

# Prognosis of multiple instances in time-aware declarative business process models



Diana Borrego\*, María Teresa Gómez-López, Rafael M. Gasca

Department of Languages and Computer Systems, Universidad de Sevilla, Escuela Técnica Superior de Ingeniería Informática, Dpto. Lenguajes y Sistemas Informáticos, Av Reina Mercedes s/n, 41012 Sevilla, Spain

## ARTICLE INFO

### Article history:

Received 27 September 2019  
Received in revised form 7 April 2020  
Accepted 7 April 2020  
Available online 18 May 2020

### Keywords:

Declarative business processes  
Multiple instances  
Model-based prognosis  
Robustness

## ABSTRACT

Technological evolution, heading for industry 4.0, makes companies tend to automate their management and operation, ideally defining it through business process models. To describe policies or rules related to the execution order of the activities in an organization, Declarative Business Process Models permit a relaxed description of activity order, which needs monitoring to detect non-conforming behaviors. Commonly, the detection of a violation implies that the malfunction has already occurred, being better to avoid the violation in advance. To predict future violations, prognosis is required.

To allow the modeling of real business behavior, an extension of declarative business process models including both time patterns and multiple instances is proposed. This new model can be used to prognosticate if current process instances may violate a defined model in the future, according to the analysis of the robustness of the process instances evolution. The proposed Model-Based Prognosis is based on analyzing the event traces that represent the current instances and propagate their possible progression through the Constraint Programming paradigm. To ascertain if the model could be violated, it is analyzed how its robustness can tackle unexpected behaviors.

To complete the formalization and modeling, an implementation applied to a real medical example is included in the paper. The prognosis of concurrent instances is addressed, dealing with formalized time and activity patterns even considering the resource availability, and getting acceptable execution times.

The automatic verification and prognosis of declarative business processes are addressed considering concurrency and synchronization of multiple instances, performing well in terms of execution time.

© 2020 Elsevier B.V. All rights reserved.

## 1. Introduction

The continuous evolution of companies towards Industry 4.0 entails the need for processes to automate their management and operation tasks, for example, by defining their behavior through Business Process Models. Among those tasks, the automatic detection of malfunctions should be a major concern, since this can greatly facilitate the resolution of possible errors, lowering costs. The increasing complexity can provoke malfunctions that must be diagnosed and prognosticated [1].

Compliance analysis ascertains whether the business process is working as modeled or not according to the KPIs defined by the company [2]. Unfortunately, this detection is executed after a fault has been produced. The idea presented in this paper is not limited to

continuous observation and diagnosis of past compliance violations (reactive management) but also includes the skill to predict possible future violations (pro-active management) at run-time, through monitoring a potentially large number of process instances.

What violation is detected depends on the elements monitored during process execution. Business process compliance can involve the different dimensions of a process model (i.e. control flow, data flow, resources or time). Business processes cannot always be described imperatively and implemented by commercial Business Process Management Systems (BPMSs), especially when there is a high human interaction. An example of this type of scenario is the medical environment, where medical guidelines describe the correct and incorrect order among actions. Declarative models describe what is permitted and prohibited instead of a strict description of the order of the activities. When the executions of the activities are not guided by software that establishes what activity is executed in each moment, the activities can be executed violating the business rules defined by the experts. It is even more complex when the guidelines involve several instances that can

\* Corresponding author.

E-mail addresses: [dianabn@us.es](mailto:dianabn@us.es) (D. Borrego), [maytegonmez@us.es](mailto:maytegonmez@us.es) (M.T. Gómez-López), [gasca@us.es](mailto:gasca@us.es) (R.M. Gasca).

be executed at the same time and have shared resources. For this reason, our proposal is centered on the modeling of declarative business processes that include a time perspective and multiple instances. Monitoring the event traces that include the timestamps of executed activities, we can detect a declarative model violation and even forecast a violation to avoid in the future. A previous approach was analyzed in [3], where declarative models supported time perspective to prognosticate an error in the future. However, the previous paper did not include the problem of how multiple instances can affect the shared resources and the prognosis in the future according to how instances can evolve.

### 1.1. Problem statement

Usually, regulations, laws, or guidelines (often available as textual descriptions) give rise to compliance requirements in business processes, such as ‘an anesthesia test must be done 4 days before an operation’. This rule will be violated if, on the day of the operation, the anesthesia test has not been done. But if the test needs three days to be analyzed in the laboratory, we can detect the violation even two days before the operation, although it will be too late to avoid the malfunction. The prognosis tries to detect four days before the operation that the violation is about to happen, but it still can be avoided.

There is another problem that makes the prognosis more complex since frequently the time that we need to execute an activity (three days for anesthesia test in the example) is affected by the number of process instances that are executed at the same time. For example, a laboratory can analyze 1000 tests per day, then the time the process needs is related to the number of instances running at the same time. In this context, we propose to perform the prognosis of declarative business processes to detect that a misbehavior is close to happening according to the execution of other instances, and considering we still have time to avoid the non-conformity. Likewise, determining how close an execution is to a possible violation (e.g., using robustness) is gaining relevance. Therefore, we are interested in the compliance objectives which are related to time, specified as compliance rules or constraints limiting, for example, the exact time of execution of the activities or even time distances between them. These compliance rules regarding time will be verified over the past events of all running instances, predicting future behaviors.

To analyze the current process state and how it can evolve in the future, it becomes necessary to get past events from an event log, which may be extracted from a variety of data sources, e.g., databases, flat files, message logs, transaction logs, ERP systems, and document management systems. From the information stored for those past events, temporal data is taken into account, so that the prognosis about the future attainment of running instances on time is performed.

Likewise, to get an accurate and realistic prognosis, process instances cannot be assumed to be independent of each other [4]. Moreover, the dependencies among activities can be defined in terms of synchronization of instances to execute the same activity in batch for a set of instances [5]. Batch activities permit, for example, to execute the same activity when at least 10 instances reached it, or how multiple resources permit the execution of the activities in parallel. Then, different types of activities regarding synchronization and/or execution coordination require specific treatment. For example:

- The execution of activities requiring resources is limited to the corresponding availability. This may bring process instances to wait for resources to be able to continue their progress.

- Activities whose execution is limited to certain dates, either periodically or in particular fixed dates.
- Activities whose execution is ruled with batch activation may lead to groups of process instances waiting for their synchronization.

These more complex models can be compared to the current instances to avoid malfunctions. But to detect that a malfunction is about to happen when it is still solvable, we propose the analysis of the robustness of the system. Robustness is the capacity of a system to operate correctly even if an unexpected situation occurs. Thus, the prognosis of business processes and the determination of their robustness should be performed on account of all previous ideas.

### 1.2. Contribution

In this paper, we consider the execution of multiple instances simultaneously. Since the execution of some activities may require, for example, the use of available resources or even the synchronization of different instances, the fulfillment of the time restrictions of a process instance may depend on the timing of other process instances.

In this paper, we address the prognosis of the compliance of declarative business processes, determining their robustness, making several contributions:

- Consideration of the simultaneous execution of multiple instances. To deal with it, certain activity patterns are identified and formalized, so that the coordination and synchronization of different cases are performed, even taking into account the resource availability. We identify various types of activities whose execution patterns are different depending on resources, dates, number of cases, etc. So, they are not independent for each process instance. Those activity patterns are analyzed and formalized.
- Description of Model-Based Prognosis, including the description of declarative business process models by compliance rules [6] with an extension of Declare [7], and the definition of the observational model coming from the event log.

Likewise, and to get a more accurate and realistic prognosis, some time restrictions defined in business policies may be changed getting more constrained ones. This way, they still fulfill policies but provide us with a major capability to properly determine the process robustness. This is due to predictions made at design time may be affected by the number of running instances and/or the available resources, being necessary to adjust them according to problematic scenarios.

- Carrying out of the prognosis to determine a potential non-compliance of business rules in the future. To this end, the process model and event traces are formulated through the Constraint Programming paradigm to predict violations.
- Management of the robustness of the running instances. For that purpose, the cases in the event log corresponding to current instances are analyzed to determine how sensitive they are to small changes related to possible delays in the execution of activities. The concept of robustness used in this paper is oriented towards the capacity of a system to be adapted to external situations, maintaining its correctness regarding the model.

The paper is structured as follows: Section 2 tackles the modeling of the problem, defining necessary concepts and identifying time and activity patterns; Section 3 presents a real multi instance example; Section 4 explains how Constraint Programming paradigm can be used to perform the proposed verification and prognosis, carried out in detail in Section 5; Section 6 deals with the evaluation; Section 7 gives backgrounds on related areas; and

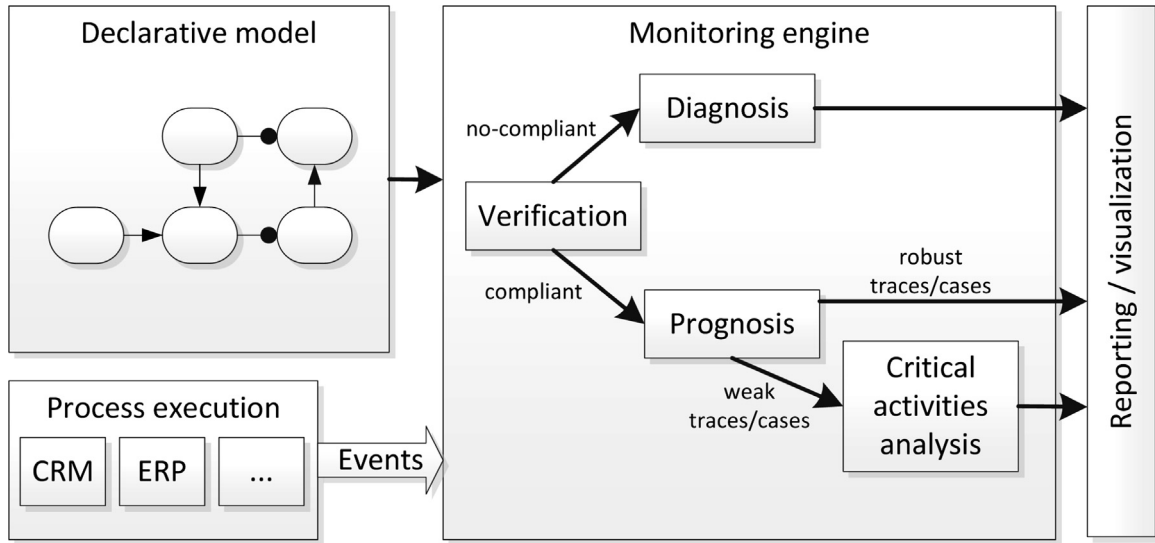


Fig. 1. Compliance monitoring architecture.

finally, conclusions are drawn and future work is proposed in Section 8.

## 2. Formalization of the time patterns in declarative models for multiple instances

As mentioned, to prognosticate a system it is necessary to compare the model with the observations. Centering the problem into declarative models that include descriptions about multiple instances and time restrictions, we aim to monitor multiple instances compliance, identify future inconsistencies and manage the robustness of current cases. To this end, we propose an architecture that combines the modules for modeling, monitoring and prognosticating to detect violations in advance and analyze robustness.

The proposed Compliance Monitoring Architecture is depicted in Fig. 1. It consists of four modules: (1) in the Declarative model module, the business model is defined, getting a final specification as a work-flow and its compliance rules; (2) these model and rules are verified over events coming of the execution of cases in the Process Execution module; (3) the verification and the corresponding diagnosis and/or prognosis are performed in the Monitoring engine module, aim of this paper, which is detailed below; and finally (4) the results from that monitoring are shown to users through the Reporting/Visualization module, and may be different depending on the monitoring that was carried out.

Regarding the *Monitoring engine module*, it receives two types of data: declarative models stating compliance objectives, and events (from an event log) of past and current process instances that should accomplish the model. Usually, events can have attributes, being *activity*, *time*, *costs*, and *resource* typical examples of attribute names [8]. From this information, which is usually presented in an event log, and to perform the prognosis proposed in this paper, we are interested in the information about *activity* and *time*, since it does not affect, for example, what resources were used in the past, but we will only consider their possible lack of availability in the future.

To perform the prognosis of declarative models, our approach is based on Model-based diagnosis. It is used to ascertain whether the behavior of a system is correct or not, and to find out the responsible in case of malfunction. This identification is carried out by comparing the expected behavior (the model) with the observed behavior (the observational model).

### 2.1. Observational model formalization

In this case, the information in the event log conforms to the *Observational Model* of the system. Then an observational model is described by that sequence of events, ordered by the time when they were thrown. As the proposed prognosis is based on the correct order execution of a set of activities, according to a declarative model, some definitions about event, trace, and log are included below.

**Definition 1 (Event).** Let  $A$  be the set of activities of a process model, an event is described by means of a tuple  $\langle tr, a, t, r \rangle$ , where: trace (Definition 2) ( $tr$ ) represents the identifier or the process instance that traces their executions that facilitates to distinguish an instantiation from the others;  $a \in A$  is the activity executed in a particular instant of time  $t$ ; and, in case  $a$  needs some resource to be executed,  $r$  is the used resources, empty otherwise.

**Definition 2 (Trace).** A sequence of events  $[e_1, e_2, \dots, e_n]$  is a trace, if  $\forall e_i, e_j$ , with  $1 \leq i < j \leq n$ ,  $e_i.tr = e_j.tr$  and  $e_i.t < e_j.t$ .

**Definition 3 (Log).** An event log  $L = \{l_1, \dots, l_m\}$  is a set of traces  $l_i$ .

Continuous monitoring of the events in a system implies that those events can conform partial or full traces. Each trace of events (partial or full) can be compliant with the declarative model or not. A compliant partial trace can be *robust* or *weak*. To clarify these terms, we use the definitions introduced in [3].

**Definition 4 (Full Trace (FT) and Partial Trace (PT)).** Being  $T$  a trace and  $Ex(T)$  the set of non-optional activities executed in  $T$ ,  $T$  is a Full Trace if there is no other compliant trace  $R$  that meets  $Ex(T) \subset Ex(R)$ , and a Partial Trace otherwise.

**Definition 5 (Promising (PrT), Robust (RT) and Weak Trace (WT)).** Being  $ST$  the set of concurrent running traces of a declarative BP model, and being  $T$  a Partial Trace in  $ST$ ,  $T$  is a:

- **Promising Trace:** If  $T$  is compliant with the model.
- **Robust Trace:** If  $T$  is promising and, besides, there is more than one possible option (regarding time) to execute all the necessary activities to get  $T$  to be FT. These options are affected and limited due to the concurrent execution of the traces in  $ST$ .
- **Weak Trace:** If  $T$  is promising but not robust.



Fig. 2. Single activity pattern.

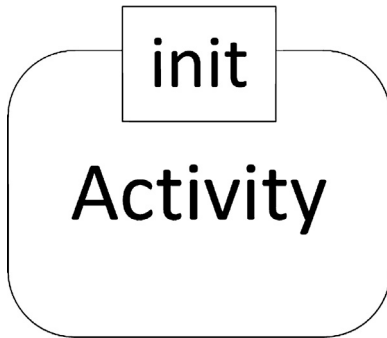


Fig. 3. Initiation activity pattern.

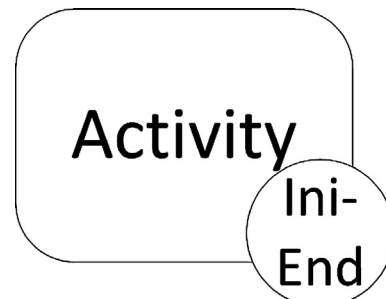


Fig. 4. Interval activity pattern.

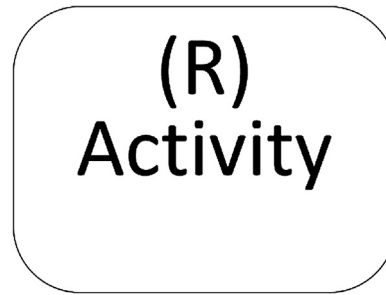


Fig. 5. Resource activity pattern.



Fig. 6. Date activity pattern.



Fig. 7. Batch activity pattern.

## 2.2. Model formalization

The model describes the expected and correct behavior designed by business experts. As commented, in contexts where a high human interaction exists, declarative models are more suitable, since models can be described as a set of rules or guidelines permitted or prohibited to achieve the organizational goal. Different languages have been defined to describe declarative models, as analyzed in [6].

The proposed model is based on Mobucon with new types of activities and time patterns extension of ConDec [9], formed of activities and order relations between them.

### 2.2.1. Activity patterns

As mentioned, and to model declarative models considering multiple instances and address the prognosis, different types of activities are identified regarding the way they affect the concurrent execution of instances. Therefore, this subsection aims to depict those activity patterns, so that a solution for each one of them is proposed in the following sections.

Activities patterns:

- Single activity: activity that can be executed at any time by any number of instances, without limitations (Fig. 2).
- Initiation Activity: first activity to be executed by every process instance (Fig. 3).
- Interval Activity: the activity should be executed between the moment *Ini* and *End*, measured from the execution time of the initial activity (Fig. 4).
- Activity depending on resource perspective: activity whose execution is limited by the available resources, which can be classified as human resources, technological resources or data resources (Fig. 5).
- Activity executed on a date: activity whose execution is performed one day, either in a concrete date or regularly (i.e. after some periodical meeting) (Fig. 6).
- Batch activity: activity clustering a set of active activity instances together and synchronizing their execution according to predefined rules [5] (Fig. 7). Two types of use of case are differentiated: (1) achieving an increased process performance (the activity is executed when a certain number of cases are synchronized, to save setup costs); and (2) comparing business cases (the synchronization is needed to compare cases according to specific criteria). Likewise, a batch activity can be provided to multiple available resources, being possible to run several batches in parallel when they are needed.

Also, any combination of patterns is allowed. For instance, an activity executed on a date may also be limited by resources.

### 2.2.2. Order and time relation between activities

Likewise, as it was aforementioned, declarative models let to describe the order relation between activities:

- **Precedence:** If activity  $A$  precedes  $B$ , then to execute  $B$ ,  $A$  must have been executed before.
- **Response:** Activity  $B$  is a response of activity  $A$  if, whenever  $A$  is executed, then  $B$  must be executed afterwards.
- **Succession:**  $A$  and  $B$  are in succession relation if  $A$  precedes  $B$  and  $B$  is a response to  $A$ .
- **Distance:** Time distances between activities (i.e. minimum time to wait between the execution of two activities) may be affected by the number of available resources and/or running instances. These distances were defined as business policies, which are rules and regulations to define the limits within which decisions must be made and actions must be taken. Therefore, they should be fulfilled. However, and to perform a more realistic prognosis, our proposal considers the number of resources and/or running instances to internally work with modified distances (more restrictive ones) which are consistent with the original distances and do not contradict the business policies.

As a summary, Fig. 8 represents the types of activities and order relations between them.

### 3. Motivating example

A relevant example of processes that have a high human interaction are medical guidelines [10], not only for the use of declarative models but also to demonstrate the importance of detecting a fault related to human health before it occurs. In this paper, we use as an example the protocol of pregnancy of the Health System of Andalusia, described in [3], but enriched with some activity patterns that may affect the execution of concurrent instances. As explained in [3], this example of declarative model includes activities with specific time restrictions, due to some tests of the pregnancy exam schedule are related to the size of the fetus and/or the legal connotations of the gestation week. Intending to facilitate the following explanations, this section only presents a part of the real example, where only a fragment of the overall process is modeled, shown in Fig. 9. We are based on Mobucon EC [11], the time extension of Declare, since it fits the necessities of the example better than others, including our proposed and previously detailed activity patterns that include multiple instances. Also, like in [3], the used language is an extension of the Mobucon framework, where the time interval and the time needed between activities have been included. The problem of modeling health protocols is an open problem in the area [12].

Detailing the example, the process begins when the patient detects a possible pregnancy (amenorrhea, day 0). Then starts the process of booking medical appointments, attending medical consultations and conducting certain clinical tests, which must be performed at certain times of pregnancy (defined as ranges in weeks counted from day 0).

As mentioned, the objective of our proposal is not based on the definition of a new language, it is an extension of the existing, but only to establish the necessary patterns for the problem requirements including multiple instances patterns. In detail, for the example: (i) some activities count on time restrictions on when they should be executed, indicated with time intervals inside circles over the activities; (ii) likewise, minimum and maximum time distances between activities are indicated in the corresponding

**Table 1**  
CSP solutions.

$t_A$	$t_B$	$t_C$	$t_D$
5	6	7	-1
5	7	8	-1
5	8	9	-1
5	6	8	7
5	7	9	8

arrows; (iii) and also, some activities present different behaviors when executing multiple concurrent instances, following some patterns, such as the execution of the activity *Blood test* is limited by resources of type  $R1$ , the *Gynecologist Appointment* only can take place in certain dates, and the *Urine Culture Test* is a batch activity that can be performed when the executions of a certain number of instances are synchronized there.

### 4. Constraint programming to prognosticate declarative Models

Based on the declarative model extended with time patterns and a set of traces of events, it is possible to verify and prognosticate the system. The proposed methodology for prognosticating is based on the system robustness, which is described as the capacity of a system to work correctly under unexpected situations. To compute the robustness of a system, we use the concept of super solutions coming from the Constraint Programming paradigm. Some definitions must be introduced to understand the meaning of super solution.

Constraint programming is based on the algorithmic resolution of Constraint Satisfaction Problems (CSP), and is an Artificial Intelligence technique which provides us a way to model declarative models, describing what can happen or what is prohibited instead of exactly what must occur.

**Definition 6 (Constraint Satisfaction Problem (CSP)).** A CSP [13] consists of the triple  $(V, D, C)$ , where  $V$  is a set of  $n$  variables  $v_1, v_2, \dots, v_n$  whose values are taken from finite domains  $D_{v_1}, D_{v_2}, \dots, D_{v_n} \in D$  respectively, and  $C$  is a set of constraints on their values. A constraint  $c_k (x_{k_1}, \dots, x_{k_j}) \in C$  is a predicate that is defined on the Cartesian product  $D_{k_1} \times \dots \times D_{k_j}$ . This predicate is true iff the value assignment of these variables satisfies the constraint  $c_k$ .

**Definition 7 (Solutions of a CSP).** The solutions of a CSP  $Const$  are the elements of the set of tuples  $(\langle v_1, v_2, \dots, v_n \rangle, \langle val_1, val_2, \dots, val_n \rangle)$ , where  $val_i \in D_{v_i}$ ,  $v_i$  and every value  $val_i$  assigned simultaneously to a variable  $v_i$  satisfy the constraints  $C$  of  $Const$ .

To facilitate the understanding of these definitions, we proceed to illustrate them with the following example:

*Variables and domains:*

$t_A, t_B, t_C, t_D$ : Integer

*Constraints*

$t_A < t_B$

$t_B < t_C$

$t_D > 0 \rightarrow t_B < t_D \wedge t_D < t_C$

When determining if this CSP has any solution, it is possible to count on previous instantiations of some variables (establishing specific values or constraining their domains even more), therefore it is necessary to run the CSP solver to obtain only the possible values (if any) of the remaining variables.

For our example, a previous instantiation could be:  $t_A = 5, t_C < 10$ . In this specific case, the solutions found by the solver would be the ones collected in Table 1.

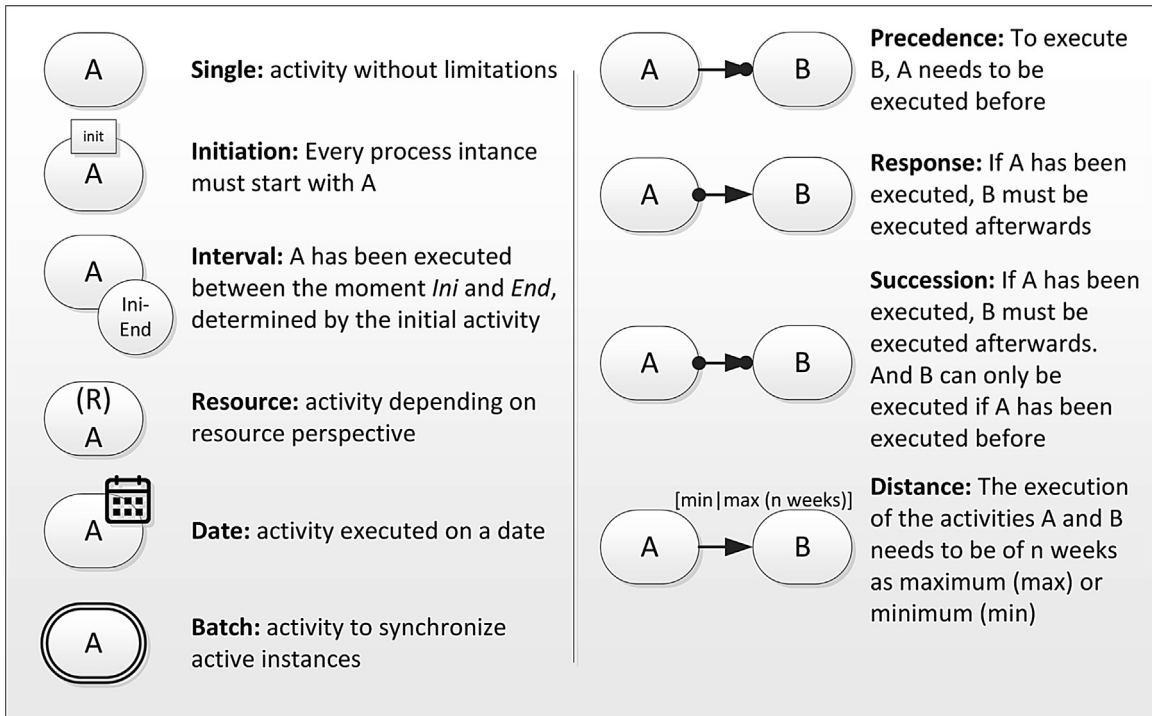


Fig. 8. Declarative activities and relations.

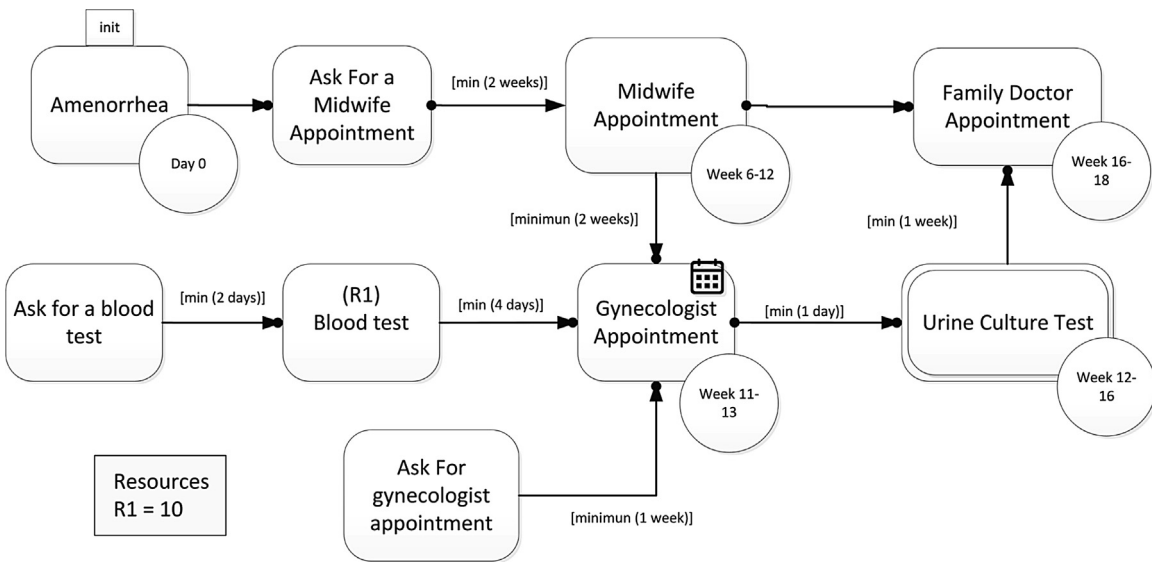


Fig. 9. Motivating example.

In this proposal, being a CSP a model that represents the declarative model and the running instances in a concrete current time, if the CSP has a solution, it means it is possible to finish the execution of the instances as expected. However, if no solution is found, the execution may not finish properly. The small example presented above would represent the CSP model, instantiation and resolution of the small declarative example in Fig. 10, with only one running instance.

To carry out this idea of using Constraint Programming for our prognosis process, the CSP creation and resolution process depicted in Fig. 11 is followed. For this, as can be seen in the diagram, the CSP model is first obtained from the complete declarative model (that is, both its structure and everything related to temporal constraints and multiple instances).

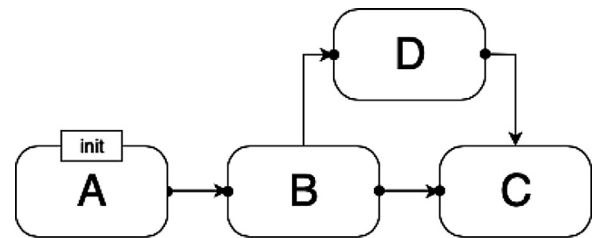


Fig. 10. Small declarative BP model.

Once the CSP Model has been obtained, the information contained in the event log is used to partially instantiate the CSP (that is, events already occurred in traces executed or in execution). In this way, a CSP is obtained where the non-instantiated variables are

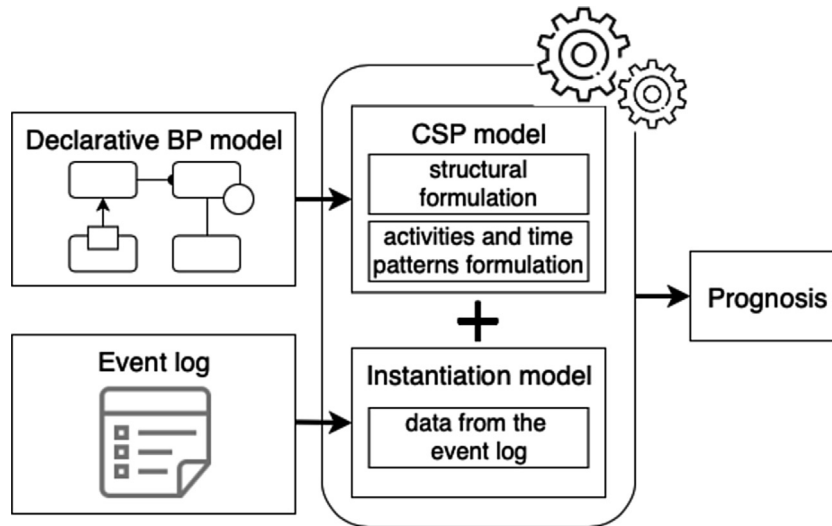


Fig. 11. CSP creation and resolution process.

those that represent the events that have not yet occurred. Thus, the resolution of this CSP allows seeing if, under the current conditions, the future execution of these events would be possible or not.

To model the problem as a CSP, we should distinguish between the modeling of (1) event traces for multiple instances, and (2) the declarative modeling of activity patterns, time intervals of the activities, and time distance between activities. First, we detail the structural formulation of the declarative model, which is based on the formulation presented in [3] with some changes due to the treatment of multiple instances.

Regarding Definition 6, the parts of the CSP adapted to our problem are:

- **Variables:** There is a variable for each instance and each activity, which represents when the activity was executed for an instance. These variables represent the formulation of the event traces.
- **Domain:** For each variable, the finite set of *Long* values that represent the plausible timestamps of the execution of each activity.
- **Constraints:** According to the declarative model, the permitted time relations between the variables must be included, which represent the activities and time patterns formulation.

#### 4.1. Formulation of event traces

In the modeling of the problem as a CSP, all running instances at a particular time should be considered. Then, there is a variable in the CSP for each execution of each activity, which represents its timestamp.

Regarding the treatment of multiple instances, these events are modeled in the CSP by means of two arrays for each activity: the Boolean array  $Act_{EX}$  and the Integer array  $Act_T$ . Each boolean in the  $i$ -th position of  $Act_{EX}$  indicates if the  $i$ -th instance executes the activity  $Act$  either in the past or in the future. Similarly, each Integer in the  $i$ -th position of the array  $Act_T$  points out the particular time when the  $i$ -th instance executes the activity  $Act$ ,  $-1$  otherwise. We are inspired by the formulation in [3], but it was limited to one instance. Derived from the new patterns introduced, where constraints among different instances are included, the CSP must consider variables for every instance. To represent if an activity has been executed or not for an instance, the following constraints are included in the model:

- If an activity has not been executed for a particular instance, then its execution time for that instance should be greater than the current time, due to it would be executed in the future.
- The model should therefore support the representation of executed and future events, as a tuple  $(Instance, Activity, Time, Resources)$ , being each concept, respectively, the concrete *Instance* that executed *Activity* in a particular *Time* using *Resources*. Each event  $(ins, act, t, r)$  should satisfy the following rules [14]:
  - if  $ins$  already executed  $act$ , then  $t \leq currentTime$  and  $act_{EX}[ins] = true$ .
  - if  $ins$  did not execute  $act$  yet, but it will do it in the future, then  $t > currentTime$ .
  - if  $ins$  does not execute  $act$  at any time, then  $t = -1$  and  $act_{EX}[ins] = false$ .

#### 4.2. Activities and time patterns formulation

This subsection details the formulation of each previously mentioned activity pattern, as well as the time limitations (i.e. time intervals and time distances between activities).

Likewise, the formulation of the activity relations is depicted in Fig. 12.

- **Activities with limited resources:** their formulation requires an array *resources* storing which type of resource is used by each activity, and an array  $num\_r$  *resources* holding the number of available resources per type (i.e. how many available resources per type and time unit). In case the execution of an activity is not affected by limited resources (or it counts on infinite units), the corresponding position in *resources* holds a zero.

Then, if, for example, activities  $A_x$  and  $A_y$  need the same limited resource of type  $R_i$  (i.e.  $resources[x] = resources[y] = i$ ), the  $ith - 1$  position of  $num\_r$  *resources* holds a number  $N$  indicating the amount of available resources of type  $R_i$  per time unit. So, the values in the arrays  $A_{xT}$  and  $A_{yT}$  are constrained so that they cannot be repeated more than  $N$  times among both arrays. This way, and due to  $A_{xT}$  and  $A_{yT}$  store the times when  $A_x$  and  $A_y$  are executed respectively, we are therefore constraining the number of uses of  $R_i$  per time unit.

$$\forall a \cdot ((resources[a] = i) \rightarrow \forall value \in \cup A_{aT} (\sum A_{aT}.cardinality(value) \leq num\_resources[i])) ; 1;$$

- **Activities executed on a date:** their execution is limited to certain dates, which can be classified as follows.


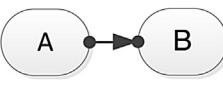
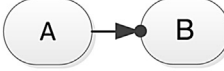

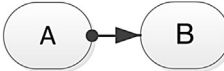

Template	Logical Formula	Template	Logical Formula
	$A_{Ex} \wedge A_T = 0$		$A_{Ex} \rightarrow B_{Ex} \wedge A_T < B_T$ $\wedge B_{Ex} \rightarrow A_{Ex} \wedge A_T < B_T$
	$B_{Ex} \rightarrow A_{Ex} \wedge A_T < B_T$		$A_{Ex} \wedge A_T \geq \text{Ini}$ $A_T \leq \text{End}$
	$A_{Ex} \rightarrow B_{Ex} \wedge A_T < B_T$		$\text{min: } A_T + n \geq B_T$ $\text{max: } A_T + n \leq B_T$

Fig. 12. Formulation of declarative patterns.

–If an activity  $a$  can only be executed in some date/s known a priori (either periodical or concrete dates), this just limits the values defined in the domain of the array  $A_T$ .

For concrete dates:

$D: \{date1, date2, date3, \dots\}$

;1;

For periodical dates:

$D: \{date, date + period, date + 2 * period, date + 3 * period, \dots\}$

;1;

–If the dates when an activity  $A$  can be executed are known but limited to some conditions, it requires the inclusion in the CSP of a constraint accordingly. For example, if a new execution of  $A$  can only be performed at least  $minTime$  time units and at most  $maxTime$  time units from the last one, the constraint would be:

$\forall x, y \in A_T (x = y \vee minTime \leq |x - y| \leq maxTime)$

;1;

- **Batch activities**, whose execution pattern entails some considerations:

–The different executions of a batch activity take place when a batch of at least  $N$  instances reaches it. Then, to perform the prognosis process, we should constraint the future execution times in  $A_{BT}$  of each batch activity  $A_B$  for the running instances that have not reached  $A_B$  at the current time.

–Due to the limitation of at least  $N$  instances reaching  $A_B$  to allow its execution, it can come up that, after the execution of some prior batches, a remainder of  $M$  instances ( $M < N$ ) is still running. To deal with it, our methodology would consider that these  $M$  running instances could finish on time iff it is still possible the appearance of at least  $N - M$  new instances which could arrive on time to the batch activity (that is, making possible the fulfillment of the  $M$  running instances without violating any time restriction), therefore allowing the execution of all instances in batch. In the following, we name this fact as *future arrival of instances on time*.

To formally formulate these concepts, we add the next variables and constraints to the CSP model.

–An array *batch* is used to store on each  $i$ -th position the minimal number of running instances that are needed to execute each  $i$ -th activity of the model. In case an activity is a batch activity, its corresponding value would be greater than 1.

–Also, and to determine the possibility of a *future arrival of instances on time*, the calculation of the minimal time it takes to arrive at each activity from the activity *init* is performed. These minimal times are obtained considering the best cases regarding time, hence by adding time distances and intervals of time in the fastest path to each activity.

For the example in Fig. 9, the calculations are shown in the following box:

$int\ minTime\_Amenorrhea = 0$

$int\ minTime\_AskForMidwifeA = minTime\_Amenorrhea + 1$

$int\ minTime\_AskForABloodT = 0$

$int\ minTime\_BloodTest = minTime\_AskForABloodT + 2$

$int\ minTime\_MidWifApp = max(minTime\_AskForMidwifeA + 14, 5 * 7 + 1)$

$int\ minTime\_AskForGynecologistApp = 0$

$int\ minTime\_GynecologistApp =$

$max(10 * 7 + 1, minTime\_MidWifApp + 14, minTime\_BloodTest + 4, minTime\_AskForGynecologistApp + 7)$

$int\ minTime\_UrineCulture = max(minTime\_GynecologistApp + 1, 11 * 7 + 1)$

$int\ minTime\_FamilyDocApp =$

$max(15 * 7 + 1, minTime\_MidWifApp, minTime\_UrineCulture + 7)$

;1;

–Finally, and being  $A_B$  a batch activity, the values in  $A_{BT}$  are constrained taking into account three aspects: (1) the particular times when the activity  $A_B$  was executed in the past (i.e.  $\forall i | A_{BT}[i] < currentTime$ ) are not verified since they are supposed to be correct; and the future executions of  $A_B$  should be carried out (2) in batches of at least  $N$  instances (where  $N = batch[A_B]$ ), or (3) in a particular time that is after  $currentTime + minTime\_A_B$  and hence allowing the *future arrival of instances on time*.

$\forall x \text{ in } A_{BT} ((x > currentTime) \rightarrow A_{BT}.cardinality(x) \geq batch[A_B] \vee x > currentTime + minTime\_A_B)$

;1;

- **Time distances between activities:** as it was aforementioned, these distances are internally modified to more restrictive versions so that we can get a more accurate prognosis. More precisely, the minimum time would be increased under certain circumstances, such as lack of resources and/or too many running instances. So, these new distances are no longer fixed, but they are dependent on these changing elements.

–Regarding the resources' availability, and being  $A$  an activity that should be executed in a determined interval of time  $I$ , if the amount of available resources for  $A$  is under a certain threshold, the distance of time between the activity (or activities) just preceding  $A$  should be increased. This is due to the possible delay in the execution of  $A$  because of the lack of available resources, causing the need of anticipation of the prior activities so that the execution of  $A$  in  $I$  may be fulfilled.

–As for the number of running instances, we have the opposite case: the more instances are running, the more the execution



**Table 2**  
Structural formulation of the motivating example.

```

Variables and domains
AmenorrhoeaEx[], AskForMidwifeAppointmentEx[], BloodTestEx[],
...: Boolean
AmenorrhoeaT[], AskForMidwifeAppointmentT[], BloodTestT[], ...:
Integer
...

Structural formulation: compliance rules of the model
for each instance ins
(AmenorrhoeaEx[ins] ∨ AmenorrhoeaT[ins] = -1)
(AskForMidwifeAppointmentEx[ins] ∨
AskForMidwifeAppointmentT[ins] = -1)
...
end for
for each instance ins
AskForMidwifeAppointmentEx[ins] → AmenorrhoeaEx[ins]
∧ AmenorrhoeaT[ins] < AskForMidwifeAppointmentT[ins]
AskForMidwifeAppointmentEx[ins] → MidwifeAppointmentEx[ins]
∧ AskForMidwifeAppointmentT[ins] < MidwifeAppointmentT[ins]
MidwifeAppointmentEx[ins] → FamilyDoctorAppointmentEx[ins]
∧ MidwifeAppointmentT[ins] < FamilyDoctorAppointmentT[ins]
∧ FamilyDoctorAppointmentEx[ins] →
FamilyDoctorAppointmentEx[ins]
∧ MidwifeAppointmentT[ins] < FamilyDoctorAppointmentT[ins]
...
end for

Time constraints
for each instance ins
tmi = AmenorrhoeaT[ins]
tmi + 42 ≤ MidwifeAppointmentT[ins] ≤ tmi + 84
tmi + 112 ≤ FamilyDoctorAppointmentT[ins] ≤ tmi + 126
...
end for

```

of the prior activity/activities should be anticipated (i.e. the minimum distance should be increased too).

In order to include these ideas, we consider that the minimum distance [ $\min(\text{time})$ ] between a pair of activities  $A, B$  were defined for (i) an optimal number of instances running ( $\text{optI}$ ); and, (ii) in case the execution of  $B$  is limited by resources of type  $R$ , an optimal number of resources of this type ( $\text{optR}$ ). Therefore, if the number of running instances is more than  $\text{optI}$ , and/or the number of available resources is less than  $\text{optR}$ , thus the new distance should be calculated. Then, being  $\text{numI}$  the number of running instances and  $\text{numR}$  the number of available resources, the distance between each pair of activities would be as follows:

$$[\min(\text{time} * (\max(1, \text{numI}/\text{optI})) * (\max(1, \text{optR}/\text{numR})))$$

This way, the minimum time would be affected only if there are more instances and/or fewer resources than enough.

#### 4.3. CSP model of the motivating example

To illustrate these described ideas to get a CSP model, Table 2 shows a fragment of the CSP created for the motivating example in Section 3, including (i) the declaration of variables, (ii) the explained structural formulation, and (iii) the constraints regarding time limits.

Likewise, Table 3 collects a fragment of the formulation of the activities and time patterns for the motivating example in Section 3.

### 5. Automating prognosis based on robustness analysis

During the execution of a system and depending on its declarative model, it is possible to detect inconsistencies (*verification* task in Fig. 1) by checking the expected behavior of the system, defined by the model, versus the real behavior collected in an event log. Besides, in this paper, we do not only want to check if a partial

**Table 3**  
Activities and time pattern formulation of the motivating example.

```

Formulation of resources
for each resource r
bagT = bag with the content of the arrays ActT of all
activities needing r
for each value v in bagT
bagT.cardinality(v) ≤ num.resources[r]
end for
end for

Formulation of activities executed on a date
for each activity act with execution on certain dates
for each par of instances ins1, ins2
|actT[ins1]-actT[ins2]| ≡ 0 (mod P) //being P the periodicity of
execution of act
end for
end for

Formulation of batch activities
for each batch activity act
for each value v in actT
if v > currentTime
actT.cardinality(v) > batch[act] ∨ v > currentTime + minTimeAct
end for
end for

Formulation of time distances
AskForMidwifeAppointmentT[ins] + (14 * max(1, numI/20))
≥ MidwifeAppointmentT[ins]
BloodTestT[ins] + (4 * (max(1, numI/20)) * (max(1, 10/numR)))
≥ GynecologistAppointmentT[ins]
...

```

**Table 4**  
Event log (on date 30-Nov-2018).

Case ID	Activity name	Timestamp	Resource	...
1	Amenorrhoea	27-Sep-2018	-	...
2	Amenorrhoea	4-Oct-2018	-	...
3	Amenorrhoea	9-Oct-2018	-	...
4	Amenorrhoea	9-Oct-2018	-	...
5	Amenorrhoea	10-Oct-2018	-	...
5	Ask for midwife appointment	22-Oct-2018	-	...
3	Ask for midwife appointment	27-Oct-2018	-	...
2	Ask for midwife appointment	5-Nov-2018	-	...
5	Ask for a blood test	6-Nov-2018	-	...
5	Blood test	11-Nov-2018	R1=1	...
2	Ask for a blood test	16-Nov-2018	-	...
3	Ask for a blood test	16-Nov-2018	-	...
3	Midwife appointment	18-Nov-2018	-	...
4	Ask for midwife appointment	20-Nov-2018	-	...
3	Blood test	21-Nov-2018	R1=1	...
2	Blood test	21-Nov-2018	R1=2	...
1	Ask for midwife appointment	22-Nov-2018	-	...
1	Ask for a blood test	25-Nov-2018	-	...
4	Ask for a blood test	28-Nov-2018	-	...
5	Midwife appointment	29-Nov-2018	-	...

instance is satisfiable, but we also want to prognosticate potential problems to avoid malfunctions in the future for multiple instances that are executed concurrently (*prognosis* task in Fig. 1), determining the activities which are the source of inconsistencies (*critical activities analysis* task in Fig. 1).

These three tasks addressed in this paper, which are related to model-based prognosis, are based on Constraint Programming to determine if the event traces analyzed during the monitoring are  $PrTs$ ,  $Rts$  or  $Wts$ . To clarify this idea, Table 4 shows  $PTs$  for the example in Fig. 9, collected in the event log. Thanks to the use of Constraint Programming, it is possible to know if these  $PTs$  are  $PrTs$ , meaning that there exists a way to finish all instances satisfying the compliance rules for the possible events in the future. Therefore, using Constraint Programming, future events could be inferred for the non-executed activities to get  $FTs$ , trying to find out a solution of the CSP.

**Table 5**  
Instantiation from the event log.

---

Instantiation of variables according to past cases in the event log (Table 4)

```
AmenorrhoeaT[0] = 0
AmenorrhoeaT[1] = 7
...
AskForMidwifeAppointmentT[5] = 25
AskForMidwifeAppointmentT[3] = 30
...
currentTime = 64
```

---

**Table 6**  
Objective function to get the widest FTs.

---

Objective function

```
Maximize (AmenorrhoeaEx[] + BloodTestEx[] + ...)
```

---

In the following subsections, the automatic computation of both verification and prognosis are explained.

### 5.1. Computing verification for declarative models and multiple instances

In order to compute the verification of multiple traces, the complete formulation of the CSP as presented in Section 4 must be created, which means: (i) *Variables (V)*, formed by the aforementioned arrays  $Act_{Ex}$  and  $Act_T$ , whose (ii) *Domains (D)* are Boolean and Long respectively, and being affected by the (iii) *Constraints (C)* previously defined as structural and patterns formulation in Section 4.

The CSPs are created according to the trace, at run-time when a verification process must be executed.

Regarding the content of the arrays  $Act_T$ , they collect the times when each activity of the model is executed by each instance. Then, this content is in line with the granularity of the events recorded in the event log [15].

In order to illustrate the verification, Table 5 shows the setting of variables that correspond to past events collected in the event log in Table 4, which is the instantiation model that should be included to the already defined CSP model (in Tables 2 and 3).

To assure that current traces are promising, this CSP must be solved and all variables should be instantiated, so that it is possible to assign a future time of execution to all non-executed activities for each trace, therefore obtaining at least a FT for each  $PrT$ .

### 5.2. Computing prognosis for declarative models and multiple instances

To prognosticate declarative models, once the aforementioned CSP for verification is built, some considerations should be made, such as (1) how to obtain the FT?; (2) when is the model robust?; and (3) how to get alternative solutions? Let us detail them below.

#### 5.2.1. How to obtain the FT?

As explained, the satisfiability of the CSP built in the verification phase implies that there is at least a FT for each running  $PrT$  (i.e. for each current instance). This assures all running traces may be able to finish on time. However, and to perform the best prognosis, we are interested in obtaining the widest FT for each concurrent  $PrT$ , so that more activities are analyzed.

The way to get those widest FTs is by transforming the CSP into a Constraint Optimization Problem (COP), which is a CSP that also includes an objective function to be optimized. The objective then is to maximize the number of executed activities (that is, the maximization of the number of *true* values in the arrays  $Act_{Ex}$ , as shown in Table 6).

**Table 7**  
Example of variables and constraints for (1-0)-super solutions.

---

Example of variables

```
act1T[], act2T[], act1Ex[], act2Ex[]
act2T[] ' //act2 should be executed in the future
```

Example of the constraints' format

```
for each instance i
act2Ex[i] → act1Ex[i] ∧ act1T[i] < act2T[i]
act2Ex[i] → act1Ex[i] ∧ act1T[i] < act2T[i] '
act2T[i] < act2T[i] ' ∧ currentTime < act2T[i] '
end for
```

---

#### 5.2.2. When is the system robust?

In case the verification determines that there are FTs for all current instances, the model would be robust if those FTs are also RTs. That is, the model is robust iff all activities that should be executed in the future as part of those FTs (of all instances) can be executed in at least two different times. So, the current instances would count on alternative solutions to successfully end on time.

As mentioned, the performed prognosis should determine the existence of different options to execute upcoming activities. To carry this out, we propose the use of super solutions in Constraint Programming, where the idea is to know if, when an expected behavior occurs, the system has several options to evolve during the future and be satisfiable.

**Definition 8** ((*m,n*)-super solution). An (*m,n*)-super solution of a CSP is a solution in which, if  $m$  variables  $v_1, \dots, v_m$  cannot take their values  $val_1, \dots, val_m$ , another solution of the same CSP can be found where  $v_1, \dots, v_n$  take values  $val'_1 \in D_{v_1}, \dots, val'_m \in D_{v_m}$  and at most  $n$  other variables must change their values.

Super solutions are a generalization of super models in propositional satisfiability. In this paper we use (1,0)-super solutions, where if one variable cannot take its value in a solution of a CSP, it is possible to find another solution by re-assigning this variable  $v_i$  with a new value  $val'_i$  [16], being not necessary to change the value of the remaining variables.

More precisely, the variables whose values can be reassigned are those that correspond to activities that have not been executed yet. In the case of one of those activities has only one possible date to be executed, we would be facing a non-robust solution.

To translate this idea into the presented CSP, so that it is possible to determine if the current traces are robust, certain new variables are included in the CSP to represent that the upcoming activities can be executed in at least two different future times. The general idea is shown in Table 7, which consists of duplicating those not executed activities.

Therefore, to determine the robustness, the CSP formulation of all upcoming activities must be complemented by including their corresponding duplicates at the variables section, and also duplicating the constraints where they are involved. Once this new CSP is solved, the critical activities which are not robust (i.e. do not count on two possibilities of execution) are established.

To clarify the idea of the new CSP formulation, Table 8 shows the affected fragments of the CSPs in Tables 2 and 3 where the formulations of the non executed activity *MidwifeAppointment* for instance 1 has been duplicated.

## 6. Evaluation

This section aims to analyze the effectiveness and efficiency of the presented approach. So, this section provides an empirical study for performing the prognosis of multiple traces running in a declarative process scenario, as well as analyzing the robustness of the non-conforming cases, based on Constraint Programming. Taking

**Table 8**  
Example of CSP formulation for determining robustness.

---

*Variables and domains*  
 Amenorrhoea<sub>Ex</sub>[], MidwifeAppointment<sub>Ex</sub>[], ...: Boolean  
 Amenorrhoea[], MidwifeAppointment<sub>T</sub>[], ...: Integer  
 MidwifeAppointment<sub>T</sub>'[], ...: Integer

*Structural formulation: compliance rules of the model*  
 ...  
 (MidwifeAppointment<sub>Ex</sub>[1]  $\vee$  MidwifeAppointment<sub>T</sub>[1] = -1)  
 (MidwifeAppointment<sub>Ex</sub>[1]  $\vee$  MidwifeAppointment<sub>T</sub>[1]' = -1)  
 ...  
 AskForMidwifeAppointment<sub>Ex</sub>[1]  $\rightarrow$  MidwifeAppointment<sub>Ex</sub>[1]  
 $\wedge$  AskForMidwifeAppointment<sub>T</sub>[1] < MidwifeAppointment<sub>T</sub>[1]  
 AskForMidwifeAppointment<sub>Ex</sub>[1]  $\rightarrow$  MidwifeAppointment<sub>Ex</sub>[1]  
 $\wedge$  AskForMidwifeAppointment<sub>T</sub>[1] < MidwifeAppointment<sub>T</sub>[1]'  
 ...

*Time constraints*  
 $T_{ini} + 42 \leq \text{MidwifeAppointment}_T[1] \leq T_{ini} + 84$   
 $T_{ini} + 42 \leq \text{MidwifeAppointment}_T[1]' \leq T_{ini} + 84$   
 ...  
 MidwifeAppointment<sub>T</sub>[1] < MidwifeAppointment<sub>T</sub>[1]'  
 $\wedge$  currentTime < MidwifeAppointment<sub>T</sub>[1]'

*Formulation of time distances*  
 AskForMidwifeAppointment<sub>T</sub>[1] + (14 \* max(1, numI/20))  
 $\geq$  MidwifeAppointment<sub>T</sub>[1]  
 AskForMidwifeAppointment<sub>T</sub>[1] + (14 \* max(1, numI/20))  
 $\geq$  MidwifeAppointment<sub>T</sub>[1]'  
 ...

---

these aspects into account, the current evaluation addresses two evaluation questions (EQs):

- EQ1, which checks the suitability of the proposed methodology for the prognosis of multiple traces, using Constraint Programming. Specifically, it may be divided into two subquestions, since both effectiveness (EQ1(1)) and efficiency (EQ1(2)) of the method are checked.
- EQ2, which checks the efficiency of the proposed analysis of robustness by using (1-0)-super solutions.

For this case study, the full example of the pregnancy protocol is used. It consists of 28 activities (including all different activity patterns), 33 relation orders, 12 interval relations, and 17 distance relations.

To answer the EQs, these analyses should be performed:

- To answer EQ1(1), we count on 280 event logs recording compliant partial traces. To check the effectiveness, either these traces or even the original declarative model are intentionally modified to introduce or change data or components of the model so that the partial traces are no longer compliant. When performing the prognosis of these modified cases, the proposed method behaves correctly for the 100 percent of them, identifying their non-conformity.
- To answer EQ1(2), while these cases (conforming or non-conforming to the model) are checked, the execution times are measured.
- Likewise, to answer EQ2, the robustness analysis is performed for the cases where the prognosis determines non-conformity, and the execution times are also measured.

As for the suitability of our proposal, it depends on the processes that are intended to be monitored have time intervals (between consecutive events) with a size larger than the computation time of the CSPs that have to be resolved for the prognosis. Specifically, this is of interest to Corporate Governance, since prognosis information can be obtained in a dashboard, indicating possible future violations through alerts and thus facilitating decision-making.

Regarding the resolution complexity of CSPs, its analysis is based on the significant increase of computational cost around critical values, called phase transition [17]. In CSPs, the order parameter is the constraint tightness & ratio (constraint tightness is defined as the ratio of the number of forbidden tuples to the total number of possible combinations). This causes that for low values of the order parameter (underconstrained CSPs with few constraints and therefore many possible solutions) there is low computational cost, because, having so many solutions, finding the first solution is easy and therefore fast. For high values of the order parameter (overconstrained CSPs, with many constraints, few solutions or even none) there is also low computational cost because the constraints are pruned very prematurely and the search space is reduced. Therefore, the computational problem occurs with intermediate values of the order parameter, where the cost rises substantially.

The evaluation of the proposal is run on an Intel(R) Core (TM) CPU i7, 2.2 GHz, 8 GB memory, running macOS High Sierra.

Figs. 13 and 14 show, respectively, the execution times for the verification and the prognosis processes. In both of them, the time(s) (*y*-axis) is displayed depending on the number of simultaneously running instances (*z*-axis) and the percentage of executed activities (*x*-axis) of the total number of activities of all running instances.

Both charts show acceptable times, which increase as the number of simultaneous instances increases, and decrease as the number of activities to verify or prognosticate decreases (due to the number of variables of the CSP to assign values to is reduced). It is possible to sense minor increases when several resources or batch activities are instantiated since they cause a significant domain reduction.

## 7. Related works

The complexity of the description of business process models is aligned with the new scenarios that the organizations perform to get their goals. This paper presents contexts where multiple instances influence in the execution of the declarative models that include time-perspective. The new model can be used to prognosticate a possible deviation. Thereby, the proposal in this paper is centered on the enlargement of the declarative models and the definition of mechanisms to avoid deviations of the expected behavior ahead of their appearance, so that it is not too late to solve them. For this reason, this section is divided into three subsections, according to the languages related to declarative modeling, the possibility of prognosing business process models, and the evaluation of that prognosis.

### 7.1. Declarative business process models

Declarative models have been widely used in the last decade in business processes, and then several paradigms and languages have been developed to support them. Declarative models are understood as a mechanism to specify a trajectory in state space and the constraints that describe, in a declarative way, the correct movements in that state space [18]. The differences between declarative process languages are centered in the different perceptions of what is a state, as studied in [19,20]. Some of the most relevant are the case handling paradigm [21], Artifact-centric modelling [22], PENELOPE [23], the constraint specification framework [24] or Declare language [25]. Some of them, such as PENELOPE, include the possibility to describe time as a parameter to model. However, none of them take into account multiple instance perspective and how it affects to time perspective.

The incorporation of time information can be found in several real-world examples [12], where it is necessary to include elements

## Verification

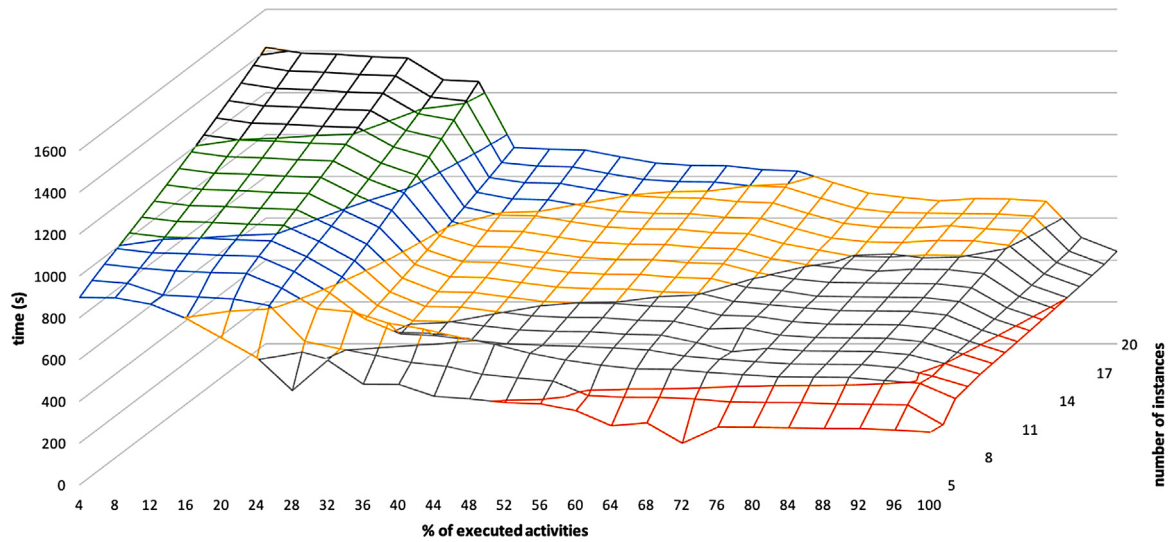


Fig. 13. Execution time of the verification process.

## Prognosis based on robustness

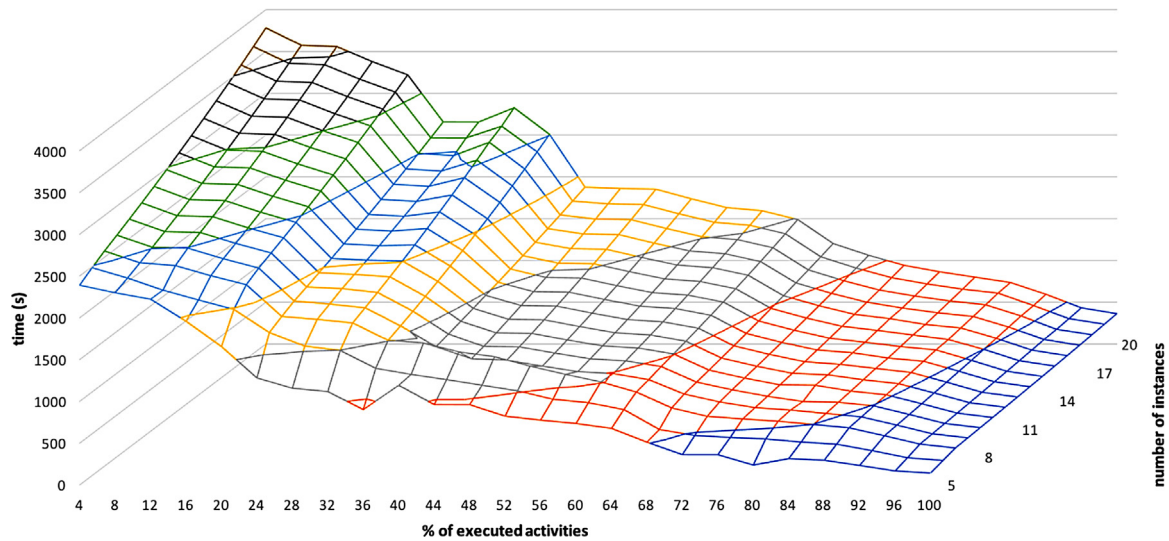


Fig. 14. Execution time of the prognosis process.

to describe the time patterns necessary in each case [26]. These elements, included in the model, can help in the fault detection or prevention, combined with a monitoring architecture [14], even when the information is partial or incomplete [27].

Different techniques have been used to model and analyze temporal elements combined with business process artifacts [28]. This specifically necessitates incorporating a notion of metric time as well as time distances and deadlines for process activities. Referring to [29] again, it can be argued that none of the existing approaches supports these temporal specifications within compliance constraints or to multiple instances.

### 7.2. Prognosing declarative models

Conformance checking has been used to verify the correctness of declarative models in different scenarios and for different lan-

guages [30,31]. The detection of redundancies or inconsistencies in declarative process models has been an object of study [32], not being tackled the inclusion of time components neither multiple instances in this analysis before our proposal.

However, not only verification is relevant. As discussed in [29], the *proactive management* of compliance violations during process run-time constitutes an important functionality, advancing a possible incongruity before it occurs. Even though the evaluation of existing approaches provided in [29] shows that some of them support proactive management, these approaches focus on detecting violations caused by the interplay between compliance constraints. By contrast, in our approach, the focus is centered on foreseeing violations that might commit during the process execution for a loss of robustness.

However, the new challenges tackled in this paper are related to the time-patterns adapted or enlarged to support how multiple

instances can be time-affected. The search of general mechanisms that help organizations to align their decisions has been analyzed previously [33], but not from the declarative model perspective. In a previous work [3], the time-aware prognosis was included, but not taking into account the multiple instance perspective.

### 7.3. Evaluation of business process prognosis

The verification of the declarative models has been guided by the type of paradigm used to model them [34]. In this paper, we opt for Constraint Programming, although it is not the unique reasoning method used for compliance monitoring, as found in the literature. Linear Temporal Logic and Event Calculus are also important references. One of the most used techniques is Linear Temporal Logic (LTL) expressions [35][36] that can be used to represent desirable or undesirable patterns. LTL formula can be evaluated by obtaining an automaton that is equivalent to the formula and checking whether a path corresponds to the automaton. Unfortunately, the use of automata does not allow us to infer the correct time intervals even before the compliance rules are activated. It is due to our proposal does not only analyze the compliance rules activated because the antecedent occurred. We include the whole model in the diagnosis process as in [37], but with the difference that in this case a declarative language is used instead of an imperative language. Another evaluation technique to verify the process correctness is the Event Calculus [38], which is a first-order logic programming formalism that represents the time-varying nature of facts, the events that have taken place at given time points and the effect that these events reflect on the state of the system. Although one of the advantages of the use of event calculus is deductive reasoning about the effects of the occurrence of events and, more importantly, the abductive reasoning to discover a hypothesis about the malfunction to explain the evidence of events. Unfortunately, it cannot propose a new set of data (events in this case) to avoid this malfunction, inferring possible faults in the future. Once a model is described, the next challenge is how to compute its robustness and correctness. We have decided to use Constraint Programming because: (1) it is a very mature area that has been applied to a wide range of problems, and with a high level of complexity; (2) it uses propagation techniques to reduce the search space efficiently; (3) there are numerous tools and algorithms to model and solve problems; (4) it permits an easy definition of the Complex data using a wide range of constraints (such as implication constraints, disjunctive constraints, reified constraints, global constraints, and channeling constraints), which allow the modeling of all parts of the problem (both declarative model with time patterns and running instances); and therefore (5) it makes possible and easier the performing of the verification, prognosis and robustness analysis effectively and efficiently.

## 8. Conclusions and future work

The detection of malfunctions in a system is very important, but it is always performed a posteriori implying that the malfunction has already happened. This is why this paper presents a framework for prognosticating a possible error before it occurs. The proposal develops a mechanism for the prognosis at run-time of declarative business processes considering the concurrent execution of multiple instances.

To this end, and as the previous modeling of the problem, we define, analyze and formalize time and activity patterns, so that different multi-instance casuistry regarding concurrency and synchronization is taken into account.

To compute the verification and prognosis automatically, the Constraint Programming paradigm is used, making use in particular

of the search of (1,0)-super solutions for the robustness analysis, and using a real scenario to deploy it.

The framework performs well in terms of execution time, with promising results due to the use of Constraint Programming.

As future work, some interesting lines are being analyzed, such as the consideration of scenarios subject to some uncertainties like, for example, to prioritize instances so that they may not execute an activity by their arrival time, or even the possibility of count on a batch execution that requires the grouping of instances with some similar characteristics.

Likewise, we plan to include activities with non-atomic executions, and also activities requiring more than one type of resource.

### Authors' contributions

**Diana Borrego:** Methodology, Software, Validation, Formal analysis, Investigation, Writing – Original Draft, Writing – Review & Editing, Visualization

**María Teresa Gómez-López:** Conceptualization, Methodology, Investigation, Resources, Supervision

**Rafael M. Gasca:** Formal analysis, Investigation, Supervision

### Conflicts of interest

The authors declare no conflicts of interest.

### Acknowledgement

This work has been partially funded by the Ministry of Science and Technology of Spain by ECLIPSE projects and the European Regional Development Fund (ERDF/FEDER).

### References

- Yan, R., Chen, X., Wang, P., Onchis, D.M., 2019. Deep learning for fault diagnosis and prognosis in manufacturing systems. *Comput. Ind.* 110, 1–2, <http://dx.doi.org/10.1016/j.compind.2019.05.002>.
- Pérez-Álvarez, J.M., Maté, A., López, M.T.G., Trujillo, J., 2018. Tactical business-process-decision support based on kpis monitoring and validation. *Comput. Ind.* 102, 23–39, <http://dx.doi.org/10.1016/j.compind.2018.08.001>.
- Gómez-López, M.T., Parody, L., Gasca, R.M., Rinderle-Ma, S., 2014. Prognosing the compliance of declarative business processes using event trace robustness. In: *On the Move to Meaningful Internet Systems: OTM 2014 Conferences – Confederated International Conferences: CoopIS, and ODBASE 2014, Amantea, Italy, October 27–31, 2014, Proceedings*, pp. 327–344, [http://dx.doi.org/10.1007/978-3-662-45563-0\\_19](http://dx.doi.org/10.1007/978-3-662-45563-0_19).
- Liu, J., Hu, J., 2007. Dynamic batch processing in workflows: model and implementation. *Future Gener. Comput. Syst.* 23 (3), 338–347, <http://dx.doi.org/10.1016/j.future.2006.06.003>.
- Pufahl, L., Weske, M., 2013. Batch activities in process modeling and execution. In: *Service-Oriented Computing – 11th International Conference, ICSOC 2013, Berlin, Germany, December 2–5, 2013, Proceedings*, pp. 283–297, [http://dx.doi.org/10.1007/978-3-642-45005-1\\_20](http://dx.doi.org/10.1007/978-3-642-45005-1_20).
- Goedertier, S., Vanthienen, J., Caron, F., 2015. Declarative business process modelling: principles and modelling languages. *Enterprise IS* 9 (2), 161–185.
- Maggi, F.M., Westergaard, M., Montali, M., van der Aalst, W.M.P., 2011. Runtime verification of ltl-based declarative process models. In: *Runtime Verification – Second International Conference, RV 2011, San Francisco, CA, USA, September 27–30, 2011*, pp. 131–146, [http://dx.doi.org/10.1007/978-3-642-29860-8\\_11](http://dx.doi.org/10.1007/978-3-642-29860-8_11).
- van der Aalst, W.M.P., 2011. *Process Mining – Discovery, Conformance and Enhancement of Business Processes*. Springer, <http://dx.doi.org/10.1007/978-3-642-19345-3>.
- Pesic, M., van der Aalst, W.M.P., 2006. A declarative approach for flexible business processes management. In: *Proceedings of the 2006 International Conference on Business Process Management Workshops, BPM'06*. Springer-Verlag, Berlin, Heidelberg, pp. 169–180.
- Mulyar, N., Pesic, M., van der Aalst, W.M.P., Peleg, M., 2007. Declarative and procedural approaches for modelling clinical guidelines: addressing flexibility issues. *Business Process Management Workshops*, Vol. 4928 of *Lecture Notes in Computer Science*, 335–346.
- Montali, M., Maggi, F.M., Chesani, F., Mello, P., Aalst, W.M.P.v.d., 2014. Monitoring business constraints with the event calculus. *ACM Trans. Intell. Syst. Technol.* 5 (1).
- Dunkl, R., Fröschl, K.A., Grossmann, W., Rinderle-Ma, S., 2011. Assessing medical treatment compliance based on formal process modeling. In: *Information Qual-*

- ity in e-Health – 7th Conference of the Workgroup Human-Computer Interaction and Usability Engineering of the Austrian Computer Society, USAB 2011, Graz, Austria, November 25–26, 2011, pp. 533–546, [http://dx.doi.org/10.1007/978-3-642-25364-5\\_37](http://dx.doi.org/10.1007/978-3-642-25364-5_37), Proceedings.
- Rossi, F., van Beek, P., Walsh, T. (Eds.), 2006. Handbook of Constraint Programming. Elsevier.
- Gómez-López, M.T., Gasca, R.M., Rinderle-Ma, S., 2013. Explaining the incorrect temporal events during business process monitoring by means of compliance rules and model-based diagnosis. In: 17th IEEE International Enterprise Distributed Object Computing Conference Workshops, EDOC Workshops, Vancouver, BC, Canada, September 9–13, 2013, pp. 163–172, <http://dx.doi.org/10.1109/EDOCW.2013.25>.
- van der Aalst, W.M.P., et al., 2011. Process mining manifesto. In: Business Process Management Workshops – BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, pp. 169–194, [http://dx.doi.org/10.1007/978-3-642-28108-2\\_19](http://dx.doi.org/10.1007/978-3-642-28108-2_19).
- Hebrard, E., Hnich, B., Walsh, T., 2004. Super solutions in constraint programming. In: Régin, J.-C., Rueher, M. (Eds.), CPAIOR, Vol. 3011 of Lecture Notes in Computer Science. Springer, pp. 157–172.
- Cheeseman, P., Kanefsky, B., Taylor, W.M., 1991. Where the really hard problems are. In: Proceedings of the 12th International Joint Conference on Artificial Intelligence – vol. 1, San Francisco, CA, USA, pp. 331–337.
- Pesic, M., van der Aalst, W.M.P., 2006. A declarative approach for flexible business processes management. In: Business Process Management Workshops, BPM 2006 International Workshops, BPD, BPI, ENEI, GPWW, DPM, semantics4ws, Vienna, Austria, September 4–7, 2006, Proceedings, pp. 169–180, [http://dx.doi.org/10.1007/11837862\\_18](http://dx.doi.org/10.1007/11837862_18).
- Parody, L., López, M.T.G., Gasca, R.M., 2016. Hybrid business process modeling for the optimization of outcome data. *Inf. Softw. Technol.* 70, 140–154, <http://dx.doi.org/10.1016/j.infsof.2015.10.007>.
- Goedertier, S., Vanthienen, J., Caron, F., 2015. Declarative business process modelling: principles and modelling languages. *Enterp. Inf. Syst.* 9 (2), 161–185, <http://dx.doi.org/10.1080/17517575.2013.830340>.
- van der Aalst, W.M.P., Weske, M., 2005. Case handling: a new paradigm for business process support. *Data Knowl. Eng.* 53 (2), 129–162, <http://dx.doi.org/10.1016/j.datak.2004.07.003>.
- Bhattacharya, K., Gerede, C.E., Hull, R., Liu, R., Su, J., 2007. Towards formal analysis of artifact-centric business process models. In: BPM, Vol. 4714 of Lecture Notes in Computer Science. Springer, pp. 288–304.
- Goedertier, S., Vanthienen, J., 2006. Designing compliant business processes with obligations and permissions. *Business Process Management Workshops*, Vol. 4103 of Lecture Notes in Computer Science, 5–14.
- Sadiq, S.W., Orłowska, M.E., Sadiq, W., 2005. Specification and validation of process constraints for flexible workflows. *Inf. Syst.* 30 (5), 349–378, <http://dx.doi.org/10.1016/j.is.2004.05.002>.
- Pesic, M., Schonenberg, H., van der Aalst, W.M.P., 2007. DECLARE: full support for loosely-structured processes. In: EDOC, IEEE Computer Society, pp. 287–300.
- Lanz, A., Reichert, M., Weber, B., 2016. Process time patterns: a formal foundation. *Inf. Syst.* 57, 38–68, <http://dx.doi.org/10.1016/j.is.2015.10.002>.
- Chesani, F., Mello, P., De Masellis, R., Francescomarino, C.D., Ghidini, C., Montali, M., Tessaris, S., 2018. Compliance in business processes with incomplete information and time constraints: a general framework based on abductive reasoning. *Fundam. Inform.* 161 (1–2), 75–111, <http://dx.doi.org/10.3233/FI-2018-1696>.
- Combi, C., Gambini, M., Migliorini, S., Posenato, R., 2014. Representing business processes through a temporal data-centric workflow modeling language: an application to the management of clinical pathways. *IEEE Trans. Syst. Man Cybern.: Syst.* 44 (9), 1182–1203, <http://dx.doi.org/10.1109/TSMC.2014.2300055>.
- Ly, L.T., Maggi, F.M., Montali, M., Rinderle-Ma, S., van der Aalst, W.M., 2013. A framework for the systematic comparison and evaluation of compliance monitoring approaches. 17th Int'l EDOC Conference.
- Smedt, J.D., Ciccio, C.D., Vanthienen, J., Mendling, J., 2016. Model checking of mixed-paradigm process models in a discovery context – finding the fit between declarative and procedural. In: Business Process Management Workshops – BPM 2016 International Workshops, Rio de Janeiro, Brazil, September 19, 2016, pp. 74–86, [http://dx.doi.org/10.1007/978-3-319-58457-7\\_6](http://dx.doi.org/10.1007/978-3-319-58457-7_6), Revised Papers.
- Borrego, D., Gasca, R.M., López, M.T.G., 2015. Automating correctness verification of artifact-centric business process models. *Inf. Softw. Technol.* 62, 187–197.
- Ciccio, C.D., Maggi, F.M., Montali, M., Mendling, J., 2017. Resolving inconsistencies and redundancies in declarative process models. *Inf. Syst.* 64, 425–446, <http://dx.doi.org/10.1016/j.is.2016.09.005>.
- Voisin, A., Levrat, E., Cocheteux, P., lung, B., 2010. Generic prognosis model for proactive maintenance decision support: application to pre-industrial e-maintenance test bed. *J. Intell. Manuf.* 21 (2), 177–193.
- Cabot, J., Clarisó, R., Guerra, E., de Lara, J., 2010. Verification and validation of declarative model-to-model transformations through invariants. *J. Syst. Softw.* 83 (2), 283–302.
- van der Aalst, W.M.P., de Beer, H.T., van Dongen, B.F., 2005. Process mining and verification of properties: an approach based on temporal logic. *OTM Conferences (1)*, Vol. 3760 of Lecture Notes in Computer Science, 130–147.
- Awad, A., Decker, G., Weske, M., 2008. Efficient compliance checking using BPMN-Q and temporal logic. In: BPM, Vol. 5240 of Lecture Notes in Computer Science. Springer, pp. 326–341.
- Gómez-López, M.T., Gasca, R.M., Parody, L., Borrego, D., 2011. Constraint-driven approach to support input data decision-making in business process management systems. In: Information Systems Development, Reflections, Challenges and New Directions, Proceedings of ISD 2011, Heriot-Watt University, Edinburgh, Scotland, UK, August 24–26, 2011, pp. 457–469, [http://dx.doi.org/10.1007/978-1-4614-4951-5\\_37](http://dx.doi.org/10.1007/978-1-4614-4951-5_37).
- Kowalski, R.A., Sergot, M.J., 1986. A logic-based calculus of events. *New Gener. Comput.* 4 (1), 67–95, <http://dx.doi.org/10.1007/BF03037383>.