# Systematic Derivation of a Validation Model from a Rule-oriented Model : A System Validation Case Study using Promela/Spin

J. Christian Attiogbé
LINA FRE CNRS 2729 - Université de Nantes
Christian.Attiogbe@lina.univ-nantes.fr

**In Proceedings IEEE ICTTA'04, ISBN 0-7803-8483-0**

**Abstract**

*We propose an approach to validate a rule-oriented specification by a systematic embedding of this kind of specification into a* Promela *validation model. The approach is illustrated with a case study about a nuclear power station cooling system. We start with an initial logical specification given as a rule-oriented model of the system. The study focusses on the systematic derivation of a* Promela *validation model from the initial model and then its validation. The rule-oriented model is matched with a* Promela *model on the basis of a control-oriented architecture. The obtained model is complemented by correctness properties and is then validated with respect to these properties using the* Spin *tool.*

## 1. Introduction

We propose an approach to validate a logical rule-oriented model by a systematic derivation of the associated Promela[5] validation model. The derivation is done on the basis of a control-oriented system architecture which is carefully built. The obtained model is increased with correctness properties (safety and liveness properties) and is then validated with respect to these properties using the Spin tool [5], [6]. Spin is a model checker and more generally a verification tool that supports the design and verification of asynchonous processes. Promela is the input language of Spin. A Promela model is a CSP-like description of communicating asynchronous processes. Promela/Spin have been used for numerous studies [4], [10], [7], [9].

The interest of our proposal is that an initial logical model can be systematically reused for the validation instead of specifying a particular state machine model. This paper presents the approach and its application to an industrial case study about a nuclear power station cooling system.
The case study is taken from [3] where the authors present the system and give a logical specification. The cooling system has to supervise the cooling process –using water– of a fluid circulating with a constant delivery. The process is depicted in the Figure 1. The water used for cooling is drawn from a reservoir by a circulation pump with its associated floodgate. A feed pump is used to supply the reservoir with water via a floodgate. The cooling system must detect malfunctions and consequently recommend operations to control the system. The recommended operations are assumed to be executed by a human operator. A detected malfunction persists until the appropriate treatment is perfomed. Therefore the cooling system should help the operator to maintain the system in a good operating condition. The systematic aspect of the approach is very important for reuse purpose.

The paper is organized as follows. We present the cooling system in Section 2. In Section 3, we introduce the initial logical model. In Section 4, we show the systematic approach used to derive the validation model from the initial model. In Section 5, we describe the correctness verification of the obtained model and the experimental results. In Section 6 we draw some conclusions and present future work.

## 2. The Cooling Process and System

The cooling process is made of two parts: the first is a water reservoir with associated pump (*pump2*) and floodgate (*fg2*) to feed it; the second part is the cooling process itself; it uses water from the reservoir and makes it circulate to cool the fluid. This part is also equipped with a pump (*pump1*) and a floodgate (*fg1*). The cooling system is the control system which supervises both parts. However a particularity of the current cooling system is that it doesn't avoid *bad states* (those in which malfunctions are detected) but it detects bad states and recommends to an operator the adequate operations to correct them. The cooling system interacts with a human operator by indicating to him the state of the process and the recommended operations to be performed. All recommended operations are not executed by the operator. He can decide which one is important and give *execution orders* to the system. A supervision part is about the water reservoir level. The system should adopt an appropriate reaction if the water level is out of some given thresholds. Two all-or-none sensors (`low-wlvl, high-wlvl`) are used to detect respec-

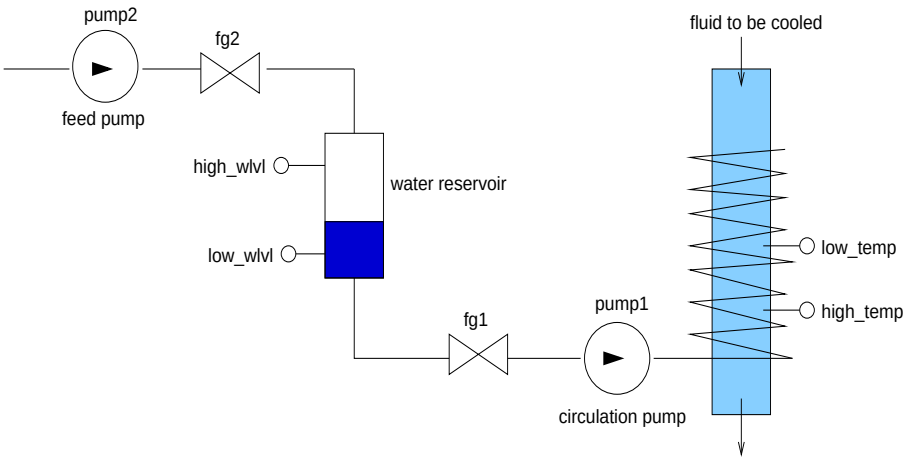


Fig. 1. The Cooling process

tively too low water level and too high water level of the reservoir. The other part of the supervision is about the temperature of the liquid being cooled. Two all-or-none sensors (`low-temp, high-temp`) are used to detect the overstepping of the liquid low and high temperature. The system should help the operator to maintain the temperature between two given thresholds. Then if the temperature is too high the circulation pump *must be engaged*; if the temperature is too low, the circulation pump *must be disengaged*. This corresponds to recommended operations. The operator can give orders to the cooling system from two control panels: a main panel (MP) and a rescue panel (RP). A pump engagement order (EO) can be issued from a panel under certain conditions (too high temperature of the liquid, lack of water in the reservoir, etc). A disengagement order (DO) can also be issued from the panels.

## 3. The Initial Logical Model

It takes the form of several implication rules. These rules relate the system variables. We classify them into input, state and output variables. An example rule is the following (adapted from [3]) :

$$\begin{aligned} R7 \quad & (pump1\_eo\_mp \ \vee \ pump1\_eo\_rp) \ \wedge \\ & \neg \ lack\_power \\ & \wedge \ \neg \ (pump1\_do\_mp \ \vee \ pump1\_do\_rp \ \vee \\ & \qquad fg1\_closed \ \vee low\_wlvl) \\ & \Rightarrow \quad pump1\_ego \end{aligned}$$

It expresses that *the conditions for global order of circulation pump engagement are: an engagement order from main or rescue panel, electric power supply, no disengagement command from main or rescue panel, reservoir level not low and floodgate open.*

Here are the description of the variables used in the initial model.

`lack_power` : Lack of power supply (380 Volts).
`low_temp` : Low temperature threshold of the fluid.
`high_temp` : High temperature threshold of the fluid.
`pump1_en` : Circulation pump (`pump1`) engaged.
`pump1_av` : Circulation pump available.
`low_wlvl` : Low water level threshold.
`high_wlvl` : High water level threshold.
`pump2_en` : Feed pump (`pump2`) engaged.
`pump2_dis` : Feed pump disengaged.
`fg1_closed` : Circulation floodgate (`fg1`) closed.
`fg2_closed` : Feed floodgate (`fg2`) closed.
`pump1_do_mp` : Circulation pump disengagement order from MP.
`pump1_do_rp` : Circulation pump disengagement order from RP.
`pump1_eo_mp` : Circulation pump engagement order from MP.
`pump1_eo_rp` : Circulation pump engagement order from RP.

The other rules of the model have the same shape :

$$\begin{aligned} R1 \quad & (\neg \ pump1\_av \vee low\_wlvl) \ \wedge \ high\_temp \\ & \Rightarrow \ htemp\_unav\_circul \end{aligned}$$

The conjunction of the unavailability of circulation and a too high temperature of the fluid results in a detectable malfunction (named *htemp_unav_circul*).

$$R2 \quad (\neg \ pump1\_av \ \wedge \ low\_wlvl) \ \Rightarrow \ unav\_circul$$

There is unavailability of the circulation if the pump is unavailable and the water level is too low.

$$R3 \quad (pump1\_en \ \wedge \ low\_temp) \ \Rightarrow \ ex\_circul$$

There is excess of water circulation if the circulation pump is engaged and the liquid temperature is too low.

$$R4 \quad (pump2\_en \ \vee \neg \ fg2\_closed) \ \Rightarrow \ pump2\_ego$$

Global order for engaging feed pump should be given if the feed pump is engaged or its floodgate is open.

$R5 \quad (pump2\_dis \vee fg2\_closed) \Rightarrow pump2\_dgo$

Feed pump global disengagement order should be given if the pump is disengaged or its floodgate is closed.

## 4. The Validation Model in Promela

Our experiment confirms that the validation model can be completely derived from a control-oriented architecture. Therefore a main stage is to elaborate a suitable control-oriented architecture wich emphasizes input and output variables. These variables will serve to describe the states needed for the validation model: that is one part of the validation model. The links between input and output variables will serve to determine the remaining (dynamic) part of the validation model. This involves an architecture of a reactive system [8] with inputs from its environment and outputs to this environment.

The architecture is depicted in the Figure 2. It enables us to completely separate the specification of the process to be cooled from the cooling system. Each part can be treated in more details. On the other hand we consider the two main parts of a Promela model: a static part made of state variable declarations and a dynamic part made of several asynchronous processes communicating via channels. Then we match the reactive system architecture with the Promela model.

The system architecture highlights interacting and concurrent processes which represent :

- sensors (`low_wlvl_sensor`, `high_wlvl_sensor`, `high_temp_sensor`, `low_temp_sensor`),
- controlled processes (`floodgate1, floodgate2, pump1, pump2`) and
- the cooling system (`CoolingSystem`).

From the logical model we extract input variables describing input informations and output variables of the system. The outputs of the system are the recommended operations and the states of components. The derivation work is divided in two parts, a static part to deal with the global system modelling and a dynamic part to deal with the behaviour of communicating processes. In Promela, *process types* are used to describe processes. A *process type* is the description of a behaviour using statements (arithmetic and logical expressions, sequences, repetition, conditional, guards, etc).

### 4.1. Capturing the Static Part

The system reacts on input informations to supervise the cooling process behaviour. Therefore, the logical model rules express by their left hand sides the conditions of some malfunctions or the conditions under which some actions (order execution by the operator) are undertaken. The rule right hand sides are considered as output variables (including recommended operations and detected malfunctions). Then we classify the logical variables into input, output and state variables.

**Input Variables** Mainly, they are input informations from the sensors. The other inputs are the orders given by the operator.

We reuse the logical propositions with associated variable names to describe the inputs of the process. The input variables are: `lack_power`, `low_temp`, `high_temp`, `low_wlvl` and `high_wlvl`.

These variables are modelled as Promela variables and they have been given Boolean type (here a bit):

```
bit  low_temp, high_temp;  /* from sensors*/
bit  low_wlvl, high_wlvl, lack_power;
```

If the operator engages an action by an order, the associated variable is set to *true* otherwise it remains *false*. The orders given by the operator from the main panel and the rescue panel have the value EO (engagement order), DO (disengagement order), EGO (engagement general order), DGO (disengagement general order). The associated variables are *order_pump1_mp*, *order_pump1_rp* and *order_pump2_mp*.

**State Variables** They are used as input and output informations. We stay at the same abstract level as in the logical model. The system state is described by (propositional) variables which represent the system component states. On the one hand, there are *pump1* and *pump2* which represent respectively circulation pump states and feed pump states. They have the values ENGAGED, DISENGAGED, AVAILABLE and are modelled with the `byte` type. On the other hand there are *fg1* and *fg2* which represent respectively circulation floodgate states and feed floodgate states. They have the values CLOSED, OPEN and are modelled as `bit`.

**Output Variables** Output variables (including recommended operations) are calculated from input variables and state variables. Some output variables indicate malfunctions of the system. These malfunctions have to be solved. They are named in the sequel: *unav_circul*, *ex_circul*, *htemp_unav_circul*, *pump1_el1* and *pump1_el2*. These variables have the value TRUE or FALSE, they are Boolean (modelled with the type `bit` of Promela). If the system detects a malfunction the corresponding variable is set to *true* otherwise the variable is set to *false*.

To summarize, we have described on the basis of the system architecture, the static part of the validation model, using input, output and state variables. These variables are extracted from the logical rules. The input variables are input informations

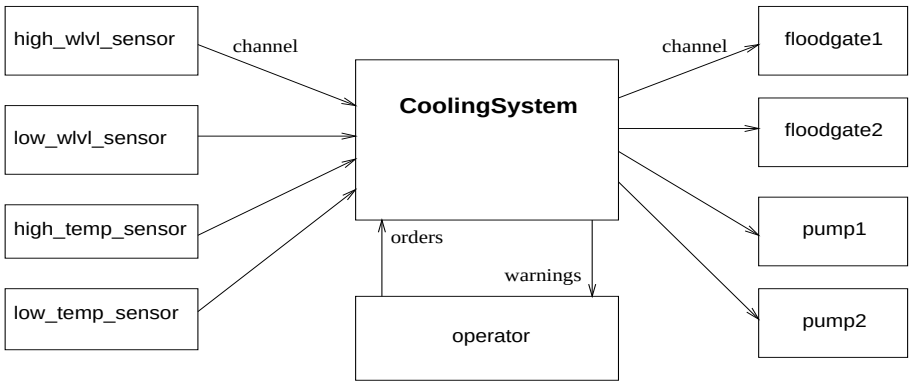


Fig. 2. The System architecture

from the sensors and orders given by the operator. The right hand sides of rules are considered as output variables (including recommended operations and detected malfunctions). The output variables are calculated from input variables and state variables. Some output variables indicate malfunctions of the system. The state variables are those used as input and output informations.

### 4.2. Capturing the Dynamic Part of the Model

The cooling system should scrutinize current state and inputs, detect malfunctions, and help to solve the detected malfunctions by interacting with the operator. This implies three steps presented below. Consequently, the logical model rules are used to detect malfunctions and to calculate recommended operations. The associated Promela *process type* has a cyclic behaviour. In each cycle, we distinguish three steps: detecting malfunctions, trying to solve malfunctions, reading inputs from operator and sensors.

#### Step 1: Detecting Malfunctions

Each logical rule ($R_i$):

$malfunction\_conditions \Rightarrow malfunction\_name$

is captured in the Promela model as follows:

```
if
::(malfunction_conditions) ->
            malfucntion_name = TRUE;
:: else -> malfunction_name = FALSE;
fi
```

where all the branches of the conditional structure are guarded statements ($guard \rightarrow statement$).
For each output variable (called *malfunction_name*) appearing in the right hand side of a rule and identifying a malfunction on the cooling process, we have an associated conditional statement in the Promela model of the cooling system. Applying this transformation to the other rules ($R1 - R9$) we get a part of the model of the cooling system. Let us consider the first rule ($R1$) as an example. This rule can be divided into two sub-rules. The first (R1-a) indicates that circulation is unavailable (`unav_pump1_low`) if either the circulation pump (*pump1*) is unavailable or the reservoir level is too low. The second sub-rule (R1-b) indicates that circulation pump is unavailable and the liquid temperature is too high.

```
if  /* R1-a */
:: (pump1 != AVAILABLE)
        || (low_wlvl == TRUE)
   -> unav_pump1_low = TRUE;
:: else -> unav_pump1_low = FALSE;
fi;
if /* R1-b */
:: (unav_pump1_low == TRUE)
        && (high_temp == TRUE)
   -> htemp_unav_circul = TRUE;
:: else ->  htemp_unav_circul = FALSE;
fi
```

This part represents one working step in the cycle of the cooling system.

#### Step 2: Solving Detected Malfunctions

The system must indicate to the operator appropriate operations to correct detected malfunctions. To capture the corresponding behaviours, we examine the cause of each malfunction, that means the left hand side of the corresponding rule. To solve the malfunction consists in modifying the process such a way that this left hand side becomes false, and then if the left hand side is false (the malfunction has disappeared) the right hand side is set to false.

To maintain the system in good operating condition, the operator handles malfunctions by executing operations indicated by the system.

Solving a malfunction results in setting some variables in such a way that it leads to the engagement of the appropriate order. The logical rules are exploited for this purpose. For example $R7$ is used to indicate a general engagement order of the circulation pump (*pump1*). The Promela specification is:

```
if /* R7 */
:: ((  (order_pump1_mp == EO)
     ||(order_pump1_rp == EO))
   && (lack_power == FALSE)
   && (fg1 != closed) && (low_wlvl == FALSE)
   ) ->  syspannel!order(pump1_ego);
:: else -> skip;
fi
```

**Step 3: Reading and Treating Inputs**

Input informations are passed through channels between processes. A channel enables the operator to send orders to the system. Thus, from this channel the cooling system receives several orders and treats them.

**The Model of the Cooling System**

The Promela model associated to the cooling system is a cyclic process (an iterative structure in Promela). Each cycle has the three steps explained above (**Step1, Step2, Step3**).

```
proctype CoolingSys()
{
start_cycle : cycle_step = NOTCHECKED;
/* step 1: the rules for
      reading orders and input events */
/* step 2: the rules for
      detecting malfunctions          */
/* step 3: the rules for
      solving malfunctions detected   */
end_st : cycle_step = CHECKED;
goto start_cycle;
}
```

In this section, we have complemented the static part with process behaviours which model the different components of the system. Now we focus on the validation of this model. The remaining steps of the approach are about the introduction of correctness properties and the validation of the obtained model.

## 5. Validation of the Model

The Promela language offers some constructs to deal with property verification: *assertions, progress labels, acceptance labels* and *never claims* [5]. We use *assertions* and *never claims* in our experiment to capture the properties of the system. The `assert` construct enables the developper to check *invariants*: properties that should always be true. The `never claims` mechanism is a more expressive construct used to detect undesirable or illegal behaviours. The Spin tool is then used to verify the model complemented with these properties.

### 5.1. The Properties of the System

We consider safety properties specified using the `assert` construct and liveness properties expressed by the `never` construct. Examples of the properties are given below:

$P_1$: *We cannot have sensors* `low_wlvl` *and* `high_wlvl` *simultaneously detecting a low and a high level of water in the reservoir.*

$P_2$: *In the same way, we cannot have* `low_temp` *and* `high_temp` *simultaneously.*

$P_3, P_4$: *pumps can not be engaged if associated floodgates are closed.*

These safety properties expressed with `assert` are introduced as shown above:

```
#define NOT_HIGH_AND_LOW
        (!low_temp || !high_temp)
#define NOT_LOW_AND_HIGH_TEMP
        (!low_temp || !high_temp)
#define ENGAGED_IF_FG_PUMP1
 ((fg1 == closed) || (pump1 == DISENGAGED))
#define ENGAGED_IF_FG_PUMP2
 ((fg2 == closed) || (pump2 == DISENGAGED))

(cycle_step == CHECKED) ->
   assert(NOT_HIGH_AND_LOW
           && NOT_LOW_AND_HIGH_TEMP
           && ENGAGED_IF_FG_PUMP2
           && ENGAGED_IF_FG_PUMP1)
```

We introduced a particular process which role is to always check this assertion as a system invariant. These properties are checked at the end of each cycle; for this purpose a variable `cycle_step` is modified at the beginning and at the end of each cycle. For liveness, the properties are first specified using LTL formula and then translated into *never claims* mechanism with the Spin translator. The Spin translator gives the Büchi automata corresponding to the LTL formula [6]. The properties of the system are similar and quite simple. In general we have to prove that if a malfunction appears then it will be solved. Some of these properties follow:
$P_5$: *if a too high temperature of the fluid is indicated then the circulation pump will be activated.*
$P_6$: *if a too low water level is indicated then the feed pump will be engaged.*

The associated LTL formula are shown in the following table, where the propositions used are predefined `pump1_engaged` stands for `(pump1 == ENGAGED)`, `h_temp` stands for `(high_temp == TRUE)`, `pump2_engaged` stands for `(pump2 == ENGAGED)` and `l_temp` stands for `(low_temp == TRUE)`.

| Property | LTL formula |
|---|---|
| $P_5$ | `[](h_temp -> <> pump1_engaged)` |
| $P_6$ | `[](l_wlvl -> <> pump2_engaged)` |

The *never claims* associated to these properties (the negated form) are joined to the validation model. Indeed we specify the desired properties and their negated forms are used as errors (undesirable states).

### 5.2. Validation and Results

Validation involves ensuring that the model satisfies certain correctness properties. Given the Promela model with associated properties described by `assert`

```
(never claims generated from LTL formulae are stutter-closed)
(Spin Version 3.4.1 -- 15 August 2000)
+ Partial Order Reduction

Full statespace search for:
never-claim          +
assertion violations + (if within scope of claim)
cycle checks         - (disabled by -DSAFETY)
invalid endstates - (disabled by never-claim)

State-vector 132 byte, depth reached 175, errors: 0
    4453 states, stored
   58228 states, matched
   62681 transitions (= stored+matched)
    3385 atomic steps
hash conflicts: 228 (resolved)
(max size 2^19 states)

3.054  memory usage (Mbyte)
```

Fig. 3. Verification report for P5

and `never`, we verify the correctness of the model using Spin. Spin explores the state space of the system, if an error occurs then it stops the exploration. Since *never claims* are used to detect undesirable states, it is required for a valid model to have zero errors after a full state space search.

The report above 3 presents the result of the $P_5$ property verification. The other properties are verified in the same way. In this case of full state search, there were 4453 generated system states; the longest non-cyclic execution sequence depth is 175, and no error is detected.

## 6. Concluding Remarks

Verification techniques are increasingly used to ensure correctness of concurrent reactive systems, especially critical systems. Spin is one among a lot of model checking tools devoted to verification. Spin is selected for this study for several reasons. Its input language Promela is adequate for the specification at hand, and for property specification. More generally the quality of the software tool (equipped with graphical interfaces which greatly help the user) is appreciate. In this study we have successfully used Spin for the verification of the cooling system. To build the system model we started with a logical model as an abstract specification and we extracted systematically the Promela model. The extraction was achieved by distinguishing a static part where we capture all information of the system with input, output and state variables and a dynamic part where the rules of the logical model are systematically used. This approach can be used in similar cases and enables one to use a simple logical model to begin the analysis of a reactive system behaviour and then to derive a validation model for the formal treatment of the system. To validate our model, we describe and specify safety and liveness properties of the system and check these properties. Some errors in the model, especially on state variables, have been corrected through the validation step. We have also simulated the system behaviour for some scenarios and the results are satisfactory. Current works focus on the automation of the process to transform some rule-based specifications into Promela models for formal analysis purpose. We may benefit from numerous existing works [2], [1] on the translation of other input formats into Spin/Promela.

## References


[1] D. Bosnački, D. Dams, L. Holenderski, and N. Sidorova. Model Checking SDL with Spin. In S. Graf and M. Schwartzbach, editors, *Proc. of the 6th International TACAS Conference*, volume 1785 of *Lecture Notes in Computer Science*, Berlin, 2000. Springer-Verlag.

[2] A. Browne, H. Sipma, and T. Zhang. Linking STEP with SPIN. In Klaus Havelund, John Penix, and Willem Visser, editors, *Proc. of the 7th International SPIN Workshops*, volume 1885 of *Lecture Notes in Computer Science*, Stanford, September 2000. Springer-Verlag.

[3] M. Gondran, J-F. Héry, and J-C. Laleuf. *Logique et modélisation*. Coll. DER - EDF. Eyrolles, ISSN 0399-4198, 1995.

[4] K. Havelund, M. Lowry, and J. Penix. Formal Analysis of a Space Craft Controller using Spin. In *Proc. of SPIN'98 Conference*. Paris, France, 1998.

[5] G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, Englewoods, Cliffs, 1991.

[6] G. J. Holzmann. The Spin Model Checker. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

[7] P. Maggi and R. Sisto. Using SPIN to Verify Security Properties of Cryptographic Protocols. In D. Bošnački and S. Leue, editors, *Proc. of the 9th International SPIN Workshops: Model Checking Software*, volume 2318 of *Lecture Notes in Computer Science*, Grenoble, France, April 2002. Springer-Verlag.

[8] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems*. Springer, 1995.

[9] J.S. Pascoe, R.J. Loader, and V.S. Sunderam. The Agreement Problem Protocol Verification Environment. In D. Bošnački and S. Leue, editors, *Proc. of the 9th International SPIN Workshops: Model Checking Software*, volume 2318 of *Lecture Notes in Computer Science*, Grenoble, France, April 2002. Springer-Verlag.

[10] T. Ruys and R. Langerak. Validation of Bosch' Mobile Communication Network Architecture with Spin. In *Proc. of SPIN'97 Conference*. Twente University, The Netherlands, 1997.