) default (when H in I_W) default Def_Conf_W
    | I_W := ] (when (not H) when End_Delay_H), Def_Conf_W ]
    | (| End_Delay_W := when ((#Time) = Delay_W)
      | Def_Conf_W := when W when End_Delay_W
      |) each I_W
    |) each I_H
  |) each I_PH
|)

```

Figure 2: Code for the Confirmation process.

navailable (in some sense it is out of time, its clock being cut). A *suspensive task* is written P on I : it re-starts at its current state when re-entering I . An *interruptible task* is written P **each** I : it re-starts at its *initial state* (as defined by the declarations of its state variables). Processes can themselves be decomposed into sub-tasks: this way, the specification of *hierarchies* of preemptive behaviors is possible.

Task sequencing and preempting make it possible to specify general hierarchical parallel place-transition systems. *Sequencing tasks* is achieved by constraints on bounding events of task intervals (the end of the one equals the beginning of the other). *Parallelism* between several tasks is the composition of tasks sharing the same or overlapping intervals.

The encoding of time intervals and tasks into the SIGNAL kernel [10] is implemented as a pre-processor to the SIGNAL compiler, called SIGNALGTi [11]. Applications other than the one discussed in this paper concern an interactive reflex game and the sequencing of visual servoing tasks in a robot vision system [7].

4 Design in SIGNAL and SIGNALGTi

The confirmation phase: an interruption hierarchy.

Figure 3 illustrates the Confirmation process specified in SIGNALGTi in Figure 2. The three constant parameters $Delay_PH$, $Delay_H$, and $Delay_W$ correspond to each of the three kinds of electrical defects. The input event $Time$ is the base clock i.e., it is the clock of the logical inputs PH , H , and W (presence of the

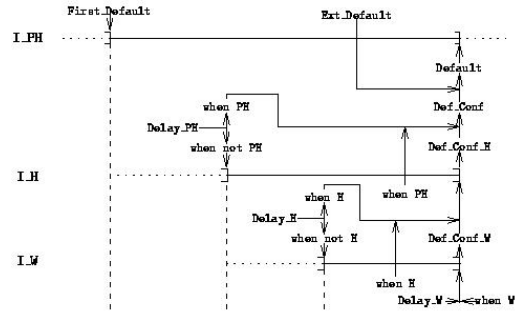


Figure 3: Confirmation phase: interruption hierarchy.

defects) and contains the clocks of the two other input events Ext_Defect and $First_Defect$. The process emits the output event Def_Conf when the defect is confirmed and the output logical signal $Defect$ which gives the state of the cell. This latter is *true* when an external defect is detected (reception of Ext_Defect) or when the defect is confirmed (Def_Conf), otherwise it is *false* when a defect is not present (i.e., when the disjunction of the three logicals is false). When a defect is detected ($First_Defect$), the interval I_PH is entered. It is closed by the occurrence of $Defect$ at the value *true*, causing the interruption of the confirmation task executed **each** I_PH (and also of its sub-tasks) until the next defect.

The interruption hierarchy, illustrated in Figure 3, is as follows. Each time I_PH is entered, a counter of $Time$ is fired during $Delay_PH$. At the end of this delay:

- If the logical PH is *true*, or if PH becomes *true* during the sub-interval I_H or if the defect is

confirmed at a lower level (`Def_Conf_H`), then the defect is confirmed at this level (`Def_Conf`). This causes `I_PH` to close, thereby terminating the confirmation task.

- In the other case (`PH` is `false`), `I_H` (sub-interval of `I_PH`) is entered, with a sub-task behaving in a quite similar way: a counter of `Time` is fired, and at `Delay_H`:
 - if `H` is `true`, or becomes so during interval `I_W`, or if the defect is confirmed at a lower level (`Def_Conf_W`), the defect is confirmed at this level (`Def_Conf_H`) causing the interval to close.
 - In the other case (`H` is `false`), `I_W` (sub-interval of `I_H`) is entered, and a last sub-task counts `Delay_W` and tests for `W`.

The *treatment phase* is another task [8], we will only sketch it here. Its sequencing with the confirmation phase is done simply by specifying that the latter is a task with an interval beginning on the occurrence of a defect confirmation `Def_Conf`, which also causes exiting the confirmation task (schematically: `Treatment each]Def_Conf,End_Treat]`). The specific feature of the treatment phase is its cyclical aspect: the same circuit-breaking procedure is applied starting with the request to open and ending with the reception of `Closed`; the only changing feature is the value of the delay fired on reception of `Open`. These two remarks suggest to have a signal carrying the series of values of the delay at a clock synchronous with `Open`, and to have a sub-task on a series of intervals, as in: `One_Cycle each]Req_Open,Closed]`.

Validation by graphical simulation.

We validated the specification by graphical simulation for the aspects of the functionality involving the correct scheduling of events, . . .

The *simulation environment* is built using a generic built-in graphical simulation tool for SIGNAL specifications. It performs the automatic construction of graphical input acquisition buttons and output display windows, for the signals of the interface of a program, in an oscilloscope-like fashion, as illustrated in Figure 4. We want to display the presence and values of some of the intervals of the behavior, and of some events. In order to display them on oscilloscope-like windows, we embed the controller into a process encoding them in integers present at every instant. Intervals are encoded as 1 when `inside`, `-1` when `outside` and 0 when absent. Events are encoded as 1 when present (which lasts only one instant and appears graphically as a peak), and 0 when absent. The

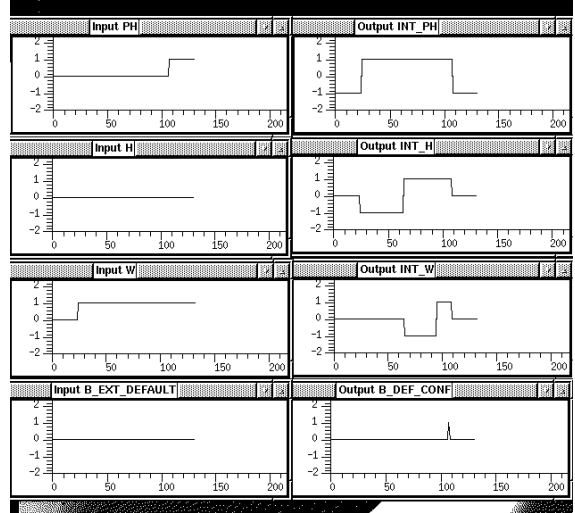


Figure 4: Simulation of the confirmation phase.

input logicals are always present, and are displayed as 1 when `true` and 0 when absent.

The *simulation of the confirmation phase* is illustrated in Figure 4. Its left column describes the trace of the different inputs (i.e. the three kinds of logical defects `PH`, `H` and `W`, and the event `Ext_Defect`). The right column shows the intervals of the hierarchical tasks of the confirmation phase, and output event `Def_Conf`. In the particular simulation trace illustrated in Figure 4, the first event occurs at time 20: the logical input `W` becomes `true`. Consequently, the interval `Int_PH` (corresponding to `I_PH`) is opened, and `Int_H` is in its initial state `outside`. At the end of `Delay_PH` (for this simulation: 40) i.e. at time 60, `open Int_H` occurs, and the interval `Int_W` is in its initial value `outside`. At the end of `Delay_H` (for this simulation: 30) i.e. at time 90, `open Int_W` occurs. Before the end of `Delay_W`, at time 110, the defect `PH` becomes `true`: it causes interruption of the confirmation on other defects, and emission of `B_Def_Conf`.

Validation by verification.

Although there is no space for a detailed presentation in this paper, we can give pointers to and a very brief outline of the formal verification technique associated to SIGNAL [6, 4] and its use in the case of this study [8]. Let us first note that the compiler itself, with its calculus on clock relations (inclusion, . . .), is a proof tool for statical properties i.e., those that hold for all instants (e.g. “*the two events E1 and E2 can never occur together*”).

The verification of dynamical properties uses a model-checking technique based upon the equational model of SIGNAL in polynomial dynamic systems over $\mathbb{Z}/3\mathbb{Z}$. The proof method is based on the theo-

ry of algebraic geometry: the systems of polynomial equations characterize sets of solutions, which are states and events. The techniques used in the method consist in manipulating the equation systems instead of the solutions sets, which avoids the enumeration of the state space. Operations involving ideals, varieties, and morphisms are used to define liveness and safety properties, the reachability of a set states, the invariance of a set of states by transitions, its invariance under control, and its attractivity. With this, one can express and evaluate properties like: “*the state where X is true is reachable from the initial state of the system*”.

5 Conclusion

This paper presents the synchronous approach to the specification and validation of discrete event control systems, applied to the preemptive controller of a circuit-breaker for a power transformer station.

The specification, validation and implementation of complex control systems, implying permanent interaction with an environment, is treated by the data-flow language SIGNAL in a discrete event system framework. The possibility of formal verification makes it particularly suited for safety-critical applications. Transitions between different modes of such an activity, i.e. the sequencing of hierarchical data-flow tasks, is the purpose the extension SIGNALGTi [10]. It makes a language-level integration of the data flow and task preemption frameworks. This way, the whole application can be specified in SIGNAL, from the discrete event driven state-based behavior down to the servoing loops. The SIGNAL approach to the specification and verification of control systems has also been experimented on other applications, such as a robotic production cell [1], a speech processing system [6] or a robotic vision system [7]. The perspectives around the SIGNAL approach concern formal methods, hybrid systems, hardware-software co-design, distributed implementation, etc ...

Among these perspectives, the automatic synthesis of discrete event controller systems is also studied: given constraints between the different signals and tasks, an automaton-like behavior can be synthesized automatically [4, 9]. This technique will be applied to the synthesis of a controller of the interactions between communicating cells in a transformer station.

References

- [1] T. Amagbegnon, P. Le Guernic, H. Marchand, and E. Rutten. Signal – the specification of a generic, verified production cell controller. *Formal Development of Reactive Systems – Case Study Production Cell*, volume 891 of *Lecture Notes in Computer Science*, Chapter VII, pages 115–129, January 1995.
- [2] A. Benveniste. Synchronous languages provide safety in reactive systems design. *Control Engineering*, pages 87–89, September 1994.
- [3] A. Benveniste and G. Berry. Real-time systems designs and programming. *Proceedings of the IEEE*, 79(9):1270–1282, September 1991.
- [4] B. Dutertre and M. Le Borgne. Control of polynomial dynamic systems: an example. Research Report 798, IRISA, January 1994.
- [5] N. Halbwachs. *Synchronous programming of reactive systems*. Kluwer, 1993.
- [6] P. Le Guernic, M. Le Borgne, T. Gautier, and C. Le Maire. Programming real time application with SIGNAL. *Proc. of the IEEE*, 79(9):1321–1336, September 1991.
- [7] E. Marchand, E. Rutten, and F. Chaumette. Applying the synchronous approach to real time active visual reconstruction. Research Report 2383, INRIA, Oct. 1994.
- [8] H. Marchand, E. Rutten, and M. Samaan. Specifying and verifying a transformer station in signal and signalgti. Research Report 2521, INRIA, March 1995.
- [9] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE; Special issue on Dynamics of Discrete Event Systems*, 77(1):81–98, 1989.
- [10] E. Rutten and P. Le Guernic. The sequencing of data flow tasks in SIGNAL. In *Proceedings of the ACM SIGPLAN Workshop on Language, Compiler and Tool Support for Real-Time Systems*, Orlando, Florida, June 1994.
- [11] E. Rutten and F. Martinez. Signalgti: implementing task preemption and time intervals in the synchronous data flow language signal. In *Proc. of the 7th Euromicro Workshop on Real Time Systems*, Odense, Denmark, Juin 14 - 16, 1995.
- [12] M. Samaan, F. Borgards, and N. Bergé. Application of a synchronous programming approach: development of an experimental distribute transformer station control system. (*to appear*).