Automated Partitioning of Concurrent Discrete Event Systems for Distributed Behavioural Identification

Jeremie Saives, Gregory Faraut, Jean-Jacques Lesage

Automated Partitioning of Concurrent Discrete Event Systems for Distributed Behavioural Identification

Jeremie Saives, Gregory Faraut, *Member, IEEE,* and Jean-Jacques Lesage, *Member, IEEE*

***Abstract*—The aim of behavioural identification of Discrete Event Systems is to build, from a sequence of observed inputs/outputs events, an understandable model that exhibits both the direct relations between inputs and outputs events (i.e. the observable behaviour of the system) and the internal state evolutions (i.e. the unobservable behaviour). Since parallelism hinders the construction of monolithic models, distributed identification builds instead models of subsystems. This paper proposes an automated partitioning of the system, optimal regarding the readability of the identified distributed models, thus fitting reverse-engineering purposes. To solve the optimization problem, a first solution is extracted from the observable behaviour, then additional solutions are computed by agglomerative clustering. The approach is applied to a benchmark, resulting in an adequate functional partition.**

***Note to Practitioners*—Identification is an approach to obtain models of an existing closed-loop Discrete Event System from an observed I/O sequence, discovering both sequential and concurrent processes within a same system. The result is an understandable and compact model, that approximates the observed behaviour and can be used for reverse-engineering. To get better insight on the behaviour of the system, splitting it into subsystems and studying distributed models might be easier. Besides, identifying smaller subsystems reduces the computational cost. The main contribution of this article is therefore a partitioning approach for distributed DES identification. Only the knowledge of the I/Os and the observed I/O sequence are required to provide a partitioning adapted to reverse-engineering and compact distributed models, in reasonable computation times.**

***Index Terms*—Discrete-event systems, Petri-nets, Identification, Distributed models**

I. Introduction

DISCRETE Event Systems (DES) are systems whose evolutions are trigerred by asynchronous discrete events. DES are known for their ability to display massive concurrency, making state-based monolithic models impossible to use, or even build, due to the classical state-space explosion problem. Distributed modelling is then considered: for instance, fault diagnosis with distributed automatata models is considered in [1], or with Place-bordered Petri Nets in [2]. These models are often built on expert knowledge, following rules and guidelines to ensure the conservation of their properties during the construction, thus making them fit for analysis [3].

J. Saives, G. Faraut and J.-J. Lesage are with the Automated Production Research Laboratory (LURPA), ENS Cachan, Univ. Paris-Sud, Université Paris-Saclay, 94235 Cachan, France.

email: *firstname.lastname@ens-paris-saclay.fr*

When lacking knowledge about an existing system, modelling can be performed by identification, provided Input Output data collected from the system. Several methods identifying DES are surveyed in [4], most being devoted to fault diagnosis. A few approaches are instead focused on reverse-engineering of closed-loop DES by various classes of Petri nets [5] [6] [7]. The objective is to provide a compact model expliciting the behaviour of the system and the relationships between its components.

However, regardless of the method used, massively concurrent DES are hard to identify: concurrency is troublesome to discover in a sequential observation, and computational cost increases with concurrency and size. Distributed identification bypasses complexity, provided a system partition. Said partition might be *a priori* known, or not, as in a blackbox identification approach. Furthermore, despite the expertise, the provided partition might not be the most adequate to a given objective. Therefore, automated partitioning aims at finding a partition optimal regarding a given purpose. For instance, the minimization of the exceeding language is pointed as an objective quality related to fault diagnosis in [8], and an optimization approach was accordingly proposed in [8] and [9]. In this paper, the objective of reverse-engineering is instead considered, requiring different objective functions for partitioning.

Consequently, a new partitioning approach is proposed, fitting for reverse-engineering purposes. It is seen as an optimization problem whose objective is to minimize the structural complexity of the identified distributed models, thus improving their understandability. The identification method from [7] is chosen to build Interpreted Petri Nets, successively building its observable, then unobservable part. Observable fragments are used to obtain a first solution of the partitioning problem. Finally, a clustering approach is proposed to group the different fragments into bigger subsystems, until a compromise is found between the size of the subsystems and the readability of the identified models. The remainder of the paper is organized as follows. Section 2 presents related works in the litterature. Section 3 recalls some notations and exposes the identification method. The optimization problem is set in Section 4. Section 5 exposes the proposed clustering approach, and finally, the approach is applied to a benchmark in Section 6.

II. Related work and motivation

A. Related work

Discrete Event System identification consists in building a mathematical model of a system from data collected by observing it. Due to a finite time of observation, collected data is necessary partial; consequently, identification is often presented as a sub-problem of *net synthesis*, where the whole behaviour of the system is known, as well as counter-examples. This difference is pointed out in [10]. Both problems have been treated in several works, surveyed in [4]. Notable contributions to solve these problems, notably the synthesis one, are Region Theory [11] or the resolution of Integer Linear Programs [12], later used for fault diagnosis by identifying faulty nets [13] [14].

Regarding identification of automated DES, the gap between technology and event-based abstraction should be minded as well. Minding it, monolithic finite automata are identified with the objective of fault diagnosis in [15], extended to distributed automata in [8]. These models, though adapted for fault diagnosis, are however less compact than Petri Nets to express concurrent behaviours, and do not highlight the reactive nature of an automated DES, *i.e.* causal relationships between inputs and outputs. For reverse engineering, Interpreted Petri Nets have been considered to express both I/O relationships, called *observable behaviour*, and unobservable dependencies implying internal variables, called *unobservable behaviour*. Only the outputs are studied in [16] or [17], whereas both inputs and outputs are considered in the approaches proposed in [5], [18] or [7]. While the former two approaches provide State-Graphs, a specific class of Petri Nets behaving like automata, the latter approach successfully extracts and represents compactly concurrent behaviour. Finding unobservable concurrent behaviour is however the hardest task, unless the hypothesis of observation completeness can be made [7]. Without it, polynomial algorithms can not guarantee a correct result and require to correct the model [17] [19], whereas algorithms guaranteeing a result run in exponential time [20].

Identification shares its core concept with *Process Mining*, which aims at getting models of businesses from registered logs [21], to improve said businesses. Activities have a role similar to the events, or the transition labels in Petri Nets, and relations (causal or concurrent) between these activities are extracted from the log; Process Mining algorithms are surveyed in [22]. The main difference lies in the absence of reactive behaviour in the considered processes. Nevertheless, problematics similar to DES identification are found; for instance, regarding the incompleteness of the observation, Conjoint Occurence Classes are proposed in [23] to infer missing relations. As well, discovering concurrency is a hard task, and algorithms providing exact solutions suffer from exponential complexity [24].

Distributed identification provides a way of limiting the calculus cost, in addition to providing separate, more understandable models, and insight on the system decomposition, thus satisfying for reverse engineering. It requires a partition of the I/Os, and can therefore be coupled to the task of finding an adequate one. The automated partitioning problem has been seldom dealt with. The authors of [8] and [9] propose to solve an optimization problem, whose objective function is designed for fault diagnosis. Distributed approaches have also been considered in Process Mining [25], where the partitioning problem is percieved as a Graph Partitioning problem [26]. Edges of the graph represent a causal relationship between the nodes which represent activities, and the aim is to minimize the number of edges required to break the graph into subgraphs. However, it requires completeness of the causal relationship, which is impossible to obtain with incomplete observation.

B. Motivation

The interest of identifying distributed models is twofold. On one hand, the main roadblock when dealing with DES is concurrency: representing all possible parallel executions of a DES leads to state-space explosion, hindering the construction of a monolithic model, especially with automata. However, by splitting a system into subsystems, with reduced concurrency within each, distributed models are easier to achieve.

On the other hand, identification approaches also suffer from size and concurrency of a system. Notably, inferring unobservable, concurrent behaviour from sequences of events which exhibit only a partial subset of all possible executions, is the hardest task. Any identification approach aims at building a model that reproduces at least the observed behaviour. Computationally efficient approaches are heuristics that can not guarantee the reproducibility in every case, unless the hypothesis of complete observation is made, but is not in this paper. Exact approaches which guarantee reproducibility are however computationally expensive, notably regarding the number of transitions in the model.

In order to use exact identification approaches despite an unavoidable computation cost, distribution helps by limiting size and concurrency of subsystems. Hence, identified distributed models can be built with guaranteed properties such as reproducibility, at moderate cost.

We choose to combine distribution and exact methods in this paper, and propose a method to automatically partition the system in favor of distribution. Since the objective is reverse-engineering, an adequate partitioning approach is proposed, inspired by the one proposed in [9]. The identification method used in this paper was proposed in [7]. It uses however an approximate method to compute the unobservable behaviour; the exact method from [20] was used instead to compute it. The identification approach is briefly presented in next section.

III. Notations and background on behavioural identification

A. Notations

A system SYS is a set of $n = |\mathbb{U}| + |\mathbb{Y}|$ inputs $\mathbb{U} = \{u_1, \ldots, u_{|\mathbb{U}|}\}$ and outputs $\mathbb{Y} = \{y_1, \ldots, y_{|\mathbb{Y}|}\}$, the notation io_i designing either one indifferently ($SYS = \{io_1, \ldots, io_n\}$). An I/O subsystem $SUB_k \subseteq SYS$ is a subset of the main system. If overlapping of I/O subsystems is allowed, the intersection of two SUB_k can be non empty. To each SUB_k

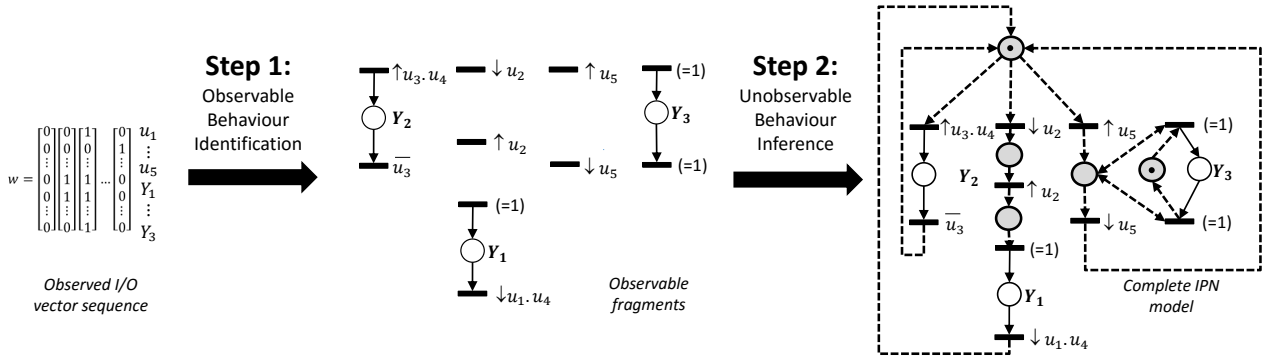


Fig. 1. Principle of the identification procedure in two steps

is associated a net N_k, identified by the chosen identification method.

In this paper, Interpreted Petri Nets are chosen to model the identified DES. A Petri Net (PN) system is a bipartite digraph represented by the 5-tuple $G = (P, T, I, O, M_0)$ where: $P = \{p_1, p_2, \ldots, p_{|P|}\}$ and $T = \{t_1, t_2, \ldots, t_{|T|}\}$ are finite sets of vertices named places and transitions respectively, $I(O) : P \times T \to \{0, 1\}$ is a function representing the edges going from places to transitions (from transitions to places), and $M_0 : P \to \mathbb{N}$ is the initial marking. An Interpreted Petri Net (IPN) [27] is a 6-tuple $N = (G, \mathbb{U}, \Sigma, \lambda, \mathbb{Y}, \varphi)$, where:

- G is a PN system
- $\Sigma = \{\uparrow u_i, \downarrow u_i \mid u_i \in \mathbb{U}\}$ the set of input events.
- $\lambda : T \to \{0, 1\}$ the labelling function of transitions. $\forall t_i \in T, \lambda(t_i) = F_i(\mathbb{U}) \bullet G_i(\Sigma)$ where:
  - $F_i : \mathbb{U} \to \{0, 1\}$ is a boolean function depicting the conditions on the levels of the inputs to fire t_i
  - $G_i : \Sigma \to \{0, 1\}$ is a boolean function depicting the conditions on the input events to fire t_i

  $\lambda(t_i) = 1$ iff $F_i(\mathbb{U}) = 1 \wedge G_i(\Sigma) = 1$
- $\varphi : R(G, M_0) \to \{0, 1\}^{|\mathbb{Y}|}$ the output function that returns the value of the outputs given a marking of the net.

In this paper, a specific subclass of IPNs is considered, where each output is associated to exactly one place, called observable (P_{Obs}). Remaining places, who are not associated to any output, are called unobservable (P_{Unobs}) and are greyed out in graphical representations.

B. Behavioural identification for reverse-engineering

The system to identify is considered *blackbox*, *i.e.* only the I/Os are known and can be observed. Structure and internal variables of the system are unknown. Therefore, the input of the identification problem we consider is only one or multiple I/O vector sequence(s).

The aim is to identify expressive models who explicitly represent the relationships between I/Os, and compactly represent behaviours such as choices and concurrency. The interpreted model must also reproduce the observed behaviour. We used the principle of the identification approach presented in [7]. It consists in two steps, illustrated by Figure 1: identifying the observable behaviour, then inferring the unobservable one.

To perform the observable identification, we used the first step of [7]. It extracts directly causal relationships between inputs and outputs from the observed I/O sequence, and translates them into observable IPN fragments, composed of observable places and transitions. The transitions are labelled with conditions on the input events and levels that cause the output events. Transitions might not be connected to any observable place; in that case, the labelling input could not be causally related to any output. The set of these unrelated inputs is written $\mathbb{D}$.

Then, the second step infers unobservable dependencies implying internal variables, such as delayed causalities or memory effects, and agregate them into unobservable places. These places connect the observable fragments, often leading to a strongly connected net. This step is the hardest part of the procedure. The method proposed in [7] to discover unobservable places being approximate, we chose instead the approach presented in [20]. Unobservable places are characterized by specific patterns in the observed sequence. Should a pattern be found in the sequence, then the associated unobservable places can be added to the net. The net is therefore built by iteratively discovering patterns, and adding places. At each step of the construction, two properties are guaranteed by a theorem: the observed sequence is reproducible, and the net is 1-bounded. The algorithm runs in exponential complexity regarding the number of transitions.

C. Problem statement and proposed procedure

The whole approach can be applied to any given subsystem, by projecting the observed I/O vector sequence on the I/Os of the considered subsystem. The smaller the subsystem, the smaller the cost of the unobservable inference as well. Distributing the identification is not a problem *per se*, once a partition of the system is provided. However, finding an adequate partition, taking into account the objective of identification, is one. Furthermore, in a blackbox approach, no insight is provided on the structure of the system.

Since partitioning aims at reducing the cost of the second step of the approach, the first step can be run on the whole system. Even more so, given that direct I/O causalities are extracted during this step, and must not be split, the partitioning must occur subsequently. The partition will therefore be

computed between the two steps, whereas in an approach such as [9], the partition is computed before building any model.

The following procedure is proposed to perform distributed identification on a system:

- Run the observable identification on the whole system, resulting in observable PN fragments. Each fragment already corresponds to an individual subsystem, thus a first partition is already provided.
- Use these PN fragments as entry point for a clustering algorithm, to provide an adequate partition of the system
- During the construction of the partition, the unobservable behaviour of the subsystems is regularly computed. At the end of the clustering, both the partition and the complete IPN models are obtained.

IV. Optimal partitioning for distributed identification

A. Formulation of partitioning as an I/O cover problem

Distributed identification consists in running the chosen identification procedure on each of the subsystem; partitioning consists in building those subsystems such that each I/O belongs to at least one (exactly one if overlapping is forbidden), as summarized in Figure 2.

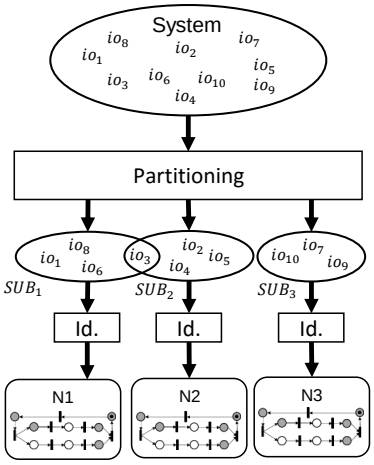


Fig. 2. Principle of the distributed approach, based on a partition of the system.

The choice of the susbsystems depends on the objective of the model, which conditions as well the model class. For instance, in [8], according to the objective of fault diagnosis, the objective function is the minimization of the exceeding language, composed of behaviours possible in the model but not observed. An alternative function was considered to minimize the branching degree in the chosen automata models.

However, for reverse engineering, the objective is instead to build a model expressing compactly complex behaviours, hence the choice of Petri Nets, and to build an understandable, simple net. Two criteria are chosen to express model simplicity: strong connexity and structural complexity. Most other complexity criteria are related to behavioural complexity, which is inherent to the studied system; nevertheless, simple to read PNs can capture complex behaviours. To express the structural complexity of a net, the chosen metric is the Coefficient of Network Complexity [28], defined below:

Definition 1. *Let $G = (P, T, I, O)$ be a PN structure. The Coefficient of Network Complexity is defined as :*

$$CNC_{PN}(G) = \frac{|I| + |O|}{|P| + |T|}$$

Using this metric to qualify identified nets, a first formulation of the partitioning problem for reverse engineering is proposed. Overlapping is allowed, since some I/Os might be located at the interface between two subsystems, making it a cover problem.

Finding a cover *Consider a DES consisting in m I/Os $\{io_1, \ldots, io_m\}$. Compute an I/O-cover of N I/O subsystems $\{SUB_1, \ldots, SUB_N\}$, with the constraints:*

1) $\forall i \in [1, m], \exists SUB_k, io_i \in SUB_k$

2) Each model N_k built on a SUB_k is strongly connected and optimal regarding the two criteria:

a) minimize $CNC_{Avg} = \frac{1}{N} \sum_{k=1}^{N} CNC_{PN}(N_k)$

b) minimize N

The first constraint implies that each I/O belongs to at least one subsystem, and that overlapping is allowed. The second one defines an acceptable solution: each subsystem must be guaranteed to be identified by a strongly connected net. The first criterion aims at minimizing the average structural complexity of the identified nets. However, an optimal solution regarding this criterion consists in building a bijection between the I/Os and the subsystems. Therefore, the second criterion is added to find a compromise between the size of the distributed models and their simplicity, by minimizing the number of subsystems. An optimal solution regarding this second criterion only is a monolothic model, provided it can be computed.

Since the optimization problem is multicriteria, there is no single solution that simultaneously optimizes each objective. Instead, the best solutions form a Pareto frontier. Although Pareto optimal, the extreme solutions (monolithic or bijective) are not interesting; the choice of a solution among the possible ones will be discussed later.

B. Adaptation of the I/O cover problem into a partitioning problem involving the observable behaviour

As shown in Section III-B, the first step builds observable fragments. When transitions and places are connected, input events and conditions labelling the transitions have been identified as causes of the output events associated to the places. I/Os associated to a same observable fragment must therefore be put in the same subsystem. Solitary transitions are labelled with input events (in $\mathbb{D}$) which could not be associated to an output. From now on, a *fragment* designs generically one connected component or one solitary transition, and a *block* designs a non-empty set of fragments. Fragments and blocks are mapped onto the I/Os by the following function:

Definition 2. *Let B_k be a block, and F_i an observable fragment. Each observable place p is associated to an output $\varphi(p)$, and each transition t to the inputs in its firing function $\lambda(t)$. The inputs used in the firing function of a transition t are designed by $\{\lambda(t)\}$. Two mapping functions Map_F and Map_B are defined as follows, mapping the elements onto the I/Os:*

$$Map_F:\ F_i \longrightarrow \bigcup_{p\in F_i}\varphi(p)\bigcup_{t\in F_i}\{\lambda(t)\} \subseteq (\mathbb{Y}\cup\mathbb{U})$$

$$Map_B:\ B_k \longrightarrow \bigcup_{F_i\in B_k} Map_F(F_i) \subseteq (\mathbb{Y}\cup\mathbb{U})$$

φ being bijective, each output is associated to exactly one fragment, whereas inputs can be shared. Additionally, inputs in $\mathbb{D}$ are constrained to be associated to only one block. At least two transitions are always labelled by these input events (rising and falling edge); thus, solitary transitions are no longer alone in their blocks. An elementary set of blocks $\{B_1,\dots,B_m\}$ can be built from the fragments, such that this condition is verified, and the number of fragments per block is minimized. By associating one subsystem to each of the blocks, a first partition is obtained, and the associated I/O mapping is a solution to the cover problem.

Example 1. *For $\mathbb{U}=\{u_1,\dots,u_5\}$ and $\mathbb{Y}=\{Y_1,Y_2,Y_3\}$, consider the observable fragments of Figure 1. Five elementary blocks are built: three are the connected fragments, and the remaining two are the pairs of transitions labelled by $\uparrow u_2$ (resp. u_5) and $\downarrow u_2$ (resp. u_5). The mapping function leads to the following I/O subsystems: $\{u_3,u_4,Y_2\}$,$\{u_1,u_4,Y_1\}$,$\{Y_3\}$,$\{u_2\}$,$\{u_5\}$, which cover the system.*

The cover problem stated on the I/O level in the previous section can now be converted into a new problem stated on blocks. A unicity constraint is added to prevent sharing blocks between subsystems (and avoid place or transition duplications).

Finding a partition *Consider a block set $\{B_1,\dots,B_m\}$. Compute a partition of N subsystems $\{SSYS_1,\dots\}$, with the constraints:*

1) $\forall i\in[1,m], \exists! SSYS_k, B_i\in SSYS_k$
2) *The addition of unobservable behaviour to $SSYS_k$ leads to a strongly connected net N_k*

and optimal regarding the two criteria:

a) *minimize* $CNC_{Avg}=\dfrac{1}{N}\sum_{k=1}^{N}CNC_{PN}(N_k)$

b) *minimize N*

A solution to this problem is directly a modelling solution as distributed identified nets, and a partition of the I/Os is deduced from the subsystems by the mapping function, solving the initial problem. This problem can be viewed as an Exact Set Cover problem [29]. Numerous subsystems can be grown out of the blocks B_i. Then, from all the candidate subsystems, an exact cover, optimal regarding the criteria, can be computed, using for instance Knuth's algorithm [29]. However, this problem is NP-complete.

In the following section, an efficient clustering method is proposed to find 'natural' partitions, and provide a hierarchy of these partitions. A discussion is conducted on the balance between size of the subsystems and the computation time; different variations of the clustering method are proposed depending on the objective of the engineer.

V. A CLUSTERING APPROACH FOR OPTIMAL PARTITIONING

The approach proposed is inspired from hierarchical clustering methods used in data mining [30]. The objective of clustering is to group objects such that objects in a same group (called a cluster) are more similar than objects belonging to different clusters. Similarity is estimated through an appropriate metric, corresponding to a measure of the 'distance' between a pair of objects.

Hierarchical clustering aims not only at grouping objects into clusters, but also at providing a hierarchy: a cluster gathers all clusters below it in the hierarchy. Agglomerative clustering is a bottom-up methodology: each object starts in its own cluster, and pairs of clusters are merged while moving up in the hierarchy.

In our problem, the objects are subsystems $SSYS_i$. Initially, each subsystem is composed of only one block ($SSYS_i=\{B_i\}$). An agglomerative clustering approach is natural to group the blocks, lower the number of subsystems and satisfy the second objective function. To balance with the first objective function, simplicity should be implied in the similarity metric, so that blocks leading to the most simple models are regrouped.

Most clustering approaches, such as the classical *k-means* [30] require to fix the number of clusters *a priori*. *Affinity Propagation (AP)* proceeds otherwise [31]: by letting the objects find by themselves to which other objects they are most similar, clusters emerge naturally. However, applying AP requires that the similarity of two objects is always finite. The second constraint of the problem implies that the union of two blocks must be a strongly connected net; the similarity of two blocks can therefore not be defined if their union does not satisfy the constraint. The approach proposed in this section is inspired by AP, and adapted to our problem.

A. Principle

First, the notion of similarity between subsystems is defined:

Definition 3. *Let $SSYS_i, SSYS_j$ be two subsystems. Let N_{ij} be the complete IPN identified after the addition of unobservable behaviour to the union of the subsystems. The similarity Sim of the two subsystems is:*

$$Sim(SSYS_i,SSYS_j)=\begin{cases} CNC_{PN}(N_{ij}) & \text{if } N_{ij} \text{ is s.c.}\\ \emptyset & \text{otherwise}\end{cases}$$

In the first case, the merged subsystem satisfy the second constraint of the optimization problem (strong connexity). Similarity is then defined and is an indicator of the closeness of subsystems; a low value corresponds to a pair of subsystems whose assembled model is simple to read, hence implying simple operations.

In the other case, the similarity is undefined ($\emptyset$), as the model resulting of the merging is not strongly connected. This similarity factor is not a distance. For instance, given subsystems A,B,C, Sim(A,B) and Sim(B,C) being defined, Sim(A,C) might be undefined, unsatisfying the triangular inequality. However, Sim(A∪B,C) is likely to be defined; adequate subsystems might include highly dissimilar subsystems (A,C) who are both similar to a third one (B).

Whenever two systems are similar, they can be merged, and the resulting net would satisfy the constraints. However, to fulfill the first objective function (low average structural complexity), the idea is to merge only the subsystems who are the most similar, such that structural complexity is minimized at each merging. The affinity of a subsystem is defined as the subsystems it is the most similar to:

Definition 4. *Let $SSYS_1, \ldots, SSYS_m$ be m subsystems. The affinity Aff of a subsystem $SSYS_i$ is the set:*

$$Aff(SSYS_i) = \{SSYS_j | Sim(SSYS_i, SSYS_j) = \min_k(Sim(SSYS_i, SSYS_k))\}$$

The affinity of a subsystem might be the empty set, a singleton, or composed of multiple subsystems. An affinity graph is derived from this definition:

Definition 5. *Let $SSYS_1, \ldots, SSYS_m$ be m subsystems. The affinity graph A=(V,E) is a directed graph, where the m vertices V represent the m subsystems and the edges represent the affinity, i.e.*

$$(N_i, N_j) \in E \Leftrightarrow SSYS_j \in Aff(SSYS_i)$$

Example 2. *Consider 4 subsystems $\{1,2,3,4\}$ such that $Sim(1,2) = Sim(1,3) = Sim(1,4) = \emptyset$, $Sim(2,3) = Sim(2,4) = 1.25$ and $Sim(3,4) = 1.15$. The corresponding affinity graph is presented in Figure 3: 1 is an isolated node, 2 has two successors, and finally 3 and 4 are eachothers unique respective affinity, forming a 2-cycle.*

Subsystems to be merged in priority are the length-2 directed cycles in the affinity net: they involve two subsystems such that each subsystem is the most similar to the other. The subsystems can be iteratively merged, similarity recomputed at each step, until no more merging is possible. The full agglomerative procedure is exposed by Algorithm 1. At each step, the affinity graph is studied; a length-2 directed cycle is picked in each strongly connected component (at least two nodes) of the graph. The nodes of the cycles are merged and the affinity graph recomputed. The procedure is repeated until there are no more edges in the graph. In the worst case, convergence is achieved when there remains exactly one node, which corresponds to the full system.

Algorithm 1 Agglomerative clustering of subsystems

Require: Blocks $B_1, \ldots, B_n$
Ensure: $PAR = \{SSYS_1, \ldots, SSYS_m\}$ a partition.
1: Compute the initial partition $PAR = \{\{B_1\}, \ldots \{B_n\}\}$
2: Compute the affinity graph $A = (V, E)$ related to PAR
3: **while** $E \neq \emptyset$ **do**
4: Pick a length-2 directed cycle $(SSYS_i, SSYS_j)$ in each strongly connected component of A
5: Merge each pair of subsystems into a new one $SSYS_{ij}$
6: Update the partition PAR
7: Update the affinity graph
8: **end while**

Sim	1	2	3	4
1	-	∅	∅	∅
2	∅	-	1,25	1,25
3	∅	1,25	-	1,15
4	∅	1,25	1,15	-

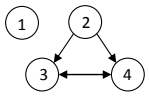


Fig. 3. Similarity table and Affinity graph deduced for Example 2

Example 3 (Example 2 cont.)**.** *The agglomerative clustering is illustrated by Figure 4. From the similarity table of Figure 3, 3-4 is the only strongly connected component. The nodes are merged, and the similarity recomputed. Sim(1,2) is already known, Sim(1,3∪4)=∅, and Sim(2,3∪4)=1.3. 2-3∪4 is a new strongly connected component, and merged. Finally Sim(1,2∪3∪4)=∅, and there is no more edge in the affinity graph, stopping the clustering. Figure 4(b) shows a hierarchical representation. Each layer is a solution of the problem, and exhibits a different number of subsystems.*

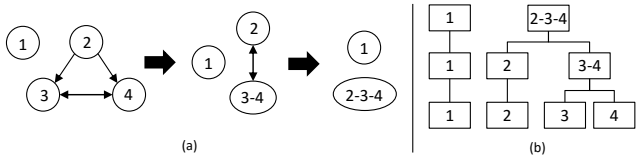


Fig. 4. (a) Evolution of the affinity graph along the clustering; (b) Hierarchical representation

The costliest operation is the computation of the affinity graph (lines 2,7), which requires the evaluation of all similarity values. An upper bound of the number of similarity values to compute during the discovery is given by the following proposition:

Proposition 1. *Consider a system with n subsystems. To run Algorithm 1, the maximal number of similarity values to compute is $(n-1)^2$.*

Proof. Given the n initial subsystems, there are initially $n(n-1)/2$ similarity values to compute to build the first affinity graph. Then, the worst case is the following: at each step, only two nodes of the graph are merged. After the first loop, $n-2$ subsystems are unchanged, and one is new, hence $n-2$ new similarity values to compute. After the second loop, it remains $n-3$ subsystems are unchanged, hence $n-3$ new values, etc. The total number is therefore:

$$\frac{n(n-1)}{2} + (n-2) + (n-3) + \cdots + 1 = \frac{n(n-1)}{2} + \frac{(n-1)(n-2)}{2} = (n-1)^2$$

□

The main advantage of the approach is to compute a full hierarchy. Suppose that the expert decides a solution with N' subsystems is not distributed enough: it suffices to go down in the hierarchy to find an already computed solution with more subsystems. Reversely, if there are too many subsystems, it suffices to go up in the hierarchy to find a coarser solution.

However, the computation of the full hierarchy is expensive. First, if there exists a monolithic, strongly connected model of the full system, then Algorithm 1 does not stop until said model is reached, whereas the whole point of the distributed approach is to avoid computing the monolithic model. Furthermore, the update of the affinity graph (line 7 of Algorithm 1) implies to compute similarity values, which requires unobservable behaviour inference and is computationally expensive. Possible limitations are introduced to ensure the efficiency of the clustering.

B. Limited clustering

To limit clustering, threshold rules are introduced to decide quickly if a similarity value is worth computing, or call it undefined ($\emptyset$). By increasing the number of undefined similarities, the number of non-empty affinities drops, and convergence is achieved before reaching the monolithic model.

A first proposition, named $|T|$-*clustering*, consists in limiting the number of transitions in a given model. If the sum of the transitions of two subsystems is over a given threshold, they are then considered dissimilar, without computing the similarity. The value of this threshold can be set arbitrarily. However, it can also be decided depending on the number of subsystems to aim for. Given that the initial number of subsystems n is known, the aim can be $n/2$ subsystems, *i.e.* a threshold $|T|_{Lim} = \lfloor 2|T|/n \rfloor$. To get more subsystems, an already computed solution can be picked. To get less subsystems, the aim can be $n/4$ subsystems ($|T|_{Lim} = \lfloor 4|T|/n \rfloor$), and the computation can be continued from the last solution; the procedure is repeated until a satisfying solution is reached. The main advantage of this approach is that the number of subsystems can be controlled. However, some elementary blocks might contain too many transitions, such that their similarity values remain undefined, missing potentially simple models. Furthermore, the computation time is not controlled, and some similarity values can be unpredictabily expensive to compute, despite a reasonable number of transitions.

A second proposition, named *time-clustering*, consists in limiting the allowed computation time of a similarity value by setting a threshold t_{lim}. A total computation time t can be arbitrarily fixed; by using Proposition 1, given n initial subsystems, the threshold can be set at $t_{lim} = t/(n-1)^2$. This is a lower threshold, ensuring that the total computation time does not exceed t; the actual value of the computation time should be far lower. Although highly computer-dependent, this approach guarantees to obtain a solution quickly. Furthermore, all subsystems are sollicited and can be merged, compared to the $|T|$-threshold. However, there is no explicit link between the computation time and the number of subsystems reached when the algorithm terminates. If the final number of subsystems is too high, the algorithm can be restarted by increasing the threshold, but it is impossible to predict the variation.

Choosing a limited clustering approach: The two approaches are complementary, as the advantages of one are the drawbacks of the other. In both cases, if the granularity of the system is too coarse (too few subsystems), previous, finer solutions have already been calculated and are available in the hierarchy.

Time-clustering provides naturally a solution within the timespan allowed by the designer, and without any additional information. It is recommended to obtain quickly a model to get insight on the system. However, the designer can also choose to pause the identification before the clustering, and look at the initial subsystems. $|T|$-clustering can then be chosen, while fixing an arbitrary limit of $|T|$ per subsystem, and should provide especially good partitionings if the initial subsystems have roughly the same number of transitions.

Both approaches are illustrated on the same benchmark in the next section.

VI. Application

A. Presentation of the benchmark

The Mechatronics Standard System (MSS) is a real-world laboratory manufacturing system developed by Bosch, available on the experimental platform of the LURPA (ENS Cachan, France).

The purpose of this system is to sort workpieces according to material and presence of a bearing. Workpieces of plastic, brass and steel are treated. The system is decomposed into 4 stations, displayed in Figure 5, and consists in 43 sensors and 30 actuators (hence 73 I/Os). Even though each workpiece is sequentially treated by each station, many workpieces are treated simultaneously in the whole chain, namely during a continuous production phase. This chain therefore exhibits massive concurrency, and the behaviours of the different stations are often interleaved. Also, the chain is filled with shared ressources, such as the chariot of the third station bieng sollicited by the two grippers and the two presses. Data was collected during the observation of 20 production cycles of 24 bearings each, leading to a sequence of length 63.797 vectors. All computations were ran on a laptop (Intel Core i5-3380M CPU @ 2.90GHz x4, 8Go RAM). More information on the procedure used to collect data can be found in [32].

Monolithic identification was performed on this system using the algorithms of [7] for the observable part and [20] for the unobservable one. A strongly connected model is achieved after 336 hours (14 days) of computation; its complexity is $CNC = 2.31$, with 101 transitions, 202 places and 699 edges. This result is both too costly to compute, and too complex to understand, thus distributed identification is considered.

B. Results of distributed identification

First, the observable behaviour is computed on the whole system (73 I/Os). The resulting model consists in 101 transitions and 30 observable places, grouped in 13 connected

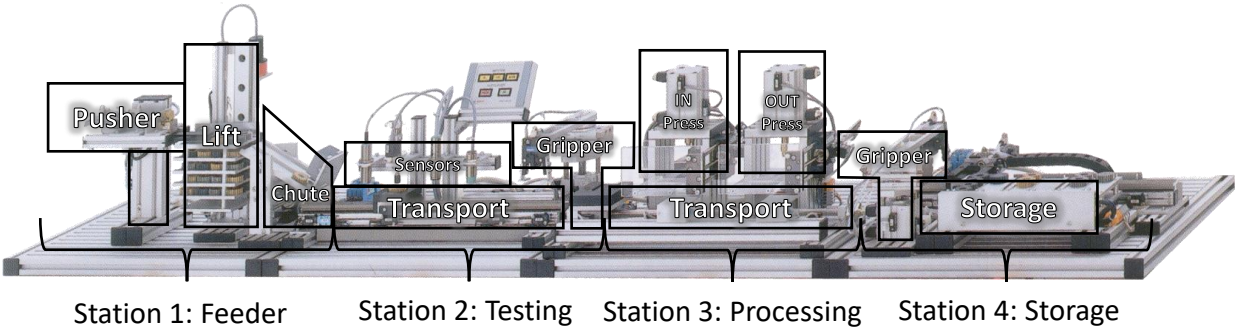
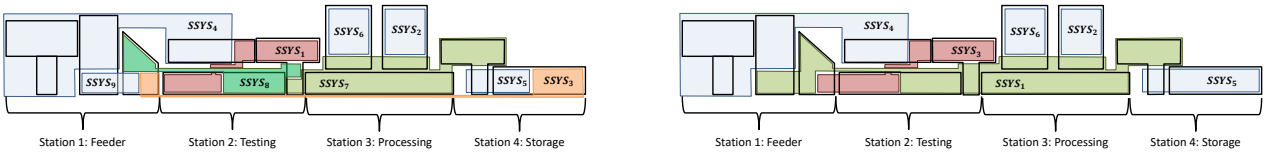


Fig. 5. The MSS, with its 4 stations and a functional decomposition



(a) (b)

Fig. 6. (a): Location of the nine computed subsystems on the MSS from $|T|$-clustering, $|T|_{Lim} = 9$. (b): Location of the six computed subsystems on the MSS from time-clustering, $t_{Lim} = 20s$

fragments and 24 isolated transitions (not shown here). Following Section IV, these fragments are regrouped into $n = 21$ elementary blocks of various size (from 2 to 28 transitions, avg 4,8, $\sigma = 6.97$). The first partitioning consists therefore in 21 subsystems.

The two limited clustering approaches have been considered:

- $|T|$-clustering. The proposed aim is to halve the number of subsystems, therefore $|T|_{Lim} = \lfloor 2|T|/n \rfloor = 9$
- time-clustering. The total computation time has been arbitrarily limited to 2 hours, therefore $t_{Lim} = 7200/400 =$ 18s, rounded up to 20s.

The clustering is then performed by Algorithm 1, and the unobservable behaviour is computed on each subsystem during the clustering. Both the partition and the distributed models are obtained when Algorithm 1 ends. For $|T|$-clustering, similarity of subsystems whose number of transitions exceeds the threshold are not computed, while for time-clustering, the computation of a similarity value is halted when the time threshold is exceeded. A solution with 9 subsystems and $CNC_{Avg} = 1.35$ is reached in 25 minutes with $|T|$-clustering, whereas a solution with 6 subsystems and $CNC_{Avg} = 1.38$ is reached in 53 minutes with time-clustering. The caracteristics of the identified subsystem models are presented in Table I. Subsystems 2, 4 and 6 are identical in both solutions. The $|T|$-solution verifies the threshold except for subsystems 4 and 7: these initial blocks are disproportionate in size and could not grow further. However $SSYS_1$ of the time-solution is a huge ($|T| = 43$) subsystem whose model can be computed in less than 20s; it is exactly the union of subsystems 7,8,9 of the $|T|$-solution, who could not be merged due to the threshold.

The subsystems location on the MSS are shown in Figure 6. On one hand, in the time-clustering solution, interestingly, $SSYS_1$, the biggest subsystem, involves all stations, and

$\|T\|_{Lim} = 9$									
Subsystem	1	2	3	4	5	6	7	8	9
$\|T\|$	8	6	8	22	6	8	28	8	7
CNC_{PN}	1.2	1.1	1.3	1.9	1.4	1.1	1.9	1.4	1.2

$t_{Lim} = 20s$							
Subsystem	1	2	3	4	5	6	
$\|T\|$	43	6	12	22	10	8	
CNC_{PN}	1.7	1.1	1.4	1.9	1.2	1.1	

TABLE I
COMPARISON OF THE SOLUTIONS OBTAINED FOR $|T|_{Lim} = 9$ AND $t_{Lim} = 20s$

represents the operations every gear undergoes through the chain. The remaining subsystems are satellites at the service of the main process (for instance, $SSYS_2$ is a press whose operation depends on the material, and $SSYS_3$ consists in subsystems who often idle while waiting for the chariot of the main process). Besides providing a solution adequate to the objective of reverse-engineering, the automated partitioning provided additional insight on the behaviour on the chain, which could hardly be thought when expertly designing. On the other hand, the $|T|$-solution provides similarly located, though often smaller, subsystems. However, a subsystem such as $SSYS_3$ is instead a gathering of irrelated components from both stations 1 and 4. The threshold prevented the algorithm from grouping the station 1 part with $SSYS_1$ and the station 4 part with $SSYS_5$, as was done in the time-solution.

Finally, the procedure was run multiple times for various thresholds. A few solutions obtained when Algorithm 1 terminates are plotted in Figure 8. The extreme solutions correspond to the monolithic model, and to the elementary blocks partition.

As a result, the storage unit of station 4 is modelled differently, as shown by the IPNs of Figure 7. These models are

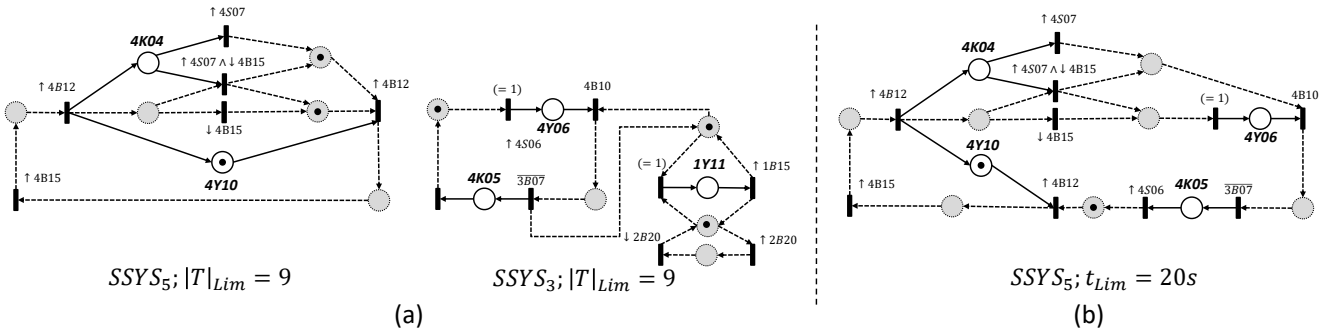


Fig. 7. Identified models of the storage unit: (a) Discovered as two subsystems ($SSYS_5$ and $SSYS_3$) with $|T|$-clustering; (b) Discovered as one ($SSYS_5$) with time-clustering

built during the clustering approach. Inputs 3B07,4S06,4B10 and outputs 4Y06, 4K05 are affected to subsystem $SSYS_3$ in $|T|$-clustering, whereas they are affected to $SSYS_5$ in time-clustering. The storage unit is represented in two subsystems ($SSYS_3$ and $SSYS_5$) in the $|T|$-solution, whereas only in one in the time-solution. Notice however that the same behaviours have been discovered in both cases, although only merged in the latter case.

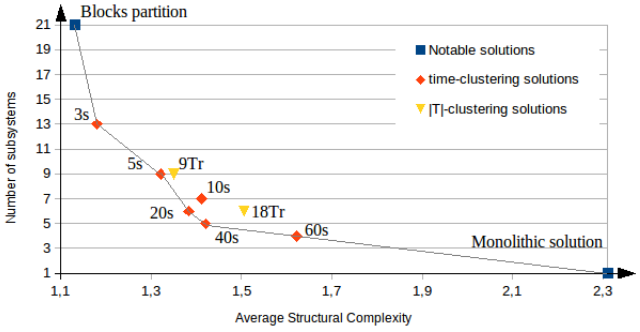


Fig. 8. Evaluation of different partitions computed with the clustering approach

$|T|$-clustering is less efficient here, as the elementary blocks are disproportionate in size (high standard deviation). Therefore, the biggest blocks can not be merged with others, despite the possibility of reaching simple models. Excepted the extreme points, the Pareto shape consists only of time-clustering solutions. Notice the lack of monotonicity, as the partition obtained for $t_{lim} = 10s$ is strictly worse than the one obtained for $20s$.

VII. Conclusion

An automatic partitioning approach was proposed to compute distributed identified model. The problem has been set first as an optimization problem to find an adequate cover of the I/Os, such that the resulting distributed models are simple to understand. Using observable fragments computed by the first step of the identification procedure, it was reformulated into finding a partition of the fragments, with the same objective, the I/O cover ensuing from the partition. An algorithm inspired from clustering methods is proposed to agglomerate the fragments into clusters, and thresholds are introduced to limit computation time. This approach was efficiently applied to a benchmark. Further work should focus on the inclusion of additional knowledge in the partitioning (greybox approach), and the variation of objective functions for different model purposes.

References

[1] P. Ribot, Y. Pencolé, and M. Combacau, "Design requirements for the diagnosability of distributed discrete event systems," *Proc. of the 19th International Workshop on Proinciples of Diagnosis, Blue Mountains, New South Wales, Australia*, pp. 347–354, 2008.

[2] S. Genc and S. Lafortune, "Distributed diagnosis of place-bordered petri nets," *IEEE Transactions on Automation Science and Engineering, 4(2)*, pp. 206–219, 2007.

[3] C. Girault and R. Valk, *Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications*. Springer-Verlag, 2003.

[4] M.-P. Cabasino, P. Darondeau, M.-P. Fanti, and C. Seatzu, "Model identification and synthesis of discrete-event systems," in *Contemporary Issues in System Science and Engineering*. IEEE-Wiley, 2015.

[5] J. Ladiges, C. Haubeck, A. Fay, and W. Lamersdorf, "Learning behaviour models of discrete event production systems from observing input/output signals," *Proc. of the 15th IFAC/IEEE/IFIP/IFORS Symposium on Information Control Problems in Manufacturing (INCOM)*, pp. 1565–1572, 2015.

[6] D. Munoz, A. Correcher, E. Garcia, and F. Morant, "Identification of stochastic timed discrete event systems with st-ipn," *Mathematical Problems in Engineering, vol. 2014, Article ID 835312, 21 pages*, 2014.

[7] A.-P. Estrada-Vargas, J.-J. Lesage, and E. Lopez-Mellado, "A black-box identification method for automated discrete-event systems," *IEEE Transactions on Automation Science and Engineering*, pp. 1–16, 2015.

[8] M. Roth, J.-J. Lesage, and L. Litz, "Black-box identification of discrete event systems with optimal partitioning of concurrent subsystems," *Proc. of the 2010 American Control Conference (ACC), Baltimore, MD, USA*, pp. 2601–2606, 2010.

[9] S. Schneider and L. Litz, "Automatic partitioning of des models for distributed fault diagnosis purposes," *Proc. of the 12th International Workshop on Discrete Event Systems-WODES 2014, Paris, France*, pp. 21–26, 2014.

[10] V. Kudryavtsev, I. Grunskii, and V. Kozlovskii, "Analysis and synthesis of abstract automata," *Journal of Mathematical Sciences, 169(4)*, pp. 481–532, 2010.

[11] E. Badouel, L. Bernardinello, and P. Darondeau, "Polynomial algorithms for the synthesis of bounded nets," *Lecture Notes in Computer Science, 915*, pp. 647–679, 1995.

[12] A. Giua and C. Seatzu, "Identification of free-labeled petri nets via integer programming," *Proc. of the 44th IEEE Conference on Decision and Control, and the European Control Conference*, pp. 7639–7644, 2005.

[13] M. Cabasino, A. Giua, C. Hadjicostis, and C. Seatzu, "Fault model identification and synthesis in petri nets," *Discrete Event Dynamic Systems, 25(3)*, pp. 419–440, 2015.

[14] F. Basile, P. Chiacchio, and J. Coppola, "Faulty model identification in deterministic labeled time petri nets," *Proc. of the 13th Workshop on Discrete Event Systems-WODES 2016, Xi'an, China*, pp. 486–492, 2016.
[15] S. Klein, "Identification of discrete event systems for fault detection purposes," Ph.D. dissertation, Ecole Normale Supérieure de Cachan, 2005.
[16] M. Meda-Campana and E. Lopez-Mellado, "A passive method for on-line identification of discrete event systems," *Proc. of the 40th IEEE Conference on Decision and Control, Orlando, Florida USA*, pp. 4990–4995, 2001.
[17] ——, "Identification of concurrent discrete event systems using petri nets," in *Proceedings of the 17th IMACS World Congress on Computational and Applied Mathematics*, 2005.
[18] A.-P. Estrada-Vargas, J.-J. Lesage, and E. Lopez-Mellado, "Input-output identification of controlled discrete manufacturing systems," *International Journal of System Science 45(3)*, pp. 456–471, 2014.
[19] T. Tapia-Flores, E. Lopez-Mellado, A.-P. Estrada-Vargas, and J.-J. Lesage, "Petri net discovery of discrete event processes by computing t-invariants," *Proc. of the 2014 IEEE 19th Conference on Emerging Technologies and Factory Automation (ETFA), paper 345, 8 pages*, 2014.
[20] J. Saives, G. Faraut, and J.-J. Lesage, "Identification of discrete event systems unobservable behaviour by petri nets using language projections," *Proc. of the 2015 European Control Conference (ECC), Linz, Austria*, pp. 464–471, 2015.
[21] W.-M.-P. Van der Aalst, *Process Mining, Discovery, Conformance and Enhancement of Business Processes*. Springer-Verlag Berlin Heidelberg, 2011.
[22] B. Van Dongen, A. Alves de Medeiros, and L. Wen, "Process mining: Overview and outlook of petri net discovery algorithms," *ToPNoC II, LNCS 5460*, pp. 225–242, 2009.
[23] T.Tapia-Flores, E. Rodriguez-Perez, and E. Lopez-Mellado, "Discovering process models from incomplete event logs using conjoint occurrence classes," *37th International Conference on Application and Theory of Petri Nets and Concurrency Petri Nets 2016 and 16th International Conference on Application of Concurrency to System Design ACSD 2016, Torun, Poland*, pp. 31–46, 2016.
[24] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser, "Process mining based on regions of languages," *Business Process Management*, pp. 375–383, 2007.
[25] W.-M.-P. Van der Aalst, "A general divide and conquer approach for process mining," *2013 Federated Conference on Computer Science and Information Systems (FedCSIS), Krakow, Poland*, pp. 1–10, 2013.
[26] J. Carmona, J. Cortadella, and M. Kishinevsky, "Divide-and-conquer strategies for process mining," *Proc. of the 7th International Conference, BPM 2009, Ulm, Germany, LNCS 5701*, pp. 327–343, 2009.
[27] R. David and H. Alla, "Petri nets for modeling of dynamic systems- a survey," *Automatica Vol.30 No.2*, pp. 175–202, 1994.
[28] T. L. Pascoe, "Allocation of resources cpm," *Revue Francaise de Recherche Oprationelle 38*, pp. 31–38, 1966.
[29] D. Knuth, "Dancing links," *Millenial Perspectives in Computer Science*, pp. 187–214, 2000.
[30] L. Rokach, "A survey of clustering algorithms," *Data Mining and Knowledge Discovery Handbook*, pp. 269–298, 2010.
[31] B. Frey and D. Dueck, "Clustering by passing messages between data points," *Science 315*, pp. 972–976, 2007.
[32] M. Roth, J.-J. Lesage, and L. Litz, "Identification of discrete event systems - implementation issues and model completeness," in *Proceedings of the 7th International Conference on Informatics in Control, Automation and Robotics (ICINCO) Funchal, Portugal*, 2010.

Jeremie Saives recieved the M.Sc. degree in complex systems engineering in 2013, and the Ph.D. degree in automatic control in 2016 from the Ecole Normale Superieure Paris-Saclay, France.

His research interests include behavioural identification of Discrete Event Systems from sensors and activators data, and activity discovery in sensor-equipped smart environments.

Gregory Faraut recieved the M.Sc. degree in electronic, electrotechnic and automatics from the University of Nice-Sophia Antipolis, Nice, France, in 2006, and the Ph.D. degree in automatic control from the Ampere Laboratory, INSA Lyon, Villeurbanne Cedex, France, in 2010.

Since 2011, he has been an Associate Professor of Automatic Control at LURPA, ENS Cachan, France. His research interests concern the field of formal methods and models of discrete event systems. Applications focus on identification and ambient assisted living.

Jean-Jacques Lesage received the Ph.D. degree from the Ecole Centrale de Paris and the Habilitation diriger des recherches from the University Nancy 1 in 1989 and 1994 respectively. He is currently Professor of Automatic Control at the Ecole Normale Suprieure de Cachan, France, where he was head of the Automated Production Research Laboratory during eight years.

His research interests are in the field of formal methods and models for synthesis, analysis and diagnosis of Discrete Event Systems (DES), with applications to manufacturing systems, network automated systems, energy production, and more recently to ambient assisted living.