# Architecture assessment for safety critical plant operation using reachability analysis of timed automata

David Gouyon, Jean-François Pétin, Thomas Cochard, Catherine Devic

# Architecture assessment for safety critical plant operation using reachability analysis of timed automata

David Gouyon[a,*], Jean-François Pétin[a], Thomas Cochard[a], Catherine Devic[b]

[a] *Université de Lorraine, CNRS, CRAN, F-54000 Nancy, France*
[b] *EDF (Electricité de France), R&D, F-78400 Chatou, France*

## Abstract

This article deals with the validation of critical industrial process architectures from the point of view of safety and operation. During the engineering phases, the objective is to complement conventional safety studies with an approach that focuses on plant operation. In this context, one of the major challenges is to provide a guarantee that the designed architecture will be able to react safely to critical situations and events.

To face the complexity resulting from the large number of functionalities and devices of the installations under consideration, the proposed approach is based on dynamic models of architectures, using the formalism of timed automata and reachability analysis to verify that, given a particular configuration of an architecture, the process can be safely operated to achieve a given objective. The result is a formal tool that allows engineers and plant operators to evaluate architecture safety with different types of dysfunctional scenarios based on their operational safety expertise.

The article presents the formal modelling framework, which emphasizes structured modelling using patterns to promote reuse and instantiation over several candidate architectures. The contribution is illustrated and discussed using an experimental laboratory platform.

*Keywords:* Plant operation, safety assessment, critical process architecture, reachability analysis, timed automata, modelling patterns

## 1. Introduction

Industrial processes involving field devices such as transmitters and actuators that are controlled, monitored and operated manually by human operators, numerical systems and controls, and control and data acquisition systems (SCADA) are complex systems. Their architectures make extensive use of func-

[*] Corresponding author
*Email address:* david.gouyon@univ-lorraine.fr (David Gouyon)

tional and physical redundancies, as well as exclusions, to ensure safety.

In addition to conventional safety studies (qualitative assessment of accident scenarios, RAMS forecasts, etc.), the operational availability and safety of plant operating scenarios must be assessed. Given the large number of possible scenarios, due to the redundancies and complexity of industrial installations, operational validation is rather carried out *a posteriori* on the basis of architecture proposals. This is dealt with during the engineering phases but must include feedback from human operators who must deal, day after day, with the various situations of danger and operation of the entire system. Indeed, some operating scenarios using redundancies may be imagined during the engineering phase but considered inappropriate or even prohibited by operators' practices [1].

The work presented in this document was supported by the Connexion project (French Excellence in Nuclear Control Systems). The document proposes a formal approach based on a dynamic process model to help designers and operators perform an automatic analysis of the reachability of situations. The objective is to prove that an architecture provides at least one solution to operate the system safely in order to achieve a targeted situation from a current situation. The added value of operators lies in the provision of significant initial and targeted situations and in the analysis of safety based on practical considerations.

Finally, several candidate architectures can be evaluated during the engineering phase and, therefore, several safety models must be built. A practical consideration of the work for reducing resource consumption, in terms of manpower and time, is to promote modularity and reuse of models.

This article is organized into six sections. The following section presents the industrial context and the problem. The section 3 provides a brief overview of the related works. The section 4 provides an overview of the proposed approach to architecture evaluation, including the selected formalism of timed automata and the generation of action sequences. The modular pattern-based modeling framework is then detailed in the section 5 devoted to architecture and sequencer modeling. The proposed approach is illustrated and discussed using a case study in the section 6.

## 2. Industrial context and problem

This article focuses on large industrial processes involving thousands of field devices (transmitters and actuators) to perform a variety of functions. These devices can be manually operated and monitored by human operators, automated reflex control systems and SCADA systems. The control of the plant consists in defining an ordered sequence of control and monitoring tasks to be applied to the devices (opening or closing of valves, consignment of devices, etc.), in order to manage the evolutions of the plant. In other words, from a current situation including the values of the process physical variables (such as temperature or pressure...), the state of the devices (such as idle or locked...), the state of the functions (filling of a tank in progress, pressurization of a fluid circuit...) and the process (reactor shutdown, rated production...), the objective is to find an appropriate and safe sequence of actions to achieve a predicted process situation. Such a sequence must satisfy logical (e.g.: priority rule), temporal (e.g.: minimum time to operate a device) and physical (e.g.: threshold required for physical values) constraints.

In the case of critical process systems, plant operation is generally under the control and validation of human operators who refer to predefined and qualified procedures. This activity is complex because architectures massively use functional and physical redundancies, as well as exclusions to ensure security. This complexity makes it difficult to assess, from a safety perspective, all possible operational scenarios [2], predict and control possible interference between devices or functions, and anticipate blockings.

During the engineering phase, the safety assessment aims to demonstrate that the plant has sufficient redundancies to deal with any critical situation. More precisely, the process architecture is designed in such a way that there is at least a safe sequence of actions that an operator can activate, from a current process situation, to achieve the expected state and status of the process. This validation activity can be performed manually by experienced operators but its level of confidence is questionable because it seems quite impossible for operators to consider all potentially possible scenarios [3, 4, 5].

The industrial problem is then to validate the architectures with regard to safety and operational requirements using formal model-based approaches. This activity is carried out from the early stages of architectural engineering and places the human operator ”in the loop” to confirm the operational availability and safety of the designed architectures. It requires helping engineers and operators to perform an automatic analysis of the reachability of situations. In other words, the objective is to generate an acceptable sequence of actions to demonstrate that the process can be safely operated to reach a planned situation. If the situation is not reachable, it can lead to a redesign of the functional and/or organic architecture of the system.

## 3. Related works

According to the industrial requirements of section 2, the qualitative evaluation of architectures is assumed to be based on demonstrating the existence of a safe sequence of actions to reach a target state from a source state. Existing approaches can be classified into two categories: static models describing the plant structure (Boolean models, graph-based models, etc.) and dynamic models whose system behaviour is characterized by a state space.

Typically, static analysis of architecture safety takes into account the relationships between a combination of faulty devices and the occurrence of a dangerous event. The most frequently used Boolean models are event or error trees. They are suitable for modelling non-repairable systems (such as safety or protection systems) and have a static set or faulty event sequences. Graph-based approaches can be considered by assuming that the generation of action sequences can be given, in a first approximation, by a path search in a graph representing the stabilized situations of the process to be operated. This type of sequence synthesis has been first applied in the chemical industry by [6] which proposed a method for automatically generating routes from the initial state to the expected state of a plant, taking into account safety rules, but without taking into account structural elements or physical values. In practice, the problem is more complex because the dynamics of the plants (operating mode, value of physical variables, etc.) must be taken into account:

- redundancies provide several operating scenarios, but their operational availability depends on the state of the process (for example, some physical values must have reached a given threshold in order to use some devices),
- the complexity of the architectures makes it difficult to take into account all possible scenarios [7]; the analysis can focus on some of them and requires initializing the analysis models with the accurate parameters.

The consideration of dynamic characteristics leads to a shift to state space models that can be used, depending on their deterministic or stochastic characteristics, to analyze sequences with two types of problem solving: determination and probability evaluation of event sequences [8]. Since the problem focuses on the deterministic generation of a sequence of actions, the probabilistic evaluation of architectures is outside the scope of this article and the required models will not address probabilities. In addition, this article assumes that a discrete and simplified representation of the evolution of physical values must be sufficient to solve this problem. Consequently, it is proposed that dynamic representations of the architecture be based on discrete event system models [9].

In the context of deterministic systems, the most commonly used state models are finite state automata (and language theory) and Petri nets [10]. Hybrid automata [11] also seem to be an interesting way to represent the structure and behaviour of the physical process. In the field of plant operation engineering, the graphical representation of operation can be done with different languages, such as OPNet (Operation Procedure NETwork) [12], Petri nets [13], UML based approaches [14], Grafchart models [15], the procedure-oriented graphical notation ProcGraph for the specification of the process control software [16] or the FRDL (Formal Recipe Description Language) [17]. On the basis of these Discrete Event Systems representations, the problem of sequence determination can be considered as a research and trajectory analysis in a state space characterizing the evolutions of a set of dynamic models. Several formal approaches can be considered, such as verification of safety properties, synthesis and analysis of reachability.

Formal verification is the provision of mathematical evidence that the operation of a given architecture meets certain safety and performance properties. The main approach is based on the verification of the model [18] which is an automatic technique of state spatial exploration in relation to properties expressed in a formal logic. It applies to prove the safety of the installation's operating procedures, which are *a priori* known [19, 20, 2], which is not the case for our problem.

The automatic control synthesis techniques [21, 22] define all possible paths in a controller model that meet a set of behavioral specifications. These techniques have been used to generate action sequences for plant operation using Petri nets [23] and state machines [24]. However, they require the modelling of a specification that presents certain choices in terms of plant operating behaviour. This is obviously not the best way to prove the existence of a safe sequence, whatever the specification.

Finally, reachability analysis clearly appears to be the appropriate means of solving the problem of qualitative evaluation of architectures and proving the existence of a legal and safe sequence from one state to another. Among the different reachability analysis techniques, model-checking can be used by considering the properties to be proven [25] as a reachability property. In case of existence, the model-checker returns a single sequence, represented by a trace, among all the possibilities. This technique has already proven its effectiveness, using the Timed Automata [26], in the field of operation [27, 28]. For hybrid automata, the reachability problem can be undecidable, which makes them unusable for an approach to analyse reachability. Although the extension to industrial installations is still an open issue, the existing literature shows the advantage of using reachability analysis on Timed Automata models to generate secure process operations.

## 4. An architecture assessment approach based on reachability analysis

### *4.1. Objectives and hypotheses*

This work focuses on the early validation of the architecture of critical complex systems in the context of plant operations. The objective is to provide designers and human operators with formal models to prove that the plant can be operated safely, particularly in critical situations, and that the architecture design provides capabilities to achieve safe states. The role of the operators is to identify critical plant operations and to introduce into the models the precise parameters that characterize the situation being analyzed.

According to related work, timed automata (TA) and model-checking reachability analysis appear to be an effective means of demonstrating the existence of at least one sequence of actions satisfying the safety and time constraints for reaching an expected process state from the current state. If the existence of at least one sequence is demonstrated, the process architecture will be considered as valid for the analyzed situation.

Two main hypotheses are retained to enrich the TA related works in the case of large industrial systems: the size and complexity of the models, and the practicability of the approach in an engineering process where several candidate architectures have to be evaluated.

The first hypothesis is that modularity and hierarchy are an effective answer to the problems of model size and complexity. It takes advantage of the hierarchical structure of the process architecture around different levels of functionality (actuation and measurement, control and monitoring, plant operation and management...) and operational devices (field devices such as valves and pumps, PLCs, SCADA systems).

Several research works (Multilevel Flow Modelling [29]) or international standards (ANSI/ISA-88.01-1995 standard [30]) provide guidelines for modelling complex industrial processes according to recommended hierarchy schemes. Applications for critical industrial processes, such as chemical and energy processes, are available [31] and will be our basis for producing modular and hierarchical TA architecture models.

The second assumption is that the reuse of models is essential to make our approach acceptable and applicable in an engineering process. Indeed, as many architectures have to be evaluated during the engineering process, a major issue is then the ability of modelers to easily build and modify the system model (for example by adding or removing component models). Pattern-based modeling, combined with a structured modeling framework, seems to be a very efficient way to allow a plug-and-play approach to compose models from a library of generic and reusable models: [32, 33, 34]. These models must have standard interfaces to ensure their interoperability and can be adapted and composed according to the architecture to be modelled.

### *4.2. Chosen modelling formalism : Timed Automata*

As mentioned previously, the chosen modeling formalism is based on timed automata defined by [26] as an extension of finite state automata. A timed automaton is a 9-uplet $\mathcal{A} = (S, V, X, L, I, T, S_m, s_0, v_0)$ where:

- $S$, $V$, $X$ and $I$ are respectively finite sets of locations, variables, clocks and invariants that label locations with some clock constraints,
- $L$ is a set of synchronization labels (or channels) divided into emitting labels $L_e$ (noted *label*!) and reception labels $L_r$ (noted *label*?),
- $T$ is a set of transitions $(s, l, g, m, s') \in S \times L \times G \times M \times S$ where $G$ is a set of guards (crossing constraints on variables of $V$ and clocks of $X$) and $M$ a set of updates on the valuation of variables and clocks [1]; $l$, $g$ and $m$ are optional,

[1] Notation convention: synchronization labels are followed by ! or ?, guards are between brackets [ ], and updates are under the following form: $v = value$

- $S_m \subseteq S$, $s_0 \in S$ and $v_0 \in V$ are respectively the set of marked locations, the initial location and the initial values variables of $V$.

The state of a TA is described in a triplet $state_{\mathcal{A}} = (s, v, x)$, where $s$ is the active location, $v$ is the set of variable values and $x$ is the set of clock values. It can evolve in two cases: after a transition crossing or after a lapse of time (evolution of the clocks) if the invariant is maintained.

An execution trace $\gamma = (s_1, x_1, v_1) \xrightarrow{t_1} (s_2, x_2, v_2)\ ...\ \xrightarrow{t_n} (s_n, x_n, v_n)$ is a sequence of size $n$ alternating states and transitions with $|\gamma| = n$.

A network of $m$ timed automata, $\mathcal{NA} = \mathcal{A}^1 \| \mathcal{A}^2 \| ... \| \mathcal{A}^m$ consists in a set of timed automata which can evolve in parallel (symbol $\|$). A network provides structuring mechanisms to represent the synchronisation of timed automata through channels (emission and reception labels) and shared variables or clocks. The overall state of the network is a triplet $state_{\mathcal{NA}} = (rs, rv, rx) \in STATE_{NA}$ where $rs$ is the set of active locations, $rv$ is the set of variables values and $rx$ is the set of clocks values. An automata network $\mathcal{NA}$ evolves with:

- a local evolution in an automaton of the network (if not related to a synchronisation label),
- the simultaneous crossing of both transitions $t_p^\alpha$, $t_q^\beta$ of an automata couple $(A^\alpha, A^\beta)$ with $t_p^\alpha$ and $t_q^\beta$ respectively labelled by $l_p^\alpha!$ and $l_q^\beta?$ such as $l_p^\alpha = l_q^\beta$, source locations of $t_p^\alpha$ and $t_q^\beta$ are active and their guards satisfied.

An execution trace on a timed automata network $\Gamma = S_1 \xrightarrow{T_1} S_2\ ...\ \xrightarrow{T_N} S_N$ is an alternating sequence of states and transitions where $S_i$ is a state of the automata network $\in STATE_{NA}$, and $T_i$ is either a transition of the automata network, or a couple of transitions $(t_p^\alpha, t_q^\beta)$ crossed simultaneously in the case of a synchronization channel.

### *4.3. Principles of the use of reachability analysis for architecture validation*

This subsection presents the general framework of the proposed approach (Figure 1). The proposal is based on the definition of modelling patterns (Process and Sequencer Models) which are presented in the section 5, on the analysis of the reachability thanks to model-checking then on the analysis of the resulting trace in terms of action sequences to validate or invalidate the architecture.

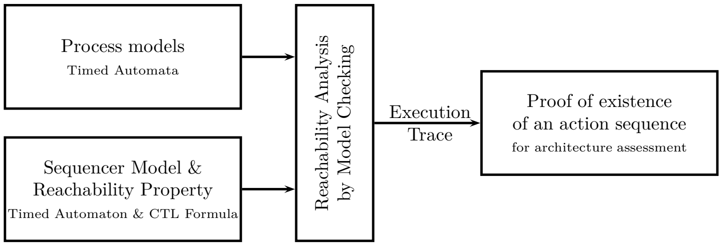


Figure 1: Framework for architecture validation

### *4.3.1. Reachability analysis by model-checking*

The reachability analysis is designed to return a trace of execution from the initial state of the sequencer to the state mentioned in the CTL property. As seen in the section 3, formal model-checking techniques [35] allow, thanks to automatic state space traversal mechanisms, to prove (or disprove) that a model satisfies (or does not satisfy) a formal property, which can be a reachability property.

If a path to the target state of the sequencer model exists, then the property will be satisfied and a model-checking tool will be able to provide an execution trace leading to this target state. A trace is then an alternating sequence of states and transitions $\Gamma = S_0 \overset{T_1}{\rightarrow} S_1 \ ... \overset{T_n}{\rightarrow} S_n$ on the automata network where $S_i$ represents a state and $T_j$ a transition.

It is important to note that the generated trace is often the first path leading to the property check that is encountered when exploring the state space. It is therefore totally dependent on how the model-checker conducts its exploration (in width or depth, for example). The operational relevance of this sequence may be questioned, but the objective of the approach proposed here is only to show that it is possible to reach a target situation, and not to try to make a system work in an optimal way. However, some model-checkers offer the possibility of providing an optimal trace (in terms of length or duration of the sequences) but with a major drawback linked to the problem of combinatorial explosion since it requires exploring the entire state space.

### *4.3.2. Architecture validation*

If such a trace can be provided, it means that there is a sequence of actions that meets the operational requirements of the facility for the analyzed context expressed through the sequencer. Otherwise, a failure of the proof means that the objective is not achievable and will probably require an evolution of the designed architecture.

From the traces that are generated, it is then possible to interpret the process operations which are sequences of steps specifying the actions and observations to be made on the process to go from a given state (considered as initial) to a desired state (considered as final). The trace given by the model-checker shall be processed to present the sequence of actions as understandable in the context of the operation of the plant. For the purpose of this study, only the actions to be applied to the process devices shall be considered. This leads to the application of a projection function on the generated run-time trace to filter the observations, i.e. the transitions identified by a report, in order to keep, for each transition drawn, only the source and reached states (locations, variables and clocks) and the requested actions.

## *4.4. Generation of action sequences*

As presented in Figure 1, the reachability of a target location in the sequencer model requiring the process models is a safety indicator to validate the architecture against the scenario described in the sequencer model under boundary conditions (especially in case of unavailability of some devices). Proof of reachability is given in the form of an execution trace from which a sequence of actions for plant operation can be deduced.

### *4.4.1. Reachability properties and generation of execution traces*

The reachability property to be proved is modelled using a CTL (Computation Tree Logic) formula. The CTL language uses logical and temporal operators to formalize different types of properties: among them, the quantifiers `A` means 'along the whole path' (inevitably) and `E` means 'along at least one existing path' (possibly). These quantifiers can be complemented by path specific quantifiers such as `G` for 'Overall' and `F` for 'Finally'. Therefore, the reachability property of a sequencer will be represented as a combination of `E` and `F` as follows : `EF sequencer.`$s_{expected\_state}$.

Verification of the reachability property $EF\ Sequencer.EndOfSequence$ on models $S_N, M_N, M_{N-1}, \ldots, M_1$ can be formalized as:

$$\Gamma \leftarrow (M_N \parallel M_{N-1} \parallel ... \parallel M_1 \parallel S_N) \models EF\ S_i.EndOfSequence$$

where $\Gamma$ is the trace of execution sought and $\models$ is the relationship of satisfaction. The obtained execution trace contains a sequence of states (active locations, clocks and variables) and transitions leading to the target location of the sequencer. An example of such a trace is given in Figure 2.

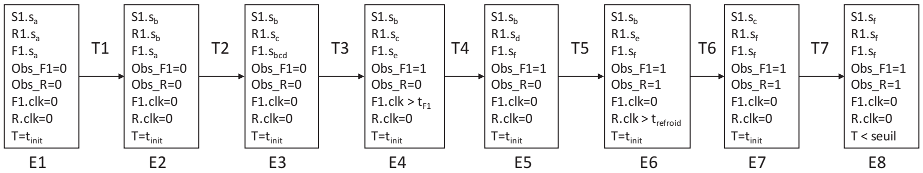


Figure 2: Example of execution trace

### *4.4.2. Generation of an admissible action sequence*

While a sequence of execution may be sufficient to prove the existence of a safe operating path, its interpretation and understanding by a plant operations expert is not without significance. Indeed, it often contains many transitions that make sense only for the internal evolution mechanisms of the models.

From the plant operation point of view, a sequence of actions consists of the different operations that must be performed sequentially on the process elements. Depending on the context, these may be field devices (process models $M_1$) such as actuation and measurement systems (opening/closing a valve, starting/stopping a pump, a motor, configuring a transmitter, ...) or computer devices ($M_i$ with process models $i \geq 1$) to control or operate the plant (starting/stopping a function, an operating mode, a subsystem, ...). Each step of an action sequence identifies the action to be executed and defines the initial and reached situations.

Formally, an action is defined as a 4-uplet $(label, v_d, v_f, n)$, where $label$ contains a $Req_i^p$ request to be applied on the $p$ element at the $i$ level to change its state, $v_d$ and $v_f$ contains the values of the observation variables $(Obs\_M_i^p)$ and the clocks respectively before and after the action, and $n$ indicates the hierarchical level of the actuated element.

The algorithm 1 allows to generate an action sequence $\Delta$ associated with $\Gamma$. In order to respect the semantics of an action sequence, the basic principle used in the logic of the algorithm is close to a projection function, since it consists in extracting from the $\Gamma$ trace all the transitions that do not involve actions to be performed on one of the elements of the process, and in constructing timestamps, values of the variables and states before and after actions.

**Algorithm 1** Generation of action sequence $\Delta$ for $S_N \| M_N \| M_{N-1} \| ... \| M_1$

```
Require:
    $\Gamma = E_1 \xrightarrow{T_1} E_2 ... \xrightarrow{T_N} E_X$        % Execution trace %
Ensure:
    $\Delta = \delta_1 \delta_2 ... \delta_X$ with $\delta_s = (label \in L, v_d \in V, v_f \in V, n \in \mathbb{N}^*)$

    % $Ch(T_k)$ returns the channel (report or request) that labels the transition $T_k$ if it exists or null otherwise.
    % $V(E_i)$ returns the value of variables of the state $E_i$ in the execution trace and the value of the global clock.
    % $\Sigma_r^{M_{i-1}^p}$ is a subset of the alphabet associated to the automata $M_{i-1}^p$ that only contains request channels.

    $k \leftarrow 1$ ; $s \leftarrow 1$
    for k=1 à X do
        if $T_k = (t_\alpha, t_\beta) \wedge t_\alpha \in M_i \wedge t_\beta \in M_{i-1}^p \wedge Ch(T_k) \in \Sigma_r^{M_{i-1}^p}$ then
            $label \leftarrow Ch(T_k) = Req_{i-1}^p$ ; $v_d \leftarrow v(E_k)$ ; $v_f \leftarrow V(E_{k+1})$; $n \leftarrow i - 1$
            $\delta_s \leftarrow (label, v_d, v_f, n)$ ; $s \leftarrow s + 1$
        end if
        if $(T_k = t_\alpha) \wedge (t_\alpha \in S_N)$ then
            $label \leftarrow \varepsilon$ ; $v_d \leftarrow v(E_k)$ ; $v_f \leftarrow V(E_{k+1})$; $n \leftarrow N$
            $\delta_s \leftarrow (label, v_d, v_f, n)$ ; $s \leftarrow s + 1$
        end if
        $k \leftarrow k + 1$
    end for
    return $\Delta = \delta_1 \delta_2 ... \delta_{s-1}$
```

## 5. Pattern-based hierarchical modelling

The previous section introduced the principles of using Reachability Analysis for architecture validation. This approach is based on a modelling framework for process and sequencer models that is presented in this section. This framework uses hierarchical and standardized synchronization mechanisms. An originality is the definition of generic models to systematize the modelling of process and sequencers and to promote the reuse of models. The standardized variables and elementary generic structures of the TA patterns will be stored in a library from which the process and sequencer models will be built using a plug-and-play approach. The expected benefits are the reduction of modelling activity time since model building is done through specialization and model instantiation.

## *5.1. Models structure*

### *5.1.1. Process models*

Process architecture modelling promotes hierarchy and modularity. For these reasons, we consider that a process architecture is structured in $N$ hierarchical levels (left part of Figure 3). These levels can be identified using ANSI.ISA-88 or internal company practices. The set of higher level models corresponds to the plant operating recipe and will be denoted as $M_N$ while the set of lower level models relates to field devices and will be denoted as $M_1$. Each level $i$ involves a set $M_i$ of models denoted as $M_i^k$ describing a component $k$ belonging to that level. The details of these models will be given later in this section.

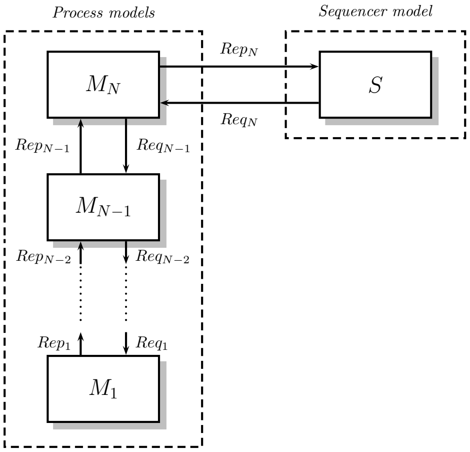


Figure 3: Structured process models $M_i$ with a sequencer model $S$

The hierarchical communication between components is based on the classic semantics of requests and reports. The evolution of reports on a $k$ component of level $i$ may require the use of one or more resources belonging to level $i-1$, which means that these resources must also be the scene of state changes. This is represented by a channel $Req_{i-1}^p!$ where a model $M_i^p \in M_i$ sends a query to a model $M_{i-1}^k \in M_{i-1}$. In Figure 3, all queries sent by the level $N-i$ are grouped under the notation $Req_{N-i-1}$. The orientation of the arcs represents a synchronization of the sent label (!) with the received label (?). By construction, lower-level templates $M_1^p$ cannot request any resources.

In response to a state change request from a level $i+1$, the requested model $M_i^p$ will report a message when its state change has occurred. This message is modeled by a synchronization channel $Rep_i^p$. In Figure 3, all reports sent by the level $N-i$ are grouped under the notation $Rep_{N-i-1}$.

In addition to this query/report mechanism, the evolutions of all models can be conditioned by shared global variables representing physical variables or clocks.

### *5.1.2. Sequencer model*

A sequencer is used to describe the plant operating scenario to be evaluated. In principle, it represents the initial process situation and one or more target situations. The transitions between these situations are conditioned by various constraints concerning physical variables, the state of some field devices, the state of functions or the operational state of the plant. Operators and engineering know-how help characterize these constraints which are, by definition, specific to each scenario.

The sequencer interacts with the highest level of the process model $M_N$ via the same request/report semantics using the $Req_N$ and $Rep_N$ synchronization channels (right part of Figure 3).

### *5.1.3. Modelling hypotheses*

At this step, some very important assumptions need to be highlighted. The first concerns the synchronisation of the models. The request sent by a component does not contain any explicit action that the called resources must execute. This only means that the state of these resources does not meet the requirements of the state the requester wants to reach and, therefore, a change in the state of the resources is mandatory. The appropriate evolution taken by the resource to meet the conditions set by the requester will be randomly selected during the reachability analysis process until a solution is found. Similarly, if several resources are able to provide the services requested by a component, the model for that component will contain several transitions, each of which will be labelled by a request to a resource, without identifying a choice between the resources requested. Again, this role will be assigned to reachability analysis.

The second hypothesis concerns the modelling of the evolution of physical quantities. In the models, it is represented by loop transitions on stable states. The crossing of these transitions is only conditioned by a clock invariant and triggers a modification of the value of the physical quantity concerned. For the purposes of this article, since the systems concerned are mainly composed of valves, pipes and pumps with constant and controlled flow rates, the modelling of the evolution of the physical parameters depends on conditions that can be modeled by first-order relationships. This discretization is therefore represented by fixed increments or decrements, the value of which depends on the clock value chosen to cross the transition. This may depend, for example, on the flow rate of a pump. The incrementation and decrementation of a physical quantity is included in only one model, which ensures that there will be no simultaneous changes or multiple updates.

The third hypothesis concerns the synchronization of the models. Figure 3 shows the exchanges between the process models at different levels, as well as with the sequencer model. These exchanges of requests and reports are updates, and not communications between distributed components of the considered system. For this reason, there is no latency when synchronizing the models.

### *5.2. Variable patterns*

Standardised synchronisation mechanisms, based on query/report semantics ($Req_i$ or $Rep_i$), do not contain all the necessary information on the status of the process and sequencer models. This information is nevertheless mandatory because the conditions of evolution within a model may depend on it (such as functional or availability constraints, exclusion guards...). This information is modelled by shared variables that can be written or read in the process and sequencer models :

- the observation of the active locations of the model $M_i^k$ is recorded in the variable noted $Obs\_M_i^k$; these variables can be boolean, in the case of a model having only two locations of interest, or integer otherwise; these variables are updated when the locations of interest are reached;
- physical values $\Phi$; these variables are integer and are updated by the models that act on them;
- the availability of a device described by the $M_i^k$ model is recorded in the $Avail\_M_i^k$ boolean variable. These variables are used to characterize the operating scenario analyzed in the presence of certain unavailable devices and/or functions. They are set at model initialization, before the reachability analysis, to evaluate the reachability of a target situation given the known unavailability of some devices.

### *5.3. Location pattern*

The modelling pattern for process and sequencer models is based on the concept of stable locations and on a set of transitions between two stable locations. A stable location in a model $M_i^k$ is a location whose deactivation requires the receipt of a state change request. A stable location is a location such as :

- all output transitions must be at least labelled by a synchronization label $Req_i^k\ ? \in L_r$,
- by symetry, all input transitions have to be labelled by synchronization label $Rep_i^k\ ! \in L_e$.

In addition, we consider that a change in physical values can only be made when a stable location is active. These modifications are modelled using a self-loop transition on the stable location that updates the shared integer variables $\Phi$. This discrete modelling of the physical variables was considered sufficiently efficient by plant operations experts in our studies. The time step that evaluates the periodic evolution of the physical variables is given by the *step* parameter that is used as a guard for self-loop transitions.

### *5.4. Transition patterns*

Sets of transitional locations between stable locations ensure the preparation and closure of operations executed in stable locations. Generic semantics are given by the following sequence: a component receives a request from a higher level to change its internal stable location, then it calls one or more resources from lower levels and finally sends a status to the higher level when the new stable location is reached. Therefore, transitions from one stable location of $M_i^k$ to another can be labelled by three types of channels :

- a request $Req_i^k?$ received from higher level that justifies leaving the current stable location;
- several requests $Req_{i-1}^p!$ that request resources for state changes; all channels that request resources are grouped together in a set noted $\{Req_{i-1}^p\}$;
- a report $Req_i^k!$ sent to a higher level when a new stable location is reached.

The authorization of a transition sequence between two stable sites may depend on conditions related to the availability of the requested resources, the configuration of the requested resources and security constraints. These conditions are formalized by generic guards.

The availability constraint is formalized by a boolean guard $G_{avail}^{(s_a,s_b)}$, defined for a pair of stable locations $(s_a, s_b)$ by:

$$G_{avail}^{(s_a,s_b)} = [f_{avail}^{(s_a,s_b)}(Avail\_M_{i-1}^p) == TRUE].$$

This constraint is computed by function $f_{avail}^{(s_a,s_b)}$, which is specific to each process model, from shared variables $Avail\_M_{i-1}^p$ indicating the availability of the resources called. If and only if the resources needed for the configuration are all available, then the function returns the value *true*.

The configuration constraint represents the active locations of the called resources and/or the values of certain physical variables that must be satisfied to allow the new stable location of the requester. Configuration conformance for a stable location $s_a$ is formalized by a boolean guard $G_{conf}^{s_a}$ defined by :

$$G_{conf}^{s_a} = [f_{conf}^{s_a}(Obs\_M_{i-1}^p, \Phi) == TRUE].$$

This constraint is computed by the function $f_{conf}^{s_a}$, which is specific to each process model, from the shared variables $Obs\_M_{i-1}^p$, and $\Phi$.

Finally, security constraints represent some properties that must be satisfied before a requester is allowed to call a given resource. These properties can be linked to priority rules (for example, a pump's engagement can only be required if upstream valves are open) or exclusion constraints (for example, incompatibility between the execution of two given functions). In other words, when a resource $M_{i-1}^p!$ is requested by a channel $Req_{i-1}^p!$, the safety constraint requires that the active locations of some same-level elements and/or the values of certain physical variables be respected. It is formalized as a boolean guard $G_{safe}^p$ defined as :

$$G^{p}_{safe} = [f^{p}_{safe}(Obs\_M^{l}_{i-1}, \mathit{\Phi}) == TRUE].$$

where $l \neq p$ (safety constraint on the requested element $p$ relates to the other elements $l$). This constraint is computed by function $f^{p}_{safe}$, which is specific to each process model, from the shared variables $Obs\_M^{p}_{i-1}$, and $\mathit{\Phi}$. If and only if the safety constraints are satisfied, then the function returns the value *true*.

### *5.5. "Two stable locations" pattern for process models*

The generic structure between two stable locations of a model $M^{k}_{i}$ for $i \neq 1$ is given in Figure 4a.

The transition from the stable location $s_a$ to the stable location $s_f$ is crossed upon receipt of a request $Req^{k}_{i}?$ from a model $M^{q}_{i+1}$ (this request is issued between the locations $s_b$ and $s_c$ of $M^{q}_{i+1}$). This request can only be accepted if the availability guard $G_d$ is true.

Assuming the request is enabled, the model starts calling lower-level resources until the conditions for reaching the stable location $s_f$ are met. The loop between $s_b$ and $s_c$ represents the successive requests $Req^{p}_{i-1}$ sent to the resources (these requests are received by $M^{q}_{i-1}$ between $s_a$ and $s_b$) until the requirements $G^{s_f}_{f}$ for the stable location $s_f$ are met. The $s_b \rightarrow s_c \rightarrow s_b$ trace means that a resource was requested and responded positively but did not satisfy the configuration guard $G^{s_f}_{f}$. This sequence is executed until $G^{s_f}_{f}$ is true and reaches the $s_d$ location.

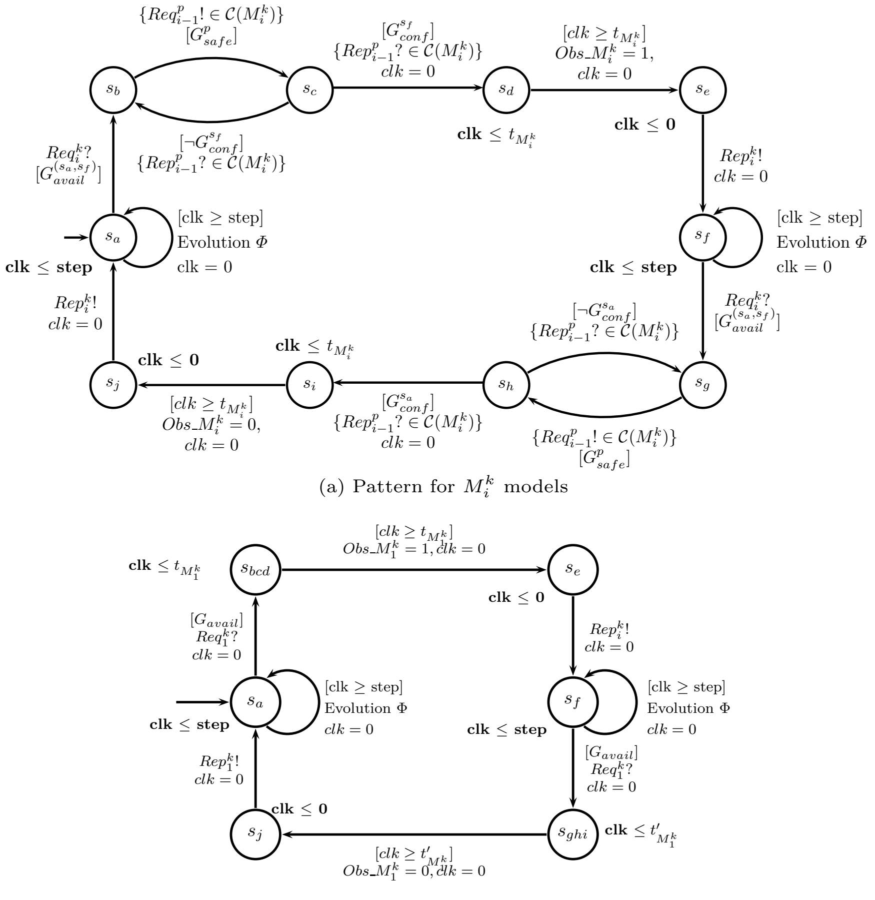
The transition from $s_d$ to $s_e$ is intended to update the observation variable $Obs\_M^{k}_{i}$ after time $t_{M^{k}_{i}}$ has elapsed. This time, which can be null, represents the transfer time from one stable location to another. Finally, the transition from $s_e$ to $s_f$ is only used to send a status change report $Rep^{k}_{i}!$ to $M^{q}_{i+1}$ (this report is received between locations $s_c$ and $s_b$ or $s_c$ and $s_d$).

The same process is applied for modelling the transient sequence between the stable location $s_f$ and the stable location $s_a$.

In order to reduce the number of request transitions in a model $M^{k}_{i}$ (potentially equal to the number of level $i-1$ elements), it is possible to restrict them only to the level $i-1$ resources used by the $M^{k}_{i}$ element. This is formalized by [35] through a set called cone of influence. In our case, this set is a set of requests $Req^{p}_{i-1}!$ which is computed by listing the observations $Obs\_M^{p}_{i-1}$ and the physical values $\mathit{\Phi}(M^{p}_{i-1})$ involved in the configuration guards of $M^{k}_{i}$. This cone of influence is defined as $\mathcal{C}(M^{k}_{i}) = \{Req^{p}_{i-1}\}$ for all $p$ where either $Obs\_M^{p}_{i-1}$ or $\mathit{\Phi}(M^{p}_{i-1})$ belongs to the guard $G^{s_x}_{c}$ for all stable locations $s_x$ of the model $M^{k}_{i}$.

Finally, let us recall that the set $\{Req^{p}_{i-1} \in \mathcal{C}(M^{k}_{i})\}$ which labels some transitions (for example from $s_b$ to $s_c$) is a notation convention which actually represents a set of transitions, each of them being labelled by an element of the set $\{Req^{p}_{i-1}\}$.

For lower level process models $M_1$ (valves, pumps, sensors, ...), the pattern is given by Figure 4b. The transient sequence from one stable location to another has no resources to trigger. Consequently, no functional constraints are taken



(a) Pattern for $M^k_i$ models

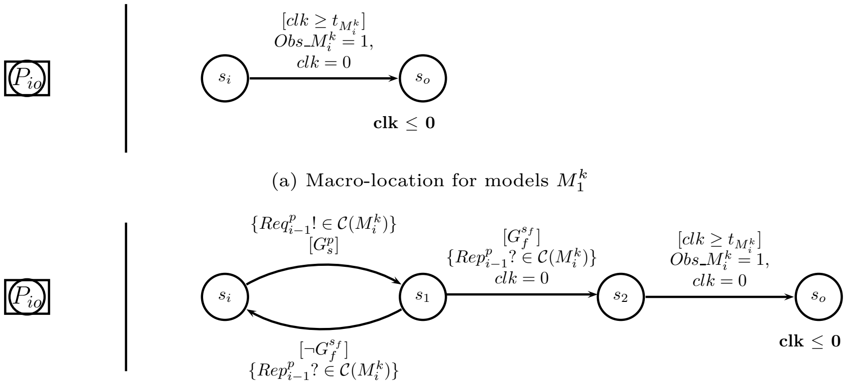
(b) Pattern for $M^k_1$ models

Figure 4: Two stable locations pattern

into account, and safety constraints are ensured by the higher levels. The two stable location model of Figure 4a is then simplified by substituting the locations and transitions from $s_b$ to $s_d$ by a single location (similarly for $s_g$ to $s_i$). The only guard considered is the availability constraint but its semantics needs to be adapted. Indeed, talking about availability of lower level resources makes no sense but availability must be considered for the device itself since it must be available to be triggered. The guard $[G^{(s_a,s_f)}_d]$ is then substituted by a guard $G_d = Avail\_M^k_i$, where $Avail\_M^k_i$ is a boolean variable representing the availability of the element $M^k_i$.

### *5.6. Using patterns to structure process models $M_n$*

The generic pattern proposed in the previous section allows to systematically obtain the evolution model between two stable locations of a process element. If this process element has only two stable states, which is often the case (on/off valve, pump, two-mode functions such as off and on, ...), the model of the process element is similar to the generic model. It will suffice to define the functions $f^{(s_a,s_f)}_{avail}$, $f^{s_a}_{conf}$, $f^{s_f}_{conf}$ and $f^p_{safe}$ which are specific to each model.



(a) Macro-location for models $M_1^k$

(b) Macro-location for models $M_i^k$

Figure 5: Macro-locations

Otherwise, in the case of a number of stable locations greater than two (three-way valves, functions with several modes such as off, on, gradient, ...), the process model is obtained by applying the ”two stable locations” for all pairs of stable locations.

To simplify the representation of the models, a graphical notation of macro localization, inspired by the Grafcet macro step of the IEC 60848 standard [36, 37], is proposed. A macro-location $P_{io}$, whose notation is given in the left part of the Figure 5 is characterized by an input location $s_i$, an output location $s_o$ and inner locations $s_{int}$. The transition from a $s_x$ location to a macro-location is a graphical representation of the transition from that $s_x$ location to the $s_i$ input location of the macro location; the transition from a macro-location to a $s_y$ location is a graphical representation of the transition from the $s_o$ output location to that $s_y$ location.

The right part of Figure 5a shows the expansion of a macro-location representing the sequence from $s_b$ to $s_e$ of the template pattern given in Figure 5a while the right part of Figure 5b shows the expansion for $M_i^k$ templates. As this notion is only a graphical notation, the syntax and semantics of macro-locations remain classical.

A process model using macro-locations must satisfy the following two properties :

- there is at most one $P_{ab}$ macro location from the stable $s_a$ to the stable $s_b$ location regardless of $a$ and $b$,
- a stable $s_a$ location must have at least one output transition to a $P_{ab}$ macro transition with $a \neq b$.

The building of a model with $k$ stable locations depends on the transitions between the different stable locations. If a change between two stable locations, for example from $s_a$ to $s_b$, is to be modeled, then a $P_{ab}$ macro-location is placed between these two stable locations, and two transitions are defined from $s_a$ to the input location of $P_{ab}$ and from the output location of $P_{ab}$ to $s_b$. Figure 6 gives an example where 3 stable locations can be reached using 5 paths.

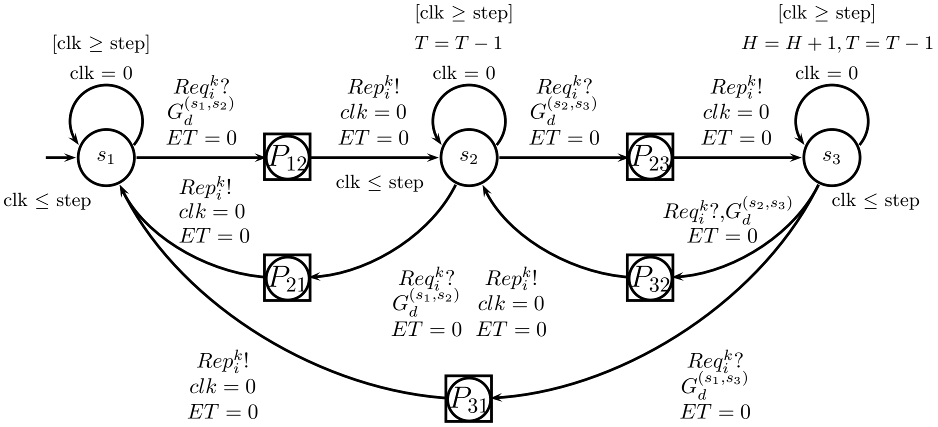


Figure 6: Example of model using macro-locations

### 5.7. Patterns for sequencer models S

As presented in Figure 3, the sequencer model is synchronized with the models $M_N$ using two channels: transmission of a set of requests $\{Req_N^k! \in Req_N\}$ and receiving a set of reports $\{Rep_N^k? \in Rep_N\}$.

The sequencer has an initial location, noted $s_a$, and an expected location, noted $s_c$. The change from $s_a$ to $s_c$ corresponds to the operating scenario being evaluated. Recalling that the objective is to find an execution trace on the process models $M_i$, the sequencer will use the request channels to trigger the process models as resources helping to reach its expected location. Accordingly, the sequencer iteratively sends requests to the $M_N^k$ models if the security conditions $(f_{safe}^k(Obs\_M_N^k, \Phi))$ are met, until the conditions in terms of active locations of the $M_N^k$ models and physical values $f_{conf}^{s_c}(Obs\_M_N^k, \Phi)$ are met. Iteration is supported by the first transition from $s_a$ to $s_b$ and by self-loop transitions on the location $s_b$. An example of a sequencer automaton is given by the Figure 7, where the location $s_c$ represents the desired target situation.

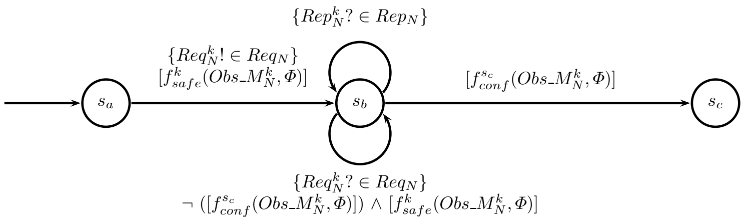

$\{Rep_N^k? \in Rep_N\}$
$\{Req_N^k! \in Req_N\}$
$[f_{safe}^k(Obs\_M_N^k, \Phi)]$
$[f_{conf}^{s_c}(Obs\_M_N^k, \Phi)]$
$s_a$
$s_b$
$s_c$
$\{Req_N^k? \in Req_N\}$
$\neg\ ([f_{conf}^{s_c}(Obs\_M_N^k, \Phi)]) \wedge [f_{safe}^k(Obs\_M_N^k, \Phi)]$

Figure 7: Sequencer pattern

This generic pattern can be used in combination, as with process patterns, to create compound sequencers. It can be useful to describe a sequence of phases in which the physical values must reach different sets of values: for example, a preparation phase, then an equilibrium state, and finally a closing phase.

## 6. Application using an Industrial case study

### *6.1. Presentation of the case study*

The CISPI platform is part of a larger demonstrator developed within the CONNEXION project[2]. It is dedicated to research work on industrial process management, which involves a set of complex processes covering different functional modes (production, shutdown, start-up, ...) and critical modes (normal, incident and accidental operation).

The CISPI platform provides auxiliary water supply to other systems and is structured in three hierarchical levels with a large number of hardware and functional redundancies: 22 field equipments ($M_1$) can be configured to perform 24 process functions ($M_2$) to reach different operational situations of a recipe ($M_3$). A partial flowchart of the platform is given in Figure 8.

The objective is to validate the process architecture by demonstrating that a target situation can always be reached safely even if some resources are not available or if changes have been made to the process and its control. The initial situation is as follows: the valves are closed, the pumps do not work, tank 001BA is full of water and tank 002BA is empty. The target situation is to supply tank 002BA with water up to a volume of 10 volume units. Three scenarios are proposed, which differ in terms of equipment availability :

- Scenario 1: all devices are available;
- Scenario 2: all devices are available except valves $VM3$ and $VE3$ and pump $PO1$;
- Scenario 3: all devices are available except valves $VE1$ and $VE3$.

[2] CONNEXION (French Excellence in Nuclear Control Systems) is a research project on the safety of control systems in the nuclear industry. It was supported by the French government.

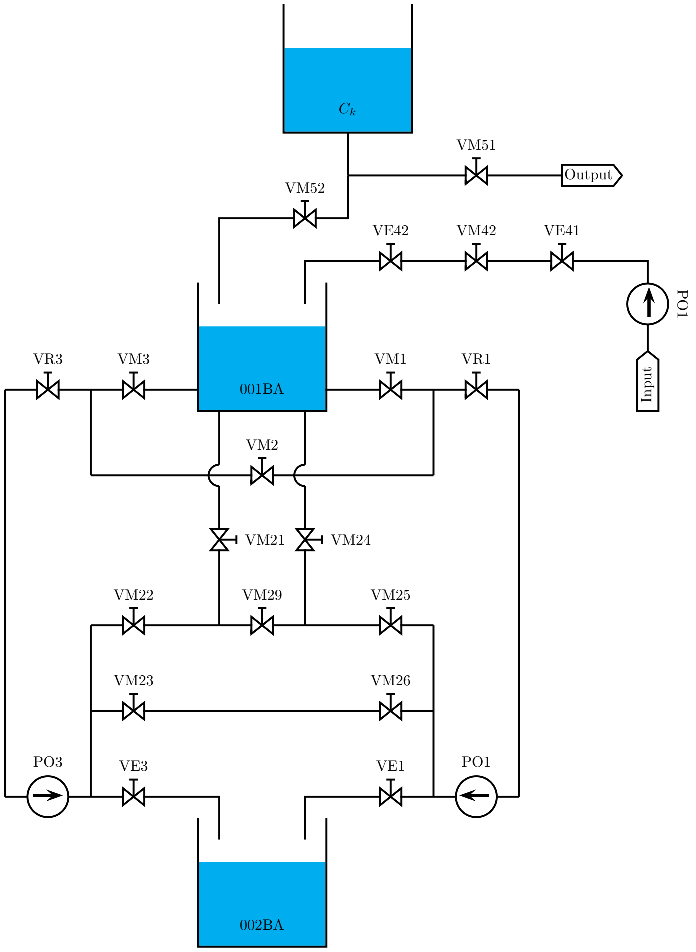


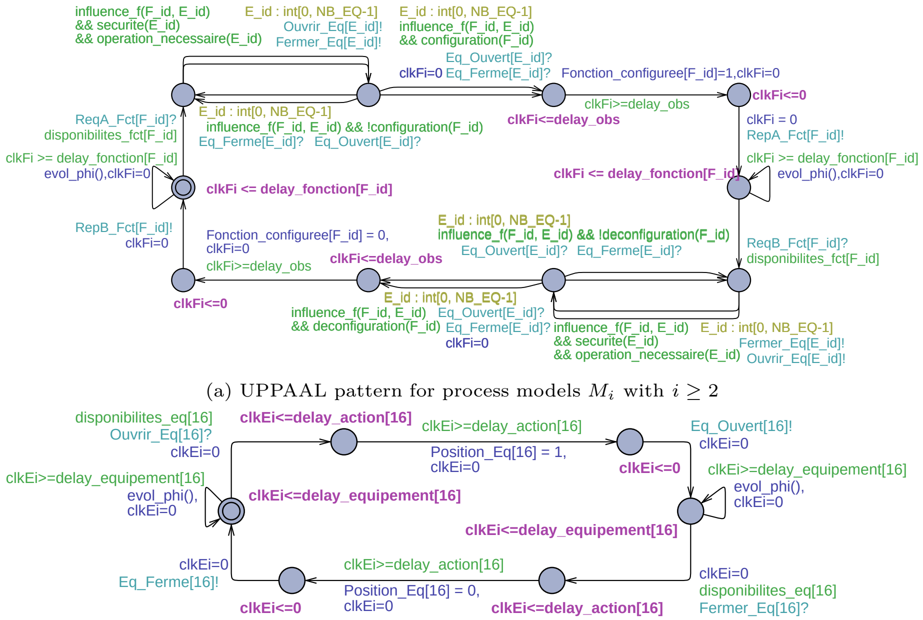
Figure 8: Process flow diagram of the CISPI platform

### *6.2. Case study modelling*

The implementation of the models of recipes, functions, devices and sequencers with timed automata, as well as the reachability analyses, were performed with the UppAal model checking tool [38]. The models presented in the section 5 have been implemented into the UppAal tool, which conforms to TA semantics, to model equipment, functions and process recipe. Although the language proposed by Uppaal is more expressive than the traditional TA definition and may allow the creation of more "readable" models, we chose to use TA to facilitate model and approach portability.

As discussed in the previous section, the main advantage of using TAs is that it reduces the time required to build the model. The UppAal tool has component-oriented features that allow the creation of parametric ”templates”, which can be used as generic models and then instantiated. Therefore, defining process models for an architecture is similar to a plug-and-play process that selects process models from the library. Their parameterization is done according to parameter tables (for variables, constants and clock thresholds) and functions (for the definition of guards) which are interpreted by C♯ scripts to set up the UppAal automata.

Figure 9a provides the UppAal implementation of the process model (at the level $\geq 2$) and Figure 9b gives an example of the instantiated valve model $VM21$. Note that a double circled UppAal location is an initial location.



(a) UPPAAL pattern for process models $M_i$ with $i \geq 2$

(b) UPPAAL model of valve $VM21$

Figure 9: Implementation of process models with UPPAAL

### 6.3. *Results and analysis*

For Scenario 1, a proof of success has been returned, meaning that a safe sequence of action exists to reach the target state of the sequencer. An execution trace has been proposed and has been interpreted according to Algorithm 1 as:

1. `Configure function CpCsExt1G`
   (a) `Open valve VR3`
   (b) `Open valve VM3`
   (c) `Start pump PO3`
   (d) `Open valve VE3`

2. `Wait 10 time units`

3. `Objective situation is reached`

For Scenario 2, for which $VM3$, $VE3$ and $PO1$ are not available, it was also possible to achieve the target situation by opening the valves $VM1$, $VR3$, $VM23$, $VM26$ and $VE1$, starting the pump $PO1$ and waiting 10 units of time. This means that the considered architecture is valid with respect to the operational objective, in the considered architecture configurations.

On the opposite, a proof of failure has been returned for scenario 3 where neither $VE1$ nor $VE3$ are available. This means that if the operational situation being assessed occurs, the current CISPI architecture will not be able to respond in a safe manner.

Even if an architecture is valid, this does not mean that the generated sequences are operationally valid. Indeed, a generated sequence can use the same device several times, on opposite actions which are consequently useless actions, or be very long compared to others. One way to validate generated sequences is to obtain the operator's point of view. If several sequences are generated for the same case, some may be considered better than others, and the operator will then have to make a choice between them. This choice can be made according to different criteria such as, for example, the duration of the sequence, the number of actions in the sequence, or the geographical distance between devices.

The simulation of the generated traces or sequences can be useful for such activities. A prototype has been developed in this way in the CONNECTION project. The process behavior has been simulated with Alices [3] while the control and operation of the plant has been implemented with Scade[4]. The generated sequences were run by an operator on this coupled simulation to visualize and evaluate the relevance of the generated sequence.

## 7. Conclusion and open issues

The objective of the work presented in this paper is to propose an approach to validate as early as possible the operability of the architecture of safety-critical systems. It considers operational validation from the operator's point of view. The approach is based on reachability analyses of timed automata representing the process behaviour, which are initiated by the operators. If the situations characterized by the operators are reachable, the architectures considered are valid; otherwise, the architectures are not and must be modified.

The proposal of a formal modeling framework, based on patterns, favors their reuse for many architecture validations. In this way, the modelling effort, and thus the time and cost of modelling, is reduced.

[3] Alices is a tool for the full-scale simulation of a power plant developed by CORYS.

[4] Scade is a formal simulation and verification tool for embedded control systems developed by ANSYS.

Work in progress is the definition of a library of models and patterns that could be used to facilitate the composition of process models and sequencers. The open issues in this definition are the classification criteria and the level of detail that the model author must address. Indeed, depending on the expertise of the user, different levels of detail may be required: for example, a low level may be required in the case of an operator who wants to validate an architecture in a particular situation, or a high level in the case of a modeler who proposes a new model.

Beyond the reachability analysis, which enables an architecture to be validated, the use of model-checking on models developed according to the proposed approach enables the models to be validated, in particular from the point of view of operating safety. In other words, from a physical point of view, it is a question, for example, of checking that physical quantities do not exceed certain thresholds (is the volume of the tank greater than a safety level?). From an operational point of view, it may be necessary to check that a configuration has been carried out before starting an action (is the valve open before starting the pump?).

Another major problem in using reachability analysis is the risk of combinatorial explosion that often occurs when verifying large-scale models, as shown by previous preliminary work on the feasibility of generating action sequences [39]. The fact that the systems covered are very large justifies the use of different levels of abstraction to progressively construct action sequences through iterative refinement.

## 8. References

[1] L. Mårtensson, Are operators and pilots in control of complex systems?, Control Engineering Practice 7 (2) (1999) 173 – 182.

[2] J. Lahtinen, J. Valkonen, K. Björkman, J. Frits, I. Niemelä, K. Heljanko, Model checking of safety-critical software in the nuclear engineering domain, Reliability Engineering & System Safety 105 (2012) 104–113.

[3] C. Devic, P. Morilhat, Connexion contrôle commande nucléaire numérique pour l'export et la rénovation - coupler génie logiciel et ingénierie système : source d'innovations, Génie Logiciel 104 (2013) 2–11.

[4] O. Goubali, P. Girard, L. Guittet, A. Bignon, D. Kesraoui, P. Berruet, J.-F. Bouillon, Designing functional specifications for complex systems, in: International Conference on Human-Computer Interaction, Springer, 2016, pp. 166–177.

[5] D. Galara, Roadmap to master the complexity of process operation to help operators improve safety, productivity and reduce environmental impact, Annual Reviews in Control 30 (2) (2006) 215–222.

[6] N. Foulkes, M. Walton, P. Andow, M. Galluzzo, Computer-aided synthesis of complex pump and valve operations, Computers & Chemical Engineering 12 (9) (1988) 1035–1044.

[7] J. Rushby, Using model checking to help discover mode confusions and other automation surprises, Reliability Engineering & System Safety 75 (2) (2002) 167–177.

[8] M. Bouissou, J. Bon, A new formalism that combines advantages of fault-trees and markov models: Boolean logic driven markov processes, Reliability Engineering and System Safety 82 (2) (2003) 149–163.

[9] C. G. Cassandras, S. Lafortune, Introduction to discrete event systems, Springer, 2008.

[10] C. A. Petri, Communication with automata (1966).

[11] T. A. Henzinger, The theory of hybrid automata, in: Verification of Digital and Hybrid Systems, Springer, 2000, pp. 265–292.

[12] G. Rotstein, R. Lavie, D. Lewin, A qualitative process-oriented approach for chemical plant operations — the generation of feasible operation procedures, Computers & Chemical Engineering 16 (1992) S337 – S344, European Symposium on Computer Aided Process Engineering.

[13] A. Castelnuovo, L. Ferrarini, L. Piroddi, An incremental Petri net-based approach to the modeling of production sequences in manufacturing systems, IEEE Transactions on Automation Science and Engineering 4 (3) (2007) 424–434.

[14] M. Theißen, R. Hai, W. Marquardt, A framework for work process modeling in the chemical industries, Computers & Chemical Engineering 35 (4) (2011) 679 – 691.

[15] S. Viswanathan, C. Johnsson, R. Srinivasan, V. Venkatasubramanian, K. E. Ärzen, Automating operating procedure synthesis for batch processes: Part i. knowledge representation and planning framework, Computers & Chemical Engineering 22 (11) (1998) 1673–1685.

[16] G. Godena, ProcGraph: a procedure-oriented graphical notation for process-control software specification, Control Engineering Practice 12 (1) (2004) 99 – 111.

[17] H. A. Gabbar, A. Aoyama, Y. Naka, Recipe formal definition language for operating procedures synthesis, Computers & Chemical Engineering 28 (9) (2004) 1809–1822.

[18] E. M. Clarke, O. Grumberg, D. Peled, Model checking, MIT press, 1999.

[19] E. Németh, T. Bartha, C. Fazekas, K. M. Hangos, Verification of a primary-to-secondary leaking safety procedure in a nuclear power plant using coloured Petri nets, Reliability Engineering & System Safety 94 (5) (2009) 942–953.

[20] D. Soliman, G. Frey, Verification and validation of safety applications based on plcopen safety function blocks, Control Engineering Practice 19 (9) (2011) 929 – 946.

[21] P. J. Ramadge, W. M. Wonham, Supervisory control of a class of discrete event processes, SIAM Journal on Control and Optimization 25 (1) (1987) 206–230.

[22] J. Zaytoon, B. Riera, Synthesis and implementation of logic controllers–a review, Annual Reviews in Control 43 (2017) 152–168.

[23] Y.-F. Wang, H.-H. Chou, C.-T. Chang, Generation of batch operating procedures for multiple material-transfer tasks with Petri nets, Computers & Chemical Engineering 29 (8) (2005) 1822–1836.

[24] M.-L. Yeh, C.-T. Chang, An automata-based approach to synthesize untimed operating procedures in batch chemical processes, Korean Journal of Chemical Engineering 29 (5) (2012) 583–594.

[25] S. Dai, M. Hong, B. Guo, Synthesizing power management strategies for wireless sensor networks with uppaal-stratego, International Journal of Distributed Sensor Networks 13 (4).

[26] R. Alur, D. L. Dill, A theory of timed automata, Theoretical Computer Science 126 (2) (1994) 183–235.

[27] C.-J. Wang, Y.-C. Chen, S.-T. Feng, C.-T. Chang, Automata-based operating procedure for abnormal situation management in batch processes, Computers & Chemical Engineering 97 (2017) 220–241.

[28] J.-H. Li, C.-T. Chang, D. Jiang, Systematic generation of cyclic operating procedures based on timed automata, Chemical Engineering Research and Design 92 (1) (2014) 139–155.

[29] M. Lind, An introduction to multilevel flow modeling, Nuclear safety and simulation 2 (1) (2011) 22–32.

[30] ISA, ANSI/ISA-88.01-1995 : Batch control - part 1 : Models and terminology, The Instrumentation, Systems and Automation Society.

[31] M. Lind, H. Yoshikawa, S. B. Jørgensen, M. Yang, K. Tamayama, K. Okusa, Multilevel flow modeling of monju nuclear power plant, Nuclear safety and simulation 2 (3) (2011) 274–284.

[32] H. Meng, L. Kloul, A. Rauzy, Modeling patterns for reliability assessment of safety instrumented systems, Reliability Engineering & System Safety 180 (2018) 111–123.

[33] M. Bonfè, C. Fantuzzi, C. Secchi, Design patterns for model-based automation software design and implementation, Control Engineering Practice 21 (11) (2013) 1608 – 1619.

[34] S. Mouelhi, M.-E. Laarouchi, D. Cancila, H. Chaouchi, Predictive formal analysis of resilience in cyber-physical systems, IEEE Access 7 (2019) 33741–33758.

[35] E. Clarke, A. Biere, R. Raimi, Y. Zhu, Bounded model checking using satisfiability solving, Formal Methods in System Design 19 (1) (2001) 7–34.

[36] IEC, IEC 60848: Grafcet specification language for sequential function charts.

[37] R. Julius, M. Schürenberg, F. Schumacher, A. Fay, Transformation of Grafcet to PLC code including hierarchical structures, Control Engineering Practice 64 (2017) 173 – 194.

[38] K. G. Larsen, F. Lorber, B. Nielsen, 20 years of uppaal enabled industrial model-based validation and beyond, in: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice, Springer International Publishing, Cham, 2018, pp. 212–229.

[39] T. Cochard, D. Gouyon, J.-F. Pétin, Generation of safe plant operation sequences using reachability analysis, in: 20th IEEE International Conference on Emerging Technologies and Factory Automation, 2015.