A Constraint-Based Approach for Managing Declarative Temporal Business Process Models

Andrés Jiménez-Ramírez *ajramirez@us.es*

Irene Barba *irenebr@us.es*

Carmelo Del Valle *carmelo@us.es*
Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla
Seville, Spain

Abstract

There is an increasing interest in aligning information systems in a process-oriented way. As an alternative of the traditional imperative models which tend to be too rigid, processes may be specified in a declarative (e.g., constraint-based) way. Nonetheless, in general, offering operational support (e.g., generating possible execution traces) to declarative business process models entails more complexity when compared to imperative modeling alternatives. Such support becomes even more complex in many real scenarios where the management of complex temporal relations between the process activities is crucial (i.e., the temporal perspective should be managed). Despite the needs for enabling process flexibility and dealing with temporal constraints, most existing tools are unable to manage both. In a previous work, we then proposed TConDec-R, which is a constraint-based process modeling language which allows for the specification of temporal constraints. However, TConDec-R revealed a number of limitations that are overcome with the present work. More specifically, this paper significantly extends and improves our previous work by (1) defining TConDec-R process models based on high-level elements from the constraint programming paradigm, (2) introducing a constraint-based tool with a client/server architecture for providing operational support to TConDec-R process models, and (3) performing an empirical evaluation of the approach.
Keywords: constraint satisfaction problems, constraint programming, business process modeling support, process flexibility.

1. Introduction

For several years, there has been an increasing interest in aligning information systems in a process-oriented way in order to operationalize business processes [25]. Thereby, a business process (BP) consists of a set of activities that jointly realize a business goal and whose execution needs to be coordinated in an organizational as well as technical environment [25]. BPs are commonly modeled using imperative languages, e.g,. BPMN [6] or Flowcharts. The resulting process models, however, tend to be too rigid to meet the flexibility demands of the actors involved in many real scenarios. Declarative business process languages, in turn, represent a promising modeling alternative in scenarios in which a high level of flexibility is demanded. Therefore, declarative approaches are becoming increasingly popular as they are able to cope with some of the limitations imperative notations are facing [24, 20, 18, 12, 7].

Regardless of the used process modeling paradigm, the resulting artifact can be viewed from different perspectives, including behavior [14, 7], data [17, 5], and resources [13]. Another perspective, which has not received sufficient attention yet, is the temporal one. In today's fast paced world, for any enterprise it is crucial to know the temporal properties of its business processes [8, 9, 2, 15].

In a previous work [16], we systematically analyzed 10 process time patterns (TPs for short) i.e., solutions for representing commonly occurring temporal constraints. In particular, the TPs

were defined independently of a specific language or paradigm for BP modeling [15]. Despite the needs for enabling process flexibility and dealing with temporal constraints, most existing approaches are unable to manage both. To fill this gap, we proposed the TConDec-R language [3], a declarative process modeling language that allows for the specification of temporal constraints related to the aforementioned time patterns. In particular, this language was implemented using a constraint-based approach (see [3] for details).

However, TConDec-R revealed a number of limitations that are overcome with the present work. More specifically, this paper significantly extends and improves our previous work by:

1. Defining TConDec-R process models based on high-level elements from the constraint programming (CP) paradigm.
2. Detailing a constraint-based software tool for managing TConDec-R business process models. Such tool allows (1) modeling declarative business processes through the TConDec-R language, (2) checking the correctness of TConDec-R models, (3) generating execution traces for such models, and (4) checking the conformance of given traces regarding a specific model.
3. Performing an empirical evaluation for checking the effectiveness and efficiency of the proposed approach.

2. Background

This section provides backgrounds on constraint-based process models (cf. Sect. 2.1), which are needed for understanding this work.
Furthermore, we discuss how such models can be managed through elements from the constraint programming paradigm (cf. Sect. 2.2).

2.1. Constraint-Based Process Models

As basis of the TConDec-R language [3], we use Declare [21] for specifying activities and their behavioral (i.e., control-flow) constraints. We consider this declarative modeling language as appropriate as it enables the specification of a wide range of process models in a flexible way. Respective process models are denoted as *constraint-based*, i.e., they comprise information about (1) the activities that may be performed during process enactment as well as (2) the constraints to be fulfilled in this context. Declare constraints can be categorized as existence constraints, relation constraints, and negation constraints [21].

Table 1. Selected process time patterns and examples

Cat.	**Time pattern (TP)**	**Example**
I	TP1 (Time Lags between two Activities) enables the definition of different kinds of time lags between two activities.	The time lag between registering a Master thesis and submitting it must not exceed 6 months.
	TP2 (Durations) allows specifying the duration of process activities.	Processing 100 requests must not take longer than 1 second.
II	TP5 (Schedule Restricted Element) allows restricting the enactment of a particular activity by a schedule.	Lab tests in a hospital can only be done on FR between 8 am and 5 pm.
	TP6 (Time-based Restrictions) provides support for restricting the number of times a specific process element may be executed within a given timeframe.	For a specific lab test at least 5 different blood samples have to be taken within 24 hrs.

In addition, TConDec-R extends Declare to allow the specification of 10 process time patterns (TPs) that we systematically identified in [15, 16] by analyzing a large collection of process models from various domains.

Table 1 shows an example of the 4 most common of the 10 TPs divided into two categories according to pattern semantics.[1] Category I (*Durations and Time Lags*) provides support for

[1] The full set of time patterns are grouped in 4 categories. The reader is referred to [16] for details.

expressing the durations of different process granularities (i.e., activities, activity sets, processes, or sets of process instances) as well as time lags between activities or process events (e.g., milestones). Category II (*Restricting Execution Times*), in turn, allows constraining execution times of single activities or entire processes (e.g., deadlines).

To properly cover the resource perspective, existing works (e.g., [3, 4, 13, 19, 20]) extended constraint-based specifications by additionally considering resource constraints for each enactment of a process activity. Few works [3, 7, 10, 12, 18, 20] enhanced constraint-based specifications with temporal constraints. In this context, TConDec-R language considers both the resource and the temporal perspectives. [2]

Definition 1. (TConDec-R activity). *A TConDec-R activity act=(a,dur,role) refers to a process activity a with its estimated duration dur and the role of the required resource.*

Definition 2. (TConDec-R process model). *A* ***{TConDec-R process model*** *TCRM= (Acts, C_T, Res) corresponds to an extended constraint-based process model, where Acts corresponds to a set of TConDec-R activities, C_T is a set of constraints that may include any control-flow constraint supported by Declare as well as any temporal constraint related to the time patterns (cf. Table 1), and Res represents the resource availability.*

A TConDec-R model is said to be correct if it represents a feasible problem without conflicts (i.e., there are some traces that satisfy the model). TConDec-R constraints are specified according to the graphical notation proposed for Declare constraints [21] and using the graphical notation proposed in [15] for visualizing the temporal constraints.

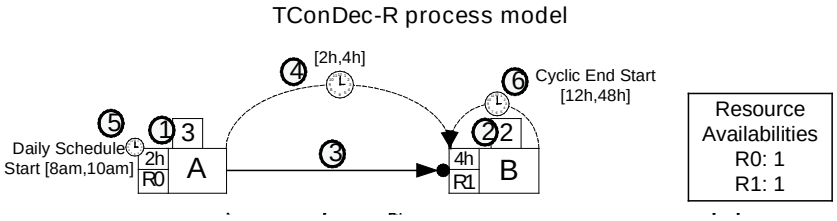

Fig. 1. A simple TConDec-R process model.

Example 1. (Simple TConDec-R process model) Figure 1 shows a simple example of a TConDec-R process model where *Acts = {(A,2h,R0), (B,4h,R1)}*, *Res* corresponds to *{(R0,1),(R1,1)}*, and C_T comprises (1) *Exactly(A,3)*, expressing that *A* shall be executed exactly three times, (2) *Exactly(B,2)*, expressing that *B* shall be executed exactly twice, (3) *Precedence(A, B)*, expressing that activity *B* may only be executed if *A* is executed before, (4) *TimeLagEndStart(A, B, 2h,4h)*, expressing that for each execution of *A*, there must be at least one execution of B such that there is a time lag of at least 2 hours and at most 4 hours, (5) *DailyScheduleStart(A, 8am, 10am)*, expressing each execution of *A* must be started between 8am and 10am, and (6) *CyclicEndStart(B, 12h, 48h)*, expressing that between the end and start of two succeeding executions of *B*, there must be a time lag of at least 12h and at most 48h.

When executing a constraint-based process model, information about the executed activities is recorded in an execution trace.

Definition 3. (Trace). *Let TCRM= (Acts, C_T, Res) be a TConDec-R process model. Then, a* ***trace*** *σ = (ID, <e_1,e_2,...e_n>) consists of an identifier ID and a sequence of start and completion events respectively. Thereby, an event e relates to a specific execution (e.g., the i-th execution) of a TConDec-R activity (a,dur,role) ϵ Acts (such execution is denoted by a_i) and has one of the following two forms: (1) e = start(a_i, R_{jk}, T), i.e., the i-th enactment of activity a using the k-th*

[2] This paper focus on the temporal perspective of TConDec-R. Details of other features like the resource perspective can be found in [3].

resource with role j was started at time T, or (2) e = comp(a_i,T), i.e., the i-th enactment of activity a was completed at time T.

2.2. Modeling Constraint-Based Process Models as Constraint Satisfaction Problems

In general, a constraint-based process model (e.g., a TConDec-R process model) can be modeled as a Constraint Satisfaction Problem (CSP), as detailed in [3]. The latter is a key concept in Constraint Programming (CP) [23], which is a powerful paradigm for correctly modeling and solving a wide variety of problems (e.g., combinatorial problems). In this section, we are mapping TConDec-R process models to CSPs with the goal of automatically dealing with the former. To be more precise, once we have mapped a TConDec-R process model to a CSP, the CSP can be implemented in any constraint-based system, i.e., we can take the advantage of a wide variety of existing algorithmic techniques and use them for different purposes, e.g., to check for the consistency of a given TConDec-R process model, to check for the conformance of specific traces with a given TConDec-R process model, or to generate traces being compliant with such model.

Definition 4. (CSP). *A **CSP** P = (V, D, C_{CSP}) is composed of a set of variables V, a set of domains D which is composed of the domain of values for each variable $var_i \in V$, and a set of constraints C_{CSP} between variables, so that each constraint represents a relation between a subset of variables and specifies the allowed combinations of values for these variables.*

A solution of a CSP assigns values to CSP variables such that the assignments satisfy all constraints. In general, a particular activity of a constraint-based process model may be executed arbitrarily often unless this number is restricted by any constraint. Accordingly, such activity may be repetitive.

Definition 5. (Repeating activity)*. A **repeating activity** ra = (a,dur,role,sacts) corresponds to a TConDec-R activity act = (a,dur,role) that may be executed several times, i.e., multiple instances of the same activity may be created in the context of a particular process instance. Additionally, sacts is the sequence of scheduling activities related to this repeating activity, i.e., the different times the process activity is executed.*

A repeating activity is then represented by a set of optional scheduling activities, where a particular scheduling activity represents a concrete instance of a process activity.

Definition 6. (Scheduling activity)*. A **scheduling activity** a_i = (ra, st, et, res) corresponds to the i-th enactment of a repeating activity ra. Thereby, st and et constitute CSP variables indicating the start and end time of the activity enactment. Moreover, res corresponds to a CSP variable representing the resource used for activity enactment.*

Based on this, a CSP-TConDec-R problem can be defined as follows [3].

Definition 7. (CSP-TConDec-R problem)*. Let TCRM = (Acts, C_T, Res) be a TConDec-R process model (cf. Def. 2) and RActs be the set of repeating activities related to TCRM (i.e., RActs = {ra | ra = (a, dur, role, sacts), (a, dur, role) ϵ Acts}). Then: A **CSP-TConDec-R problem** related to TCRM and RActs corresponds to a CSP P = (V, D, C_{CSP}) with*

- *V being a set that comprises all variables of the CSP model, i.e., V ≡{a_i.st, a_i.et, a_i.res | $a_i \epsilon$ ra.sacts, ra ϵ RActs}.*
- *D being a set covering all value domains of the respective variables from V, i.e., D ≡ {D_i | i ϵ [0,|V|)}.*
- *C_{CSP} being a set that comprises the resource constraints as well as the TConDec-R constraints included in C_T (cf. Def. 2). Furthermore, it states a specific enactment of a*

repeating activity ra ϵ RActs precedes the next enactment of the same activity, i.e., ∀a_i ϵ ra.sacts: a_i.et ≤ a_{i+1}.st.

Constraint programming allows to separate the models from the algorithms, so that once a problem is modelled in a declarative way as a CSP, a generic or specialized constraint-based solver can be used to obtain the required solution. Furthermore, constraint based models can be extended in a natural way, maintaining the solving methods.

3. Defining TConDec-R Process Models based on High-level Elements from Constraint Programming

As explained in Def. 7, we proposed the mapping of TConDec-R process models to CSPs, resulting in CSP-TConDec-R problems. In the current section, we provide a definition of the latter in terms of high-level objects and global constraints from CP. Based on this, TConDec-R process models can be implemented in any constraint-based system being able to deal with the high-level objects and constraints considered in such definition. Consequently, the wide variety of existing algorithms provided by CP becomes applicable and we can use them for different purposes, e.g., checking the consistency of a given TConDec-R process model.

According to the CSP model detailed in Section 2.2, we define the high-level object *SchedAct* to represent scheduling activities (cf. Def. 6) as follows:

SchedAct: ⟨st–CSP int, et–CSP int, res–CSP int⟩

Additionally, to represent a sequence of scheduling activities, we consider the high-level object *Sequence*, which represents a sequence of elements. Considering this, each repeating activity (cf. Def. 5) is modeled as a high-level object called *RepAct*, which includes a *sequence* of *SchedAct* objects. To be more precise, the high-level object *RepAct* is defined as follows:

RepAct: ⟨dur–int, roles–Sequence(int), sacts–Sequence(SchedAct)⟩

Note that property *roles* of the repeating activities is represented by a sequence of integers in such a way that the sequence includes the identifiers of the required roles.

This section provides a formalization of the temporal perspective of the TConDec-R language which is the most challenging perspective and was not included in the previous formalizations [3]. For this, we represent each TConDec-R constraint through an expression of global constraints [23], i.e., high-level modeling abstractions that encapsulate the behavior of a set of other constraints and therefore, allow defining more compact models, reducing the risk of modeling errors, and increasing the efficiency when solving a CSP [22]. To be more precise, the respective global constraints are taken from the catalog of global constraints as introduced in [1]. Such formalization is required in order to detail how the constraints of a TConDec-R process model are encoded when generating and implementing the respective CSP, i.e., the related CSP-TConDec-R problem.

Examples 2, 3, and 4 depict how the formalization is done for three TConDec-R constraints related to some time patterns. For this, we consider that each execution of TConDec-R activity (cf. Def. 1) have both start time (*st*) and end time (*et*) properties. In addition, being *S* a set of elements, we use the expression *S.prop* to denote the set of properties of the elements of *S*. For example, the expression *A.sacts.st* represents the start times of the scheduling activities of *A*.

***Example 2*. (TConDec-R constraint related to TP1)**. TimeLagEndStart(A,B,Low,Up) expresses that for each execution A_i of *A* there must exist at least one execution B_j of *B* in such a way that there is a time lag of at least *Low* time units and at most *Up* time units between the end time of A_i and the start time of B_j. Moreover, for each execution B_j of *B* there must exist at least one execution A_i of *A* in such a way that there is a time lag of at least *Low* time units and at most *Up* time units between the end time of A_i and the start time of B_j. We formalize

TimeLagEndStart(A, B, Low, Up) using constraint *among_interval(Value, Variables, Low, Up)* from the Global Constraint Catalog [1]. This constraint states that *Value* corresponds to the number of elements of *Variables* taking a value the is located within the interval *[Low, Up]*:

```
TimeLagEndStart(A, B, Low, Up)
      A,B: RepAtc
      Low,Up: int

forall(Ai in A.sacs){
      among_interval(1,B.sacts.st,Ai.et+Low, Ai.et+Up)
}
forall(Bi in B.sacts){
      among_interval(1,A.sacts.et,Bi.st-Low, Bi.st-Up)
}
```

Example 3. **(TConDec-R constraint related to TP2).** MaximumInstanceDuration(PI, D) expresses that the execution of a process instance[3] must not take longer than D time units. We formalize MaximumInstanceDuration(PI, D) using constraint *range(Variables, CTR, Value)* from the Global Constraint Catalog [1]. This constraint, in turn, states that the difference between the maximum and minimum value of *Variables* must fulfill constraint '*CTR Value*', where $CTR \in \{=, \neq, <, \leq, >, \geq\}$:

```
MaximumInstanceDuration(PI, D)
     PI: sequence(RepAct)
     D: int

     range(PI.sacts.st ∪ PI.sacts.et, ≤, D)
```

Example 4. **(TConDec-R constraint related to TP4)**. DeadlineEnd(A,Date) expresses that all executions of A should finish before Date. We formalize DeadlineEnd(A, Date) using global constraint *among_interval* [1]. Note that the latter was already explained in the context of Example 2:

```
DeadlineEnd(A, Date)
     A: RepAct
     Date: int

     among_interval(|A.sacts|, A.sacts.et, 0, Date)
```

The complete formalization of the TConDec-R constraints is available at http://azarias.lsi.us.es/TCR/Formalization.pdf.

Based on the high-level constraints, the behavior of the TConDec-R constraints can be monitored regarding a given trace (cf. Def. 3).

Example 5. **(States of TConDec-R constraints)**. Figure 2 depicts the states of three TConDec-R constraints (i.e., TimeLagEndStart(A,B,15,45), MaxInstanceDuration([A,B],90) and DeadlineEnd(B,11:00)), regarding a set of events of a trace. As can be seen, activities become active (ACT) after the "ts(act, i)" event (time start of the *i*-th instance of the activity *act*) happens and become completed (COMPL) after the "tc" event (time completed) is done. In turn, constraints can be in pending (PEND), satisfied (SAT) or violated (VIO) state.[4] A constraint is in satisfied state at time T if the trace fulfills such constraint at time T. However, it is in pending

[3] For the sake of simplicity, a process instance is represented as a sequence of *RepActs.*

[4] This paper uses the simplified activity life-cycle and constraint states discussed in [20]. In addition, for the sake of clarity the resources are avoided in the example.

state if it is not satisfied but there is still a chance for satisfying it with future events. In case that the constraint is impossible to be satisfied, it is in violated state. For instance, the first constraint is pending between 10:00 and 10:30 since an instance of the activity B is started at 10:00 but there is not a complete event of an instance of activity A in the time interval [14,45]. Once the instance of activity A is finished at time 10:30, the constraint became satisfied. The second constraint became violated at time 11:00 since the process instance continues after the 60 minutes. Finally, the last constraint is satisfied until the start event of the second instance of B since it stats beyond the deadline 11:00.

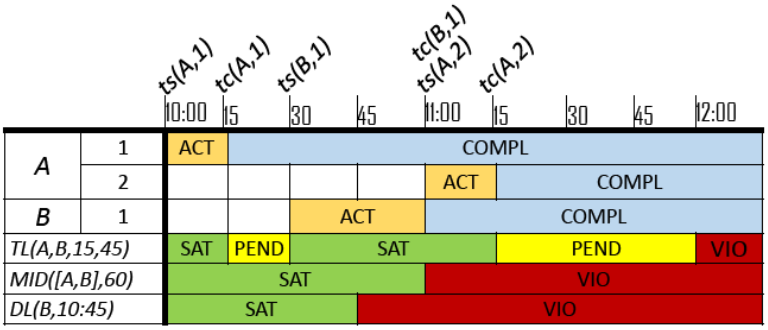

Fig. 2. States of some example constraints.

4. Constraint-Based Software Tool

The tool presented in this paper is implemented as a client-server application (cf. Fig. 3). Both sides can be deployed separately and connected through the REST API layer of the server. Moreover, the client side (cf. Sect. 4.1) deploys a light-weight web interface for modeling declarative business processes using the TConDec-R language. In turn, the server side (cf. Sect. 4.2) is in charge of (1) transforming the problem that is specified through the tool into a CSP according to the solver, (2) launching the solver and (3) interpreting the solution which is obtained by the solver.

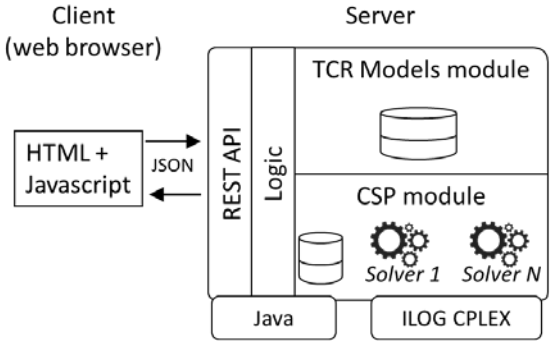

Fig. 3. Proposed architecture.

4.1. Client Side

The client comprises an HTML-based user interface (cf. Fig. 4). It can be considered as a light-weight client since it is only in charge of making the functionality offered by the server accessible.

The user is enabled to create TConDec-R specifications by including activities and constraints which are textually shown in the interface. In addition, previously created models can be loaded from the server and edited in the client through the HTML-based interface. Once models are defined and the resources which may be available in runtime are stated, different actions can be performed.

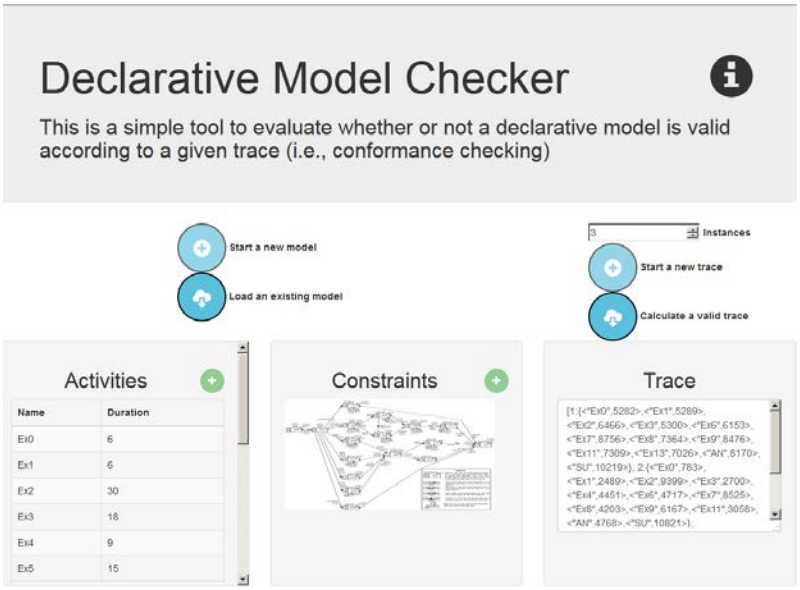

Fig. 4. HTML-based user interface of the ModelChecker tool.

For checking the correctness of the model previously specified, the server is requested for looking if the model can be instantiated or not. As explained latter, the server deploys a set of CP mechanisms for addressing this task in the CSP module. In case that the CSP module finds a solution, it means that the model is correct. In case that it explores the complete search space and there is not a solution, it means that the model is incorrect. Finally, in case that the CSP module is not able to find a solution in the given time,[5] it means that the model is too complex and the solver needs more time to elevate a conclusion beyond that.

Similar to the previous point, in order to generate traces for a given model the server will look for an instance regarding such model. In case that the model is correct, the user receives the trace which is calculated by the CSP module.

Finally, for checking the conformance of the model with a trace (or partial trace), the server will look for an instance regarding such model where all the evens of the given trace are reproducible on the instance. In case that the CSP finds a solution, it means that the conformance is checked.

4.2. Server Side

The server comprises two parts. First, the REST API which exposes the functionality in a way that it can be consumed by any client independently of the language in which it is implemented. In summary, such interface offers a series of endpoints that can be accessed via HTTP requests which trigger the different supported functionalities.

Second, the logic which eventually implements the desired functionality. For this, two different main modules are implemented. On the one hand, for managing the models there is an independent module which is in charge of that part (cf. TCR Models module in Fig. 3). Models can be created, retrieved, updated or eliminated from the system. Such module is written in Java language and stores the information in a local database.

On the other hand, the CSP module is in charge of the complex tasks, i.e., checking the correctness of models, generating traces and checking the conformance of traces. With the aim to make the architecture independent of a CSP language, this module implements inner connectors to the CSP solvers. Therefore, the CSP module first transforms the desired complex task into a CSP. Secondly, the module orchestrates the necessary executions of the solver. Finally, the solver solutions are gathered to compose the solution of the complex task which is

[5] Since the considered problems present a NP complexity, a time limit is established when solving the CSPs.

returned by the module. Part of this module is written in Java while other parts are written in different solver languages. Currently, ILOG CPlex [11] is used as a solver.

5. Empirical Evaluation

This section describes the evaluation which was performed to assess the suitability of the proposed constraint-based tool to deal with the considered problems, i.e., generating traces and check their conformance.

Objects: Different TConDec-R models with different complexities are generated. Figure 5 shows the TConDec-R representation of the generic models 10A, 10B, 10C, 20A, 20B, and 20C. There are some activities that are involved in an Existence constraint, which means that such activities must be repeated several times. We have considered 15, 30 and 60 repetitions, i.e., $N \epsilon \{15,30,60\}$. Regarding the number of available resources, in turn, for all the generated test models, two available resources of two kinds of roles (i.e., R1 and R2) are considered. Moreover, random durations and resource requirements are considered for each activity. In addition, the activity relations which are specified in the generic models are randomly instantiated using TConDec-R templates, i.e., *Relation* and *Temporal* are substituted by any non-temporal and temporal TConDec-R template respectively. Specifically, in order to average the results over a collection of randomly generated TConDec-R models, 30 test models are randomly generated for each generic TConDec-R model by varying activity durations between 1 and 10, role of required resources between R1 and R2, and TConDec-R templates.[6] In summary, 6*3*30 = 540 different synthetic TConDec-R models are considered.

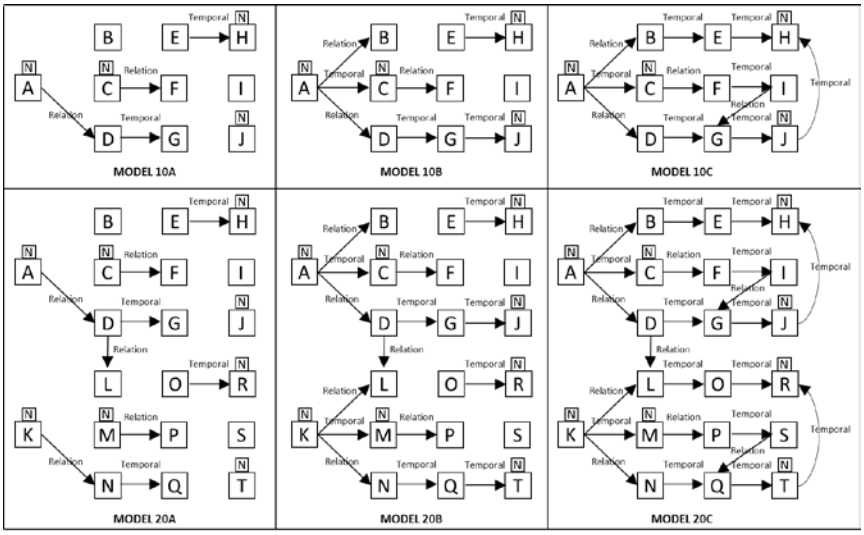

Fig. 5. Generic synthetic TConDec-R models

Independent Variables: For the empirical evaluation *M* (i.e., the generic TConDec-R model with the values *{M10A,M10B,M10C,M20A,M20B,M20C}*) and *N* (i.e., the value for the label N of the Existence constraints in the models with the values *{15,30,60}*) are considered as independent variables.

Response Variables: The evaluation is done regarding: (1) the percentage of models which are unknown, i.e., those whose related CSP has not been solved in the given time limit (i.e., *%NonSol*), (2) the percentage of models which are incorrect, i.e., those whose related CSP is not satisfiable [7] (i.e., *%NotSat*), (3) the average time for checking the correctness of models when they are not satisfiable (i.e., *TNotSat*), (4) the average time for checking the correctness of models when they are satisfiable (i.e., *TSat*), (5) the average time for the checking the

[6] The set of objects which are used are available at: http:\\azarias.lsi.us.es\TCR\EmpiricalObjects.zip

[7] The satisfiability of the models can be only ensured if the CSP solver explores the complete search space before reaching the time limit.

conformance of short traces (i.e., *TShort*), (6) the average time for the checking the conformance of medium traces (i.e., *TMed*), and (7) the average time for the checking the conformance of long traces (i.e., *TLong*).[8]

Experimental Design: 540 models are generated by considering different values for M (6 values), N (3 values) and the random generation of durations, required resources and templates (30 models). For each model, a solver which is created following the transformations described in Sect. 3 is executed until a time limit is reached. If the solver finishes before finding a solution (i.e., the model is unknown), the response variable *%NonSol* is collected. In case that it explores the complete search space and no solution is found (i.e., the model is incorrect), the response variables *TNotSat* and *%NotSat* are collected. In the contrary, when a solution is found, the response variables *TSat* and the solution of the solver (i.e., a complete trace) are collected. After that, in the case of correct models, the solver is then used to check the conformance of a trace which is created using the 25%, 50% and 75% of the solution collected previously and then, the response variables *TShort*, *TMed*, and *TLong* are collected. Finally, the final values of all the response variables consider the average of the 30 models.

Table 2. Average values related to the experimental executions (5-seconds time limit).

M	N	%NonSol	%NotSat	TNotSat	TSat	TShort	TMed	TLong
M10A	15	0%	26,7%	0,01	0,41	0,14	0,14	0,09
	30	0%	33,3%	0,02	1,08	0,50	0,42	0,23
	60	3,3%	30,0%	0,03	2,16	1,62	0,82	0,61
M10B	15	0%	50,0%	0,07	0,74	0,42	0,24	0,17
	30	0%	63,3%	0,08	1,55	0,89	0,36	0,22
	60	26,7%	60,0%	0,18	3,76	2,67	1,26	0,65
M10C	15	0%	56,7%	0,01	0,22	0,19	0,13	0,09
	30	0%	86,7%	0,02	0,33	0,23	0,18	0,08
	60	0%	70,0%	0,03	1,57	0,87	0,40	0,24
M20A	15	0%	86,7%	0,01	0,35	0,21	0,12	0,07
	30	0%	90,0%	0,02	1,55	0,55	0,28	0,20
	60	0%	90,0%	0,08	2,62	0,67	0,37	0,25
M20B	15	0%	70,0%	0,01	0,24	0,13	0,07	0,04
	30	0%	76,7%	0,06	0,55	0,23	0,18	0,07
	60	0%	86,7%	0,07	0,89	0,69	0,29	0,15
M20C	15	0%	93,3%	0,01	0,39	0,21	0,13	0,06
	30	0%	93,3%	0,01	1,19	0,36	0,26	0,11
	60	0%	93,3%	0,13	2,54	0,76	0,46	0,29

Time is expressed in seconds.

Experimental Execution: The complete approach is run in an Intel® Xeon® CPU E5530, 2.4GHz, 32GB memory, running Windows Server 2012.The ILOG CPLEX System [11] is used to solve the created constraint-based problems. The solver is run until a 5-seconds CPU time is reached.

Experimental Results and Data Analysis: Table 2 shows for each problem (i.e., *M* and *N*), the average values of the response variables. As can be seen, the approach is able to deal with most of the models in the given time limit (cf. column *%NonSol*). However, there are some cases (cf. M10A and M10B with N=60) where the related CSPs result too complex to be solved and thus, more time would be need to conclude if such models are correct or not. As expected, the percentage of incorrect models (cf. column *%NotSat*) that are created increases with the complexity of the problems since there are more chances of a constraint being in conflict with others. This is because models are generated randomly. For instance, for the simplest models

[8] In this context, short, medium and long traces consist of traces with the initial 25%, 50% and 75% of the events of a full trace respectively.

(i.e., *M10A*), there are less than 30% of incorrect models, however, more than 90% are generated for the most complex ones. It is important to note that the approach is able to detect that such models are incorrect in less than one second in all the cases (cf. column *TNotSat*). In turn, as depicted in column *TSat*, generating a complete trace (i.e., checking the correctness) of a correct model is more time consuming than of incorrect one. Nonetheless, it remains below 3 seconds in most of the cases. As expected, checking the conformance of traces is faster than generating new traces since the complexity of the CSP decreases as the size of the trace increases (cf. columns *TShort*, *TMed* and *TLong*). In general, the results show that providing support for declarative models is rather fast using the current proposal.

6. Conclusion

In the current work, we build upon a declarative business process modeling language which allows specifying sophisticated temporal constraints, i.e., the TConDec-R language [3]. Although there exists related work on declarative BP modeling [7, 18, 20, 24], only few approaches pay attention to the temporal perspective from a wider point of view. Unlike TConDec-R, existing works do not consider other requirements such as the support of constraints that may refer to a calendar or schedule, and time-based constraints.

Taking the TConDec-R language [3] as basis, this paper is focused on the definition of TConDec-R process models based on high-level elements from the constraint programming paradigm. Such a definition allows applying a variety of CP algorithms in order to provide support for the TConDec-R models, e.g., checking the correctness of a model or generating execution traces.

For providing a validation of the approach, a constraint-based tool has been described and implemented to support the TConDec-R models.[9] Such a tool allows (1) modeling scenarios through the TConDec-R language, (2) checking if the scenarios are correctly modeled, i.e., they can be instantiated, (3) generating valid execution traces according to such models, and (4) checking the conformance of execution traces.

In addition, to demonstrate the suitability of the presented approach, a set of synthetic examples of a variety of complexities are considered in an empirical evaluation.

We strongly believe that the proposed approach can be successfully applied in many sophisticated scenarios for enabling flexible process support. This is faced by integrating the high-level abstraction of BP in the CP context and contributes on improving the maturity of the declarative technology.

As future work, we will investigate the use and validation of constraint-based algorithms to improve the support to TConDec-R in several respects, e.g., to provide personal schedules or generate time predictions. In addition, we will further extend the proposed approach by considering the data perspective of business process as well. Furthermore, it is planned to extend the evaluation by considering real scenarios.

References

1. Global Constraint Catalog. http://sofdem.github.io/gccat/. Accessed April 22, 2018
2. C. Arevalo, M.J. Escalona, I. Ramos, and M. Domínguez-Muñoz. A metamodel to integrate business processes time perspective in bpmn 2.0. Information and Software Technology, 77:17–33, 2016.
3. I. Barba, A. Lanz, B. Weber, M. Reichert, and C. Del Valle. Optimized time management for declarative workflows. In Enterprise, Business-Process and Information Systems Modeling, volume 113 of LNBIP, pages 195–210. Springer Berlin Heidelberg, 2012.

[9] It is available at http://azarias.lsi.us.es/TCR/ModelChecker

4. I. Barba, B. Weber, C. Del Valle, and A. Jimenez-Ramirez. User recommendations for the optimized execution of business processes. Data & Knowledge Engineering, 86(0):61 – 84, 2013.
5. D. Borrego and I. Barba. Conformance checking and diagnosis for declarative business process models in data-aware scenarios. Expert Systems with Applications, 41(11):5340–5352, 2014.
6. Business Process Model and Notation (BPMN), Version 2.0. urlhttp:// www.omg.org/spec/BPMN/2.0/, 2011. Accessed April 22, 2018
7. A. Burattin, F.M. Maggi, and A. Sperduti. Conformance checking based on multiperspective declarative process models. Expert Systems with Applications, 65:194–211, 2016.
8. P. Dadam, M. Reichert, and K. Kuhn. Clinical workflows - the killer application for process-oriented information systems? In BIS 2000, pages 36–59. Springer, 2000.
9. Johann Eder, Euthimios Panagos, and Michael Rabinovich. Time constraints in workflow systems. In Advanced Information Systems Engineering, volume 1626 of LNCS, pages 286–300. 1999.
10. T. Hildebrandt, R.R. Mukkamala, T. Slaats, and F. Zanitti. Contracts for crossorganizational workflows as timed dynamic condition response graphs. The Journal of Logic and Algebraic Programming, 82(5):164–185, 2013.
11. IBM. CPLEX CP Optimizer. http://www-01.ibm.com/software/commerce/ optimization/cplex-cp-optimizer, 2016. Accessed April 22, 2018
12. Y. Jiang, N. Xiao, Y. Zhang, and L. Zhang. A novel flexible activity refinement approach for improving workflow process flexibility. Computers in Industry, 80:1–15, 2016.
13. A. Jimenez-Ramirez, I. Barba, Weber, B., and C. Del Valle. Generating optimized configurable business process models in scenarios subject to uncertainty. Information Software Technology, 57:571–594, 2015.
14. D. Knuplesch, M. Reichert, and A. Kumar. A framework for visually monitoring business process compliance. Information Systems, 64:381–409, 2017.
15. A. Lanz, M. Reichert, and B. Weber. Process time patterns: A formal foundation. Information Systems, 57:38–68, 2016.
16. A. Lanz, B. Weber, and M. Reichert. Time patterns for process-aware information systems. Requirements Engineering, 2012.
17. Z. Liu, S. Fan, H.J. Wang, and J.L. Zhao. Enabling effective workflow model reuse: A data-centric approach. Decision Support Systems, 93:11–25, 2017.
18. F.M. Maggi and M. Westergaard. Using timed automata for a Priori warnings and planning for timed declarative process models. International Journal of Cooperative Information Systems, 23(1), 2014.
19. M. Montali. Specification and Verification of Declarative Open Interaction Models: a Logic-Based Approach. PhD thesis, Department of Electronics, Computer Science and Telecommunications Engineering, University of Bologna, 2009.
20. M. Montali, F.M. Maggi, F. Chesani, P. Mello, and W.M.P. van der Aalst. Monitoring business constraints with the event calculus. ACM TIST, 5(1):17, 2013.
21. M. Pesic. Constraint-Based Workflow Management Systems: Shifting Control to Users. PhD thesis, Eindhoven University of Technology, Eindhoven, 2008.
22. J.C. Régin. Global constraints: A survey. In Hybrid optimization, pages 63–134. Springer, 2011.
23. F. Rossi, P. van Beek, and T. Walsh, editors. Handbook of Constraint Programming. Elsevier, 2006.
24. W.M.P. van der Aalst, M. Pesic, and M.H. Schonenberg. Declarative workflows: Balancing between flexibility and support. Computer Science - Research and Development, 23(2):99–113, 2009.
25. M. Weske. Business Process Management: Concepts, Languages, Architectures. Springer, 2007.