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

Thibault Lemattre, Bruno Denis, Jean-Marc Faure, Jean-François Pétin, Patrick Salaün

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. Salaün
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 addresses the design of the operational architecture of a critical system control. This design results from the allocation of control functions onto physical devices by taking into account numerous constraints such as capability, safety, time performance or reliability constraints. This paper focuses on the two first ones, capability and safety constraints by proposing a method based on reachability analysis in a network of communicating automata. The link with complementary studies about time performance or reliability constraints is established using an UML architecture meta-model that captures and shares information about control architectures. Automata models and their parameters used for reachability analysis are derived from this meta-model and, conversely, generated operational architectures give rise to a set of meta-model instances that can be used as input for complementary evaluation.

1 Introduction

The architecture of an automation system can be characterized by three complementary views [1]:

- the functional architecture, consisting of interconnected control functions, which expresses the users' needs; each function operates a transformation from inputs (from the process or from other functions) to outputs (to the process or to other functions); the functions are leave functions which are considered as indivisible; moreover, in the case of critical systems, functions are characterized by a safety-level that represents how critic is the impact of their failure on material or human being;
- the physical architecture, consisting of controllers (PLCs or real-time industrial computers), which provides hardware resources for executing the control functions and generating signals towards the process according to 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; in the case of critical systems, the devices involved in the physical architecture are characterized by an integrity factor, that represents the reliability of the device, thanks for example to internal hardware redundancies;
- the operational architecture, built by allocating the leave functions of the functional architecture onto the control devices involved in the physical architecture.

The design of operational architecture must be done by taking into account various and numerous constraints or criteria such as:

- capabilities constraints that are related to the physical capabilities of the controllers (such as energy, memory, numbers of inputs/outputs, CPU charge, etc.);
- safety constraints that are related to the deterministic allocation rules answering safety purposes; for example, the function safety level must be compliant with the controller integrity factor in which it is allocated (such as highest critical functions should be implemented in most reliable controllers); enabling or disabling the combination of different functions within a single controller (due to functional redundancies, commissioning or testing constraints, etc.) is another example of safety constraint;

- time performance constraints, for example the time needed by the control functions to react to stimuli occurrence from the process; obviously, the quality of service of the communication networks that link the controllers would have a crucial impact on time constraint analysis as well as the functional chain distribution among several controllers;

- reliability constraints, that are related to both controller and functional reliability and may impact the design of operational architecture in terms of functional and physical redundancies.

It clearly appears that the problem of designing an operational architecture that fulfil all these constraints cannot be addressed by a single technique or model: the two first constraints can be addressed by using deterministic and untimed model for constraint solving while the two last ones require respectively using temporal and stochastic models. In the current industrial practice, design of operational architecture is a tedious and time-consuming activity, based on the expertise of designers and then performed in a non-automated way. Note that actuators and sensors location cannot provide help since the system architecture is basically centralized (one or several computing centres) for safety constraints (protected area against fire, external attacks, etc.).

This paper aims to automate part of this activity and focuses on the two first constraints related to capability and safety issues:

- generation an operational architecture that fulfil capability and safety constraints is seen as a constraint solving problem and is addressed using an original approach based on reachability analysis on a discrete state space using communicating automata;

- a meta-model that centralizes all useful information regarding the architecture is sketched out to share a common and consensual information about the operational architecture between the various design activities; the proposed meta-model is built to integrate capabilities and safety constraints the paper is focusing on, as a basis to be further enriched by time performance and reliability constraints.

In section 2, the architecture meta-model is given using UML class diagram to represent the objects of functional and physical architectures while Object Constraint Language (OCL) expressions [2] allows the description of capabilities and safety constraints to be applied for operational architecture design. Illustrative examples of capability and safety constraints are provided by OCL constraints respectively related to the number of inputs/outputs and to the compliance between controller integrity factor and the function safety level. Then, an automatic method to allocate all leave functions to controllers, while respecting the capabilities and safety constraints identified in the architecture meta-model, is proposed in section 3. This method relies on reachability analysis on a network of communicating automata whose transition guards implements OCL constraints; conversely, the result of the first operational architecture design must be translated into meta-model instance diagrams to be used for further engineering activities (temporal performance evaluation, reliability analysis, etc.). Two case studies illustrate this way of building operational architecture in section 4 and prospects for further work are sketched in Section 5.

2 Architectures meta-model

The aim of the architecture meta-model is to capture useful information about functional, physical and operational architecture in order to facilitate the analysis of the control architecture with regards to several constraints or criteria and using various modelling and analysis techniques. Main benefit rely in the ability for various engineering activity (architecture sizing according to capabilities and safety constraints, architecture time performance analysis, architecture reliability analysis, etc.) to share common concepts (functions, controllers, inputs/outputs etc) and their attributes using UML class diagrams and to formalise the various constraints to be satisfied using Object Constraint Language (OCL) expressions. This Model Driven Engineering 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 [3] or Corfu [4] or by UML profile for Process Automation [5].

The paper focuses on the automatic design of the operational architecture based on capability constraint related to the number of I/O and safety constraint related to the relationship between function safety level and controller integrity factor. This first attempt is currently being extended using OCL to capture additional capability or safety constraints within the context of the industrial project that supports this work.

2.1 Functional architecture

A functional architecture (Figure 1) is a set of interconnected functions which receives and sends data from/to the process. Hence, the part of the meta-model (Figure 3) 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 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 sent to the controlled process;

– $NILf^i, NIAf^i \in \mathbb{N}$: numbers of respectively logic and analogic input data coming from other functions;

– $NOLf^i, NOAf^i \in \mathbb{N}$: 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) and 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 (a same data cannot act both as input and output for a same function) and the flow is directed; inter-functions data must then satisfy the following relation:

context function

$$\textbf{INV} : f1 <> f2 \qquad (1)$$

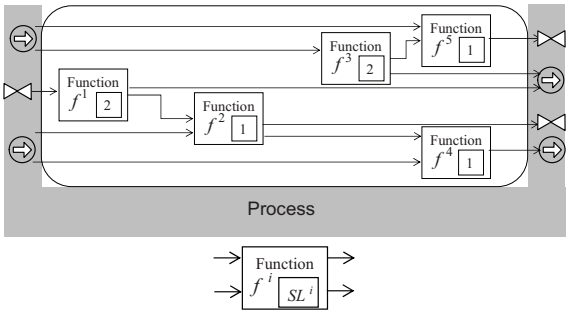

Figure 1. Example of functional architecture

2.2 Physical architecture

A physical architecture (Figure 2) 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*.

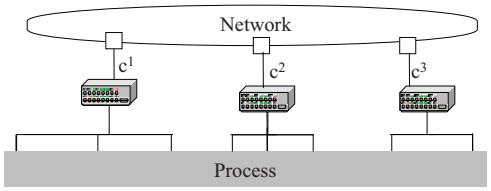

Figure 2. Example of physical architecture

- A controller $c^j \in C$ is defined as a 5-tuple $(CF, LImax, AImax, LOmax, AOmax)$ with (cf. figure 3):

  – $CF \in$ 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 $= \{1,2,3\}$;

  – $LImax, AImax \in \mathbb{N}$ 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.

context controller

$$\textbf{INV} : c1 <> c2 \qquad (2)$$

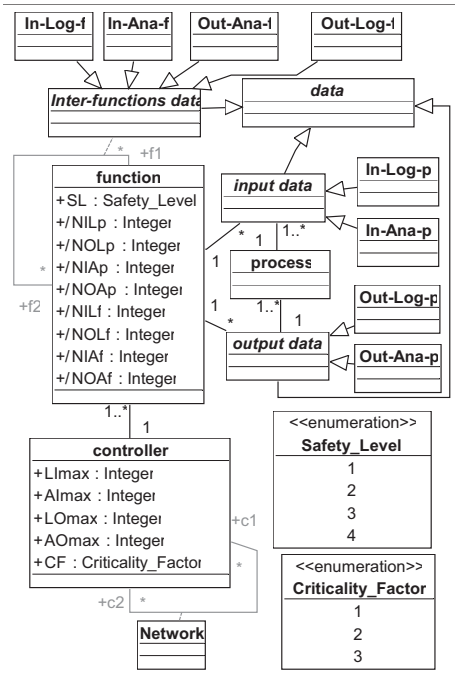

Figure 3. Architectures meta-model

2.3 Operational architecture

The operational architecture is built by allocating the leave functions of the functional architecture onto the controllers of the physical architecture. Following assumptions will be taken:

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

2. a function must be allocated 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 allocation of f^i to the controller c^j will be denoted by $c^j \leftarrow f^i$.

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

2.3.1 Capabilities constraints

Capability constraints can be modelled using OCL expressions in terms of inequations where controller attributes are involved in. In our study, capability constraints (and controller attributes) are limited to the number of I/O interfaces. Roughly speaking, these constraints mean that the sums of inputs/outputs from/to the process of the functions that are allocated to a given controller must not exceed the numbers of input/output interfaces of this controller.

The numbers of interfaces are positive integers:

context controller

$$\textbf{INV} : LImax > 0 \tag{3}$$

$$\textbf{INV} : AImax > 0 \tag{4}$$

$$\textbf{INV} : LOmax > 0 \tag{5}$$

$$\textbf{INV} : AOmax > 0 \tag{6}$$

The numbers of inputs/outputs from/to the process of a given function are defined by the number of existing relationship between this functions and its associated I/O; this information can be built from the instance diagram that defines a particular functional architecture using the OCL *size()* function:

context function

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

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

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

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

Then, the capabilities constraints state that the sum of I/O functions allocated to a given controller must be lower or equal than the controller I/O capability; it is modelled using OCL *sum()* function as follows:

context controller

$$\textbf{INV} : function./NIL -> sum() \leq LImax \tag{11}$$

$$\textbf{INV} : function./NIA -> sum() \leq AImax \tag{12}$$

$$\textbf{INV} : function./NOL -> sum() \leq LOmax \tag{13}$$

$$\textbf{INV} : function./NOA -> sum() \leq AOmax \tag{14}$$

2.3.2 Safety constraints

One of the main safety constraints is related to the compliance between function safety level and controller integrity factor in which the function is allocated. More precisely, the most critical functions ($SL = 1$) must be allocated 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:

context controller

$$\textbf{INV} : \text{self.}CF = 1 \text{ implies function.}SL = 1 \tag{15}$$

$$\textbf{INV} : \text{self.}CF = 2 \text{ implies(function.}SL = 2 \text{ or function.}SL = 3) \tag{16}$$

$$\textbf{INV} : \text{self.}CF = 3 \text{ implies(function.}SL = 3 \text{ or function.}SL = 4) \tag{17}$$

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 3; this diagram contains obviously as many instances of the class function as there are 'leave' 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 'leave' 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 to be defined from the same functional architecture while satisfying all capability and safety constraint; the choice of a solution among this set is under the responsibility of the automation system designer and can be represented as particular relationships between function and controller instances of the meta-model.

3.1 Principles

The aim of the method to construct automatically operational architectures is then to allocate a set of functions to a set of controllers, while respecting the capabilities and safety constraints (Figure 4).

As the capabilities constraints consider only the numbers of inputs/outputs from/to the process, the inter-functions data are no more useful data when allocating the functions. Hence, a function f^i becomes merely a 5-tuple $f^i = (SL^i, NILp^i, NIAp^i, NOLp^i, NOAp^i)$. Each controller c^j is still represented by a 5-tuple, but the initial value of CF, when no function is allocated to this controller, will be 0. The criticality factor will be defined during functions allocation while respecting the safety constraints.

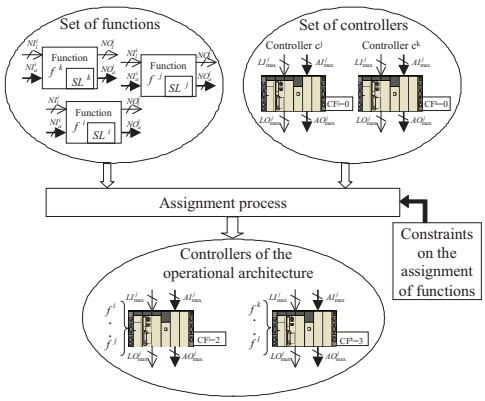

Figure 4. Aim of the construction

The problem of allocation can then be seen as a bin packing problem than can be solved by optimisation methods such as linear programming or heuristic even if suffering from the exponential explosion problem. Approaches proposed by [6] and [7] showed the effectiveness of the timed automata and reachability analysis to address scheduling problems thanks to a modular, parametric and instanciable modelling. The method proposed in this paper takes advantage of these approaches but differs from them because time is not considered in an allocation problem. Based on communicating automata, the automatic allocation of functions is based on two principles:

- modelling the allocation problem as a set of competing call-response mechanisms between models, in the form of communicating automata, of allocation 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 allocated.

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 allocated on a set of M=3 controllers c^1, c^2, c^3, whose capabilities are the same: $\forall j \in \{1,2,3\}, LImax = AImax = LOmax = AOmax = 10$

One possible allocation of these functions to the three controllers is described in Figure 5. This solution was obtained by first allocating 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 allocated to the controller c^2, thus fixing the value of its criticality factor to $CF^2 = 1$. Then the function f^3 was allocated to the controller c^1 because its safety level is consistent with $CF^1 = 2$ 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 allocated 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 allocated to controller c^2, because the remaining capabilities of the controller c^3 were too small for f^5 be allocated to this controller.

Once all functions are allocated, the list of functions which are allocated to each controller is known. These lists are instances of the relation between the two classes controller and function in the meta model (Figure 3).

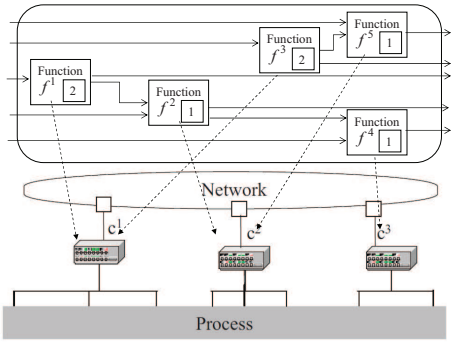

Figure 5. Example of allocation of 5 functions on 3 controllers

3.3 Generic models of allocation request and of requests acceptance

As mentioned in 3.1, allocation of functions is modelled in this approach as a set of competing call-response mechanisms between models of allocation requests and of requests acceptances. Figures 6 and 7 present respectively the generic model of an allocation request sent by a function and of the acceptance of requests by a controller; these models are denoted δ and α.

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 allocation request model

The initial location of the model is 'Function not allocated'. Only one transition, which corresponds to the emission of an allocation request can be fired from this location. Once this request has been emitted, the model waits (in the location 'allocation 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 allocated" which is a terminal marked location.

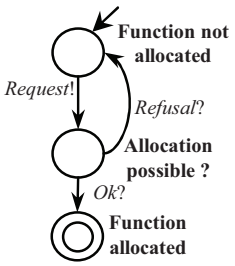

Figure 6. Generic model of allocation request (δ)

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 safety constraints. The guard Violation of one of the allocation constraints, for instance, mean that at least one of the constraints is not satisfied if the requesting function is allocated 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 allocated 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 allocated to c^j.

Then the notations $\sum_{i \in I_j} NILp^i; \sum_{i \in I_j} NIAp^i; \sum_{i \in I_j} NOLp^i; \sum_{i \in I_j} NOAp^i$ represent respectively the sums of logic and analogic inputs, logic and analogic outputs from/to the process of the functions which are already allocated to controller c^j at a given moment. The initial values of these sums, at the beginning of the allocation, 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 allocation request. The three transitions which can be fired from this latter location correspond to:

- the violation of at least one of the allocation constraints, and the label *Refusal* is then emitted;
- the acceptance of a request whereas no other function has been previously allocated to the controller (guard 'First allocation' true). The variables $\sum_{i \in I_j} NILp^i; \sum_{i \in I_j} NIAp^i; \sum_{i \in I_j} NOLp^i; \sum_{i \in I_j} NOAp^i$, 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 allocated (guard 'Additional allocation' true). Only the variables $\sum_{i \in I_j} NILp^i; \sum_{i \in I_j} NIAp^i; \sum_{i \in I_j} NOLp^i; \sum_{i \in I_j} NOAp^i$, are updated. The criticality factor CF^j remains unchanged.

In the latter two cases, the label *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 : $\sum_{i \in I_j} NILp^i = LImax^j$; $\sum_{i \in I_j} NIAp^i = AImax^j$; $\sum_{i \in I_j} NOLp^i = LOmax^j$; $\sum_{i \in I_j} NOAp^i = AOmax^j$; (guard 'no other possible allocation' 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 allocation' true), then the model evolves to the initial location 'Controller waiting'.

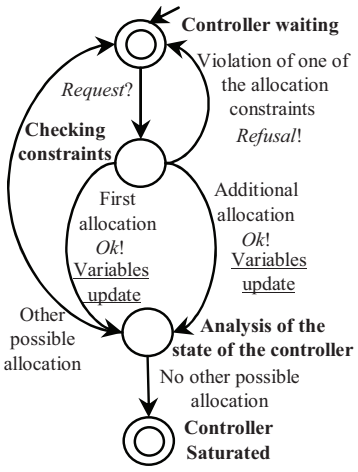

Figure 7. Generic model (α) of requests acceptance

3.4 Instantiated model

This model is a network of communicating automata $NA = \delta^1 \| \delta^2 \| ... \| \delta^L \| \alpha^1 \| \alpha^2 \| ... \| \alpha^M$ that includes:

- as many instances $(\delta^1, \delta^2, ..., \delta^L)$ of the model in Figure 6 as there are functions,
- M instances $(\alpha^1, \alpha^2, ..., \alpha^M)$ of the model in Figure 7; 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 allocation 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:

- *Request!* and *Request?*;
- *Ok!* and *Ok?*;
- *Refusal!* and *Refusal?*.

To avoid inconsistencies such as the fact that an instance α^i emits a reply to an instance δ^j which is not the one having emitted the allocation 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.

3.5 Definition of the reachability property searched

All the functions are allocated 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 *Full allocation*'

Full allocation 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 allocation*.

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 allocation 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. In the following case studies, the UPPAAL parameters have been set so that reachability analysis be done depth first to fasten analysis.

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: $\forall j \in \{1,...,M\}$,
$LImax^j = AImax^j = LOmax^j = AOmax^j = 32$

Table 1. Functions description

Functions	SL^i	$NILp^i$	$NIAp^i$	$NOLp^i$	$NOAp^i$
1	1	3	5	2	4
2	1	4	6	1	3
3	2	3	7	2	1
4	2	3	4	4	3
5	2	7	5	6	2
6	2	7	2	8	6
7	1	4	5	2	4
8	2	3	6	4	5
9	1	3	7	4	2
10	1	7	2	6	4
11	3	3	5	2	4
12	4	4	6	1	3
13	3	3	7	2	1
14	4	3	4	4	3
15	3	7	5	6	2
16	2	7	2	8	6
17	4	4	5	2	4
18	3	3	6	4	5
19	4	3	7	4	2
20	1	7	2	6	4

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 allocation solution for $N = 4$ controllers is detailed in Table 2.

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:$\forall i \in \{0,...,L\}$, $SL^i \in \{1,2,3,4\}$, $\{NILp^i \in \{0,...,9\}$, $NIAp^i \in$

Table 2. Controllers features and functions distribution for the set of functions of Table 1

c^j	CF^j	F_j	$\sum_{i\in I_j} NILp^i$	$\sum_{i\in I_j} NIAp^i$	$\sum_{i\in I_j} NOLp^i$	$\sum_{i\in I_j} NOAp^i$
c^1	3	$\{f^{12}, f^{13}\}$	7	13	3	4
c^2	2	$\{f^3, f^4, f^5, f^6, f^8, f^{16}\}$	30	26	32	23
c^3	3	$\{f^{11}, f^{14}, f^{15}, f^{17}, f^{18}, f^{19}\}$	23	32	22	20
c^4	1	$\{f^1, f^2, f^7, f^9, f^{10}, f^{20}\}$	28	27	21	21

$\{0,...,9\}$, $NOLp^i \in \{0,...,9\}$, $NOAp^i\} \in \{0,...,9\}$.
The controllers used are all the same and their characteristics are: $\forall j \in \{1,...,M\}$,
$LImax^j = AImax^j = LOmax^j = AOmax^j = 32$.
A first reachability analysis was conducted, providing an allocation 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 allocation 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 3. Duration of the reachability analysis with L=200

Initial numbers of controllers	200	37	36
Duration	110 s	46 s	50 s

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 allocation of functions and reachability analysis of a network of communicating automata.

Ongoing works are aiming at introducing new capabilities and safety constraints and to propose not only one allocation solution but a set of solutions in the automatic construction of an operational architecture. Comparison to other methods, like integer linear programming, is also planned on the basis of several case studies.

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

[1] F. Simonot Lion and J-P. Elloy. An Architecture Description Language for In-Vehicle Embedded System Development. In *15th Triennial World Congress of the International Federation of Automatic Control*, Barcelona, Spain, 2002. Elsevier Science.

[2] M. Richters and M. Gogolla. On Formalizing the UML Object Constraint Language OCL. In Tok-Wang Ling, Sudha Ram, and Mong Li Lee, editors, *Conceptual Modeling-ER'98*, volume 1507 of *Lecture Notes in Computer Science*, pages 449–464. Springer Berlin / Heidelberg, 1998.

[3] F. Auinger, R. Brennan, J. Christensen, L.M. Lastra, and V. Vyatkin. Requirements and solutions to Software encapsulation and engineering in next generation manufacturing systems: OOONEIDA approach. *International Journal of Computer Integrated Manufacturing*, 18(7):p 572–585, 2005.

[4] C. Tranoris and K. Thramboulidis. A tool supported engineering process for developing control applications. *Computers in Industry*, 57(5):p 462–472, 2006.

[5] U. Katzke and B. Vogel-Heuser. UML-PA as an Engineering Model for Distributed Process Automation. In *16th IFAC world Conference*, Prague, 2005.

[6] G. Behrmann, E. Brinksma, M. Hendriks, and A. Mader. Production Scheduling by Reachability Analysis - A Case Study. *International Parallel and Distributed Processing Symposium*, Volume 3:p. 140–148, 2005.

[7] S. Subbiah and S. Engell. Short-Term Scheduling of Multi-Product Batch Plants with Sequence-Dependent Changeovers Using Timed Automata Models. In *20th European Symposium on Computer Aided Process Engineering*, volume 28, pages 1201–1206, 2010.

[8] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen. *Systems and Software Verification*. Springer, 2001.

[9] G. Behrmann, J. Bengtsson, A. David, K. Larsen, P. Pettersson, and W. Yi. UPPAAL implementation secrets. In *Proc. of 7th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT)*, Oldenburg, Germany, September 2002.