Using a meta-model to build operational architectures of automation systems for critical processes

T. Lemattre, B. Denis, J-M. Faure
LURPA, ENS Cachan
61 av. President Wilson
94235 Cachan Cedex, France
{lemattre, denis, faure}@lurpa.ens-cachan.fr

P. Salaun
Electricite de France, R& D
6 quai Watier
78400 Chatou, France
patrick.salaun@edf.fr

J-F. Pétin
CRAN, UMR 7039 CNRS, Nancy Universite
BP 70239
54506 Vandoeuvre les Nancy, France
jean-francois.petin@cran.uhp-nancy.fr

Abstract

This paper presents first a meta-model, in the form of an UML class diagram, which describes the functional and physical architectures of automation systems for critical processes, as well as the relations between these two architectures. Then, construction of an operational architecture from functional and physical architectures is addressed. This construction must satisfy capabilities and safety-related distribution constraints; to meet this objective, a method based on reachability analysis in a network of communicating automata is proposed. The parameters of these automata are coming from instances diagrams derived from the architectures meta-model; conversely, the result of reachability analysis defines the relations between these diagrams.

1 Introduction

An automation system is characterized by three architectures [1]:

- its functional architecture, consisting of interconnected control functions, which expresses the users' needs;

- its physical architecture, consisting of controllers (PLCs or real-time industrial computers), which are in charge of executing control functions and generating signals to the process from the information gathered on this latter, and of one or several communication networks allowing data exchanges between the controllers and between the controllers and the monitoring/supervision system;

- its operational architecture, built by projecting the functional architecture onto the physical architecture.

This projection consists in assigning every element of the functional architecture to a controller, while respecting constraints that are related either to the physical capabilities of the controllers, e.g. their numbers of inputs/outputs, or to the distribution of functions on controllers, as, in the case of critical processes, the safety requirements lead to the definition of classes of functions such that two functions belonging to different classes may or may not be combined into a single controller. In the current industrial practice, this assignment of functions is a tedious and time-consuming activity, based on the expertise of designers and then performed in a non-automated way. This paper will address this problem by proposing a method to automate this activity.

It matters to note that the three architectures are used by different activities during design and validation of the automation system, such as time performances assessment, dependability attributes assessment, testing, commissioning, etc. Using a meta-model that centralizes all useful information regarding the architectures is an efficient solution to federate these activities (Figure 1). This meta-model permits that the activities share a common and consensual view of what the architectures must be. It can be used, in a Model Driven Engineering approach, to extract the necessary information that must be processed by the engineering activities which use Domain Specific Languages (DSL). This way of thinking, based on the use of UML/SysML for system specification before using dedicated models such as state-based formalisms for behavioural design, has been promoted in automation engineering by projects such as Oooneida [2] or Corfu [3] or by UML profile for Process Automation [4].
This approach will be developed in this paper, focus being put on operational architecture construction. The architectures meta-model, in the form of an UML class diagram and Object Constraint Language (OCL) expressions [5] is described in the next section. Then an automatic method to assign all functions to controllers, while respecting the capabilities and distribution constraints, is proposed in section 3. This method relies on reachability analysis on a network of communicating automata, which is then the DSL for the considered activity. The guards of the transitions of the automata model the capabilities and distribution constraints and are obtained from the OCL expressions; conversely, instances diagrams can be designed for further engineering activities once the operational architecture constructed. Two case studies illustrate this construction in section 4 and prospects for further work are sketched in Section 5.

2 Architectures meta-model

2.1 Functional architecture

A functional architecture (Figure 2) is a set of interconnected functions which receives and sends data from/to the process. Hence, the part of the meta-model (Figure 4) which describes this architecture contains two main classes: Function and Data.

- A function $f^i \in F$ is defined as a 9-tuple $(SL^i, NILp^i, NIAp^i, NOLp^i, NOAp^i, NILf^i, NIAf^i, NOLf^i, NOAf^i)$ with:
  - $SL^i \in$ Safety Level; the lower this level is, the more critical the function. In the sequel of this paper, the functions are ranked in four levels; hence, the set Safety Level is $\text{Safety Level} = \{1, 2, 3, 4\}$;
  - $NILp^i, NIAp^i \in \mathbb{N}$: numbers of respectively logic and analogic input data received from the controlled process;
  - $NOLp^i, NOAp^i \in \mathbb{N}$: numbers of respectively logic and analogic output data going to other functions.

- Data are typed; they are either logic or analogic. Moreover, they are ranked in three disconnected sets:
  - Input data, received from the process;
  - Output data, sent to the process.
  - Inter-functions data.

It matters to underline that any data issued from a function is sent either to the process or to another function, as the three sets are disconnected. Moreover, self-loops are forbidden; inter-functions data must then satisfy the following relation:

$$\text{context function}$$

$$\text{INV : } f_1 \leftrightarrow f_2$$  \hspace{1cm} (1)

2.2 Physical architecture

A physical architecture (Figure 3) is seen as a set of controllers which are connected by one or several networks. Hence, the part of the meta-model which describes this architecture contains two main classes: Controller and Network.

Figure 1. Architectures meta-model and engineering activities

Figure 2. Example of functional architecture

Figure 3. Example of physical architecture
A controller $c^j \in C$ is defined as a 5-tuple $(CF, LImax, Almax, LOmax, AOmax)$ with (cf. figure 4):

- $CF \in \text{Criticality\_Factor}$ of the controller, the lower the criticality factor is, the more dependable the controller must be; in the sequel of this paper, it will be assumed that this factor can take 3 values: Criticality\_Factor $= \{1, 2, 3\}$;

- $LImax, Almax \in \mathbb{N}$ maximum numbers of respectively logic and analogic input interfaces of the controller;

- $LOmax, AOmax \in \mathbb{N}$ maximum numbers of respectively logic and analogic output interfaces of the controller.

A network flow models communication between two controllers. It is then assumed that relation (2) holds, i.e. that the source and the target of a flow are different.

$$\text{context controller}$$

$$\text{INV} : c1 <> c2 \quad (2)$$

2.3 Operational architecture

The operational architecture is built by projecting the functional architecture onto the hardware architecture, i.e. by assigning every function to a controller. It will be assumed in what follows that:

1. a controller can host between 1 to $n$ functions;

2. a function must be assigned to one and only one controller.

These assumptions are included in the meta-model in the form of the relation between the classes Function and Controller.

If $F$ is the set of functions $f^i$, with $i \in \mathbb{N}^+$ and $C$ the set of controllers $c^j$, with $j \in \mathbb{N}^*$, the assignment of $f^i$ to the controller $c^j$ will be denoted by $c^j \leftarrow f^i$.

Moreover, each assignment has to satisfy capabilities and distribution constraints which are formalized thanks to OCL expressions below.

2.3.1 Capabilities constraints

Only the constraints on the capabilities of the controllers in terms of input and output interfaces are considered in this study. Roughly speaking, these constraints mean that the sums of inputs/outputs from/to the process of the functions which are assigned to any controller must not exceed the numbers of input/output interfaces of this controller.

The numbers of interfaces are positive integers:

$$\text{context controller}$$

$$\text{INV} : LImax > 0 \quad (3)$$

$$\text{INV} : Almax > 0 \quad (4)$$

$$\text{INV} : LOmax > 0 \quad (5)$$

$$\text{INV} : AOmax > 0 \quad (6)$$

The numbers of inputs/outputs from/to the process of a function can be obtained from the instance diagram which defines the structure of a particular functional architecture:

$$\text{context function}$$

$$\text{DEF} : /NILp = \text{self.In-Log-p} \rightarrow \text{size()} \quad (7)$$

$$\text{DEF} : /NIAp = \text{self.In-Ana-p} \rightarrow \text{size()} \quad (8)$$

$$\text{DEF} : /NOLp = \text{self.Out-Log-p} \rightarrow \text{size()} \quad (9)$$

$$\text{DEF} : /NOAp = \text{self.Out-Ana-p} \rightarrow \text{size()} \quad (10)$$

Then, the capabilities constraints are stated in OCL as follows:

$$\text{context controller}$$

$$\text{INV} : \text{function.} /NIL\rightarrow \text{sum()} \leq LImax \quad (11)$$

$$\text{INV} : \text{function.} /NIA\rightarrow \text{sum()} \leq Almax \quad (12)$$

$$\text{INV} : \text{function.} /NOL\rightarrow \text{sum()} \leq LOmax \quad (13)$$

$$\text{INV} : \text{function.} /NOA\rightarrow \text{sum()} \leq AOmax \quad (14)$$
2.3.2 Distribution constraints

These constraints are introduced to assign to a controller only functions whose safety level complies with its criticality factor. More precisely, the most critical functions \((SL = 1)\) must be assigned only to controllers whose criticality factor equals 1, the medium-critical controllers \((CF = 2)\) can accept functions whose safety level equals 2 or 3 and the less critical controllers can accept functions whose safety level equals 3 or 4. Given the instances diagrams of the functional and physical architectures, these constraints are then expressed as follows:

\[
\text{context controller}
\]

\[
\text{INV : self.CF} = 1 \implies \text{function.SL} = 1 \quad (15)
\]

\[
\text{INV : self.CF} = 2 \implies (\text{function.SL} = 2 \text{ or function.SL} = 3) \quad (16)
\]

\[
\text{INV : self.CF} = 3 \implies (\text{function.SL} = 3 \text{ or function.SL} = 4) \quad (17)
\]

3 Automatic construction of operational architecture

For a given functional architecture, it is possible to build the corresponding instances diagram from the meta-model of Figure 4; this diagram contains obviously as many instances of the class function as there are functions in the functional architecture and is one of the inputs of the method to construct automatically operational architectures; \(L\) will denote the number of functions. The other input is a parametric instances diagram of physical architecture where the number of controllers, noted \(M\), and the criticality factor of each controller are parameters. This approach allows several operational architectures be constructed from the same functional architecture; the choice of a solution among this set is under the responsibility of the automation system designer.

3.1 Principles

The aim of the method to construct automatically operational architectures is then to assign a set of functions to a set of controllers, while respecting the capabilities and distribution constraints (Figure 5).

As the capabilities constraints consider only the numbers of inputs/outputs from/to the process, the interfunctions data are no more useful data when assigning the functions. Hence, a function \(f^i\) becomes merely a 5-tuple \(\{SL^i, NI^i, NIA^i, NO^i, NOA^i\}\). Each controller \(c^j\) is still represented by a 5-tuple, but the initial value of \(CF\), when no function is assigned to this controller, will be 0. The criticality factor will be defined during functions assignment while respecting the distribution constraints.

The method proposed in this paper for the automatic assignment of functions is based on two principles:

- modelling the assignment problem as a set of competing call-response mechanisms between models, in the form of communicating automata, of assignment requests and of requests acceptances;
- investigating whether the execution of this set, which is a network of communicating automata, can lead to a state reachable from the initial state where all functions are assigned.

This proposal is based on the verification of a reachability property in a discrete state space. Such an approach has been already used successfully in [6] and [7] where scheduling issues were addressed. This contribution differs from these two references because time is not considered in an assignment problem though it must be for scheduling issues.

3.2 Toy example

Figure 6 represents a functional architecture which comprises five functions \(f^1, f^2, f^3, f^4, f^5\), whose safety level is in \(\{1, 2\} \subset SL\), and which are defined as follows:

- \(f^1 = (2, 5, 4, 1, 3)\)
- \(f^2 = (1, 5, 6, 2, 4)\)
- \(f^3 = (2, 1, 3, 6, 4)\)
- \(f^4 = (1, 6, 8, 5, 2)\)
- \(f^5 = (1, 3, 4, 5, 2)\)

These functions have to be assigned on a set of \(M=3\) controllers \(c^1, c^2, c^3\), whose capabilities are the same:

\[\forall j \in \{1, 2, 3\}, LI_{max} = AI_{max} = LO_{max} = AO_{max} = 10\]

One possible assignment of these functions to the three controllers is described in Figure 6. This solution was obtained by first assigning the function \(f^1\) to the controller \(c^1\), thus fixing the value of its criticality factor to \(CF^1 = 2\). The function \(f^2\) was then assigned to the controller \(c^2\), thus fixing the value of its criticality factor to \(CF^2 = 1\).
Then the function $f^3$ was assigned to the controller $c^1$ because its safety level is consistent with $CF^{F^1} = 2$ and the sums of the numbers of inputs/outputs of the two functions do not exceed the capabilities of the controller. The function $f^4$ was then assigned to the controller $c^3$ because the sums of logic and analogic inputs of functions $f^2$ and $f^4$ are beyond the capabilities of the controller $c^2$. The function $f^5$ was finally assigned to controller $c^2$, because the remaining capabilities of the controller $c^3$ were too small for $f^5$ be assigned to this controller.

Once all functions are assigned, the list of functions which are assigned to each controller is known. These lists are instances of the relation between the two classes controller and function in the meta model (Figure 4).

3.3 Generic models of assignment request and of requests acceptance

As mentioned in 3.1, assignment of functions is modelled in this approach as a set of competing call-response mechanisms between models of assignment requests and of requests acceptances. Figures 7 and 8 present respectively the generic model of an assignment request sent by a function and of the acceptance of requests by a controller; these models are denoted $\delta$ and $\alpha$.

3.3.1 Definition of the formalism used

The formalism used is a network of automata communicating through shared variables and synchronized by transition labels. Every transition of these automata may comprise a synchronization label, a guard (condition transition which must be true to fire the transition) and variables (numbers of input/output interfaces used, criticality factor) updates. The following conventions are used in these models:

- the initial locations are indicated by a source arc;
- the marked locations are indicated by two concentric circles;
- the location names are in bold;
- the label names are in italics and followed by an ! (resp. ?) for emission (resp. reception) labels;
- the variables updates are underlined;
- the guards (transition conditions) are in normal characters.

3.3.2 Assignment request model

The initial location of the model is 'Function not assigned'. Only one transition, which corresponds to the emission of an assignment request can be fired from this location. Once this request has been emitted, the model waits (in the location 'Assignment Possible?') for the response from an acceptance model, which can be:

- Refusal, then the model returns to the initial location;
- Ok, then the model evolves to the location "Function assigned" which is a terminal marked location.

Figure 7. Generic model of assignment request ($\delta$)

3.3.3 Requests acceptance model

Four transitions of this model include guards which are obtained from the OCL expressions which define the capabilities and distribution constraints. The guard Violation of one of the assignment constraints, for instance, mean that at least one of the constraints is not satisfied if the requesting function is assigned to the considered controller.

Some notations must be introduced to ease the description of this model.

- Let: $F_j = \{ f^i \in F | c^j \leftarrow f^i \}$ be the set of functions $f^i$ which are assigned to $c^j$;
• Let: \( I_j = \{ i \in \{ 1, ..., L \} | c^j \leftarrow f^i \} \) be the set of index of functions \( f^i \) which are assigned to \( c^j \).

Then the notations \( \sum_{i \in I_j} NILp^j \); \( \sum_{i \in I_j} NIAp^j \); \( \sum_{i \in I_j} NOLp^j \); \( \sum_{i \in I_j} NOAp^j \) represent respectively the sums of logic and analogic inputs, logic and analogic outputs from/to the process of the functions which are already assigned to controller \( c^j \) at a given moment. The initial values of these sums, at the beginning of the assignment, are obviously equal to 0.

From the initial location, which is also marked, this model can evolve to the location 'Checking constraints' only upon reception of an assignment request. The three transitions which can be fired from this latter location correspond to:

• the violation of at least one of the assignment constraints, and the label \textit{Refusal} is then emitted;

• the acceptance of a request whereas no other function has been previously assigned to the controller (guard 'First assignment' true). The variables \( \sum_{i \in I_j} NILp^j \); \( \sum_{i \in I_j} NIAp^j \); \( \sum_{i \in I_j} NOLp^j \); \( \sum_{i \in I_j} NOAp^j \), are then updated and the criticality factor \( CF^j \) is set to the value of the safety level of the function;

• the acceptance of a request whereas at least one other function has been previously assigned (guard 'Additional assignment' true). Only the variables \( \sum_{i \in I_j} NILp^j \); \( \sum_{i \in I_j} NIAp^j \); \( \sum_{i \in I_j} NOLp^j \); \( \sum_{i \in I_j} NOAp^j \), are updated. The criticality factor \( CF^j \) remains unchanged.

In the latter two cases, the label \textit{Ok} is emitted.

From the location 'Analysis of the state of the controller', two evolutions are possible which correspond to:

• the fact that all capabilities of the controller have been reached:
  \( \sum_{i \in I_j} NILp^j = LImax^j \); \( \sum_{i \in I_j} NIAp^j = Almax^j \);
  \( \sum_{i \in I_j} NOLp^j = LOMax^j \); \( \sum_{i \in I_j} NOAp^j = AOMax^j \);
  (guard 'no other possible assignment' true), then the model evolves to the marked location 'Controller saturated';

• the fact that at least one capability of the controller is not reached (guard 'other possible assignment' true), then the model evolves to the initial location 'Controller waiting'.

Figure 8. Generic model (\( \alpha \)) of requests acceptance

• as many instances \( \{ \delta^1, \delta^2, ..., \delta^M \} \) of the model in Figure 7 as there are functions,

• \( M \) instances \( \{ \alpha^1, \alpha^2, ..., \alpha^M \} \) of the model in Figure 8; the choice of the parameter \( M \) is let to the designer. An obvious value to obtain a solution is to set \( M = L \); in that case indeed, functions assignment is always possible if no function owns a number of inputs or outputs larger than the corresponding capability of controllers. From the observation of the solution yielded with \( M = L \), a more compact solution, with a smaller number of controllers, can be obtained as it will be shown in section 4.

A synchronous evolution of two automata is possible only if these two automata emit and receive one of the following label pairs:

• \textit{Request!} and \textit{Request?};

• \textit{Ok!} and \textit{Ok?};

• \textit{Refusal!} and \textit{Refusal?}.

To avoid inconsistencies such as the fact that an instance \( \alpha^i \) emits a reply to an instance \( \delta^j \) which is not the one having emitted the assignment request, the call-response mechanism must be designed as a critical section protected by a semaphore. The achievement of this critical section depends on the implementation and will not be discussed further in this paper.

3.4 Instantiated model

This model is a network of communicating automata \( NA = \delta^1 || \delta^2 || ... || \delta^M || \alpha^1 || \alpha^2 || ... || \alpha^M \) that includes:
3.5 Definition of the reachability property searched

All the functions are assigned when the marked location is reached in all the instances of δ. In this case, the active location of the instances of α may be the terminal location or the initial location, which are both marked. Hence, the reachability property to check can be informally stated as follows:

From the initial state, is it possible to reach a state of the network of automata such that the active location is a marked location in all the automata of the network?

3.6 Implementation with a formal verification tool

The techniques of formal verification by model checking [8] aim to prove that a model satisfies or does not satisfy a formal property, which may be a reachability property. It is hence natural to consider the implementation of the method proposed by using such a technique. This requires first to formally state the property searched, given in textual way in the previous section. Using the quantifiers of the CTL temporal logic, this property, noted P, can be written:

\[ P: EF \text{ Full assignment} \]

Full assignment designating the state of the network such that the active location is a marked location for all automata. This property is verified if there exists at least one trace from the initial state of the network which reaches the state Full assignment.

4 Case studies

4.1 Choice of the formal verification tool

Several model-checking-based formal verification tools, such as NuSMV, SPIN, UPPAAL, may be considered for achieving the reachability analysis on which the search for an assignment solution relies. The UPPAAL tool [9] was selected for these studies because it has a very ergonomic graphical interface and can provide an execution trace even in case of positive proof. It is important to note that only these features have motivated this choice; the ability of this tool to check properties on timed models does not constitute a selection criterion, as the communicating automata considered in this work are not timed.

4.2 First case study

This case aims to illustrate the approach. The functional architecture consists of twenty functions which are defined in Table 1, and the controllers features are as follows:

\[ \forall j \in \{1,...,M\}, \]

\[ L_{\text{max}} = \text{F}_{\text{max}} = L_{\text{Omax}} = A_{\text{Omax}} = 32 \]

A first reachability analysis with an initial number of controllers \( M = 20 \) provides a solution in which only \( N = 5 \) controllers are hosting at least one function (15 controllers are not used). By performing a new analysis with an initial number of controllers \( M = 4 \), a more compact solution can be found, in which 4 controllers are actually used. It is not possible to further reduce the number of controllers, as an analysis with \( M = 3 \) does not provide any solution. The final assignment solution for \( N = 4 \) controllers is detailed in Table 2.

<p>| Controllers features and functions distribution for the set of functions of Table 1 |
|---------------------------------|-----------------|-----------------|-----------------|-----------------|-----------------|</p>
<table>
<thead>
<tr>
<th>( c_i )</th>
<th>( F_i )</th>
<th>( \sum_{i \in \mathcal{L}} N_{\text{IP}} )</th>
<th>( \sum_{i \in \mathcal{L}} N_{\text{IAp}} )</th>
<th>( \sum_{i \in \mathcal{L}} N_{\text{LOp}} )</th>
<th>( \sum_{i \in \mathcal{L}} N_{\text{OAp}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_i )</td>
<td>( F_i )</td>
<td>( \sum_{i \in \mathcal{L}} N_{\text{IP}} )</td>
<td>( \sum_{i \in \mathcal{L}} N_{\text{IAp}} )</td>
<td>( \sum_{i \in \mathcal{L}} N_{\text{LOp}} )</td>
<td>( \sum_{i \in \mathcal{L}} N_{\text{OAp}} )</td>
</tr>
<tr>
<td>( c_1 )</td>
<td>( \text{F}^1, \text{F}^2 )</td>
<td>30</td>
<td>26</td>
<td>32</td>
<td>23</td>
</tr>
<tr>
<td>( c_2 )</td>
<td>( \text{F}^1, \text{F}^2, \text{F}^3 )</td>
<td>23</td>
<td>32</td>
<td>22</td>
<td>20</td>
</tr>
<tr>
<td>( c_3 )</td>
<td>( \text{F}^1, \text{F}^2, \text{F}^3, \text{F}^4 )</td>
<td>28</td>
<td>27</td>
<td>21</td>
<td>21</td>
</tr>
</tbody>
</table>

4.3 Second case study

To assess scalability of the proposal, a study based on 200 functions was subsequently undertaken. These 200 functions are all different from each other, and their characteristics are randomly distributed as follows: \( \forall i \in \{0,...,L\}, \)

\[ SL_i \in \{1,2,3,4\}, \{N\text{ILp}\} = \{0,...,9\}, \{N\text{IAp}\} = \{0,...,9\}, \]

\[ N\text{LOp}\} = \{0,...,9\}, \{N\text{OAp}\} \in \{0,...,9\}. \]

The controllers used are all the same and their characteristics are: \( \forall j \in \{1,...,M\} \),
\[ L_{\text{max}} = A_{\text{max}} = L_{\text{O\ max}} = A_{\text{O\ max}} = 32. \]

A first reachability analysis was conducted, providing an assignment solution in which some controllers are not hosting any function; the number of really useful controllers is then \( N = 37 \). The analysis performed with \( M = 36 \) also provides a solution. The number of controllers required to achieve the operational architecture cannot however be reduced further, as the analysis performed with \( M = 35 \) controllers does not provide any solution. The assignment solution hence uses at least \( N = 36 \) controllers.

Table 3 shows the durations of the different reachability analyses conducted in this case study. These values show that the approach proposed is quite feasible in the industrial context of operational architectures design.

<table>
<thead>
<tr>
<th>Initial numbers of controllers</th>
<th>200</th>
<th>37</th>
<th>36</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration</td>
<td>110 s</td>
<td>46 s</td>
<td>50 s</td>
</tr>
</tbody>
</table>

### 5 Conclusion and Perspectives

The contribution of this paper is twofold. First an architectures meta-model that contains all information on the architectures of automation systems for critical processes has been proposed; OCL expressions have been introduced to express the generic constraints on the operational architecture. Second, a method to construct automatically an operational architecture from instances diagrams and the OCL expressions has been presented; this proposal is based on a novel modelling of the assignment of functions and reachability analysis of a network of communicating automata. These results will allow in a near future to focus on validation of operational architectures, e.g. by time performances assessment; this will imply to extend the architectures meta-model so as to introduce time features of the classes.

### References


