AUTOMATIC GENERATION OF OPTIMIZED BUSINESS PROCESS MODELS FROM CONSTRAINT-BASED SPECIFICATIONS

IRENE BARBA* and CARMELO DEL VALLE†

Dpto. Lenguajes y Sistemas Informáticos
University of Seville, Avda Reina Mercedes s/n
Seville, 41012, Spain

**irenebr@us.es*

†*carmelo@us.es*

BARBARA WEBER

Department of Computer Science
University of Innsbruck, Technikerstraße 21a
Innsbruck, 6020, Austria
barbara.weber@uibk.ac.at

ANDRÉS JIMÉNEZ

Dpto. Lenguajes y Sistemas Informáticos
University of Seville, Avda Reina Mercedes s/n
Seville, 41012, Spain
ajramirez@us.es

Business process (BP) models are usually defined manually by business analysts through imperative languages considering activity properties, constraints imposed on the rela-tions between the activities as well as different performance objectives. Furthermore, allocating resources is an additional challenge since scheduling may significantly impact BP performance. Therefore, the manual specification of BP models can be very com-plex and time-consuming, potentially leading to non-optimized models or even errors. To overcome these problems, this work proposes the automatic generation of imperative optimized BP models from declarative specifications. The static part of these declara-tive specifications (i.e. control-flow and resource constraints) is expected to be useful on a long-term basis. This static part is complemented with information that is less stable and which is potentially unknown until starting the BP execution, i.e. estimates related to (1) number of process instances which are being executed within a particular timeframe, (2) activity durations, and (3) resource availabilities. Unlike conventional proposals, an imperative BP model optimizing a set of instances is created and deployed on a short-term basis. To provide for run-time flexibility the proposed approach addition-ally allows decisions to be deferred to run-time by using complex late-planning activities, and the imperative BP model to be dynamically adapted during run-time using replan-ning. To validate the proposed approach, different performance measures for a set of

*Corresponding author.

test models of varying complexity are analyzed. The results indicate that, despite the NP-hard complexity of the problems, a satisfactory number of suitable solutions can be produced.

Keywords: Business process management; constraint programming; planning; scheduling.

1. Introduction

A business process (BP) consists of a set of activities which are performed in coordination in an organizational and technical environment,[1] and which jointly realize a business goal. Nowadays, there exists a growing interest in aligning information systems in a process-oriented way[1,2] as well as in the effective management of BPs. BP improvement has been ranked as the number one priority for top management by the 2010 Gartner survey.[3] BP management (BPM) can be seen as supporting BPs using methods, techniques, and software in order to design, enact, control, and analyze operational processes involving humans, organizations, applications, and other sources of information.[4] Typically, the traditional BPM life cycle[1] includes several phases: Process Design & Analysis, system configuration, process enactment and evaluation. The BP Design & Analysis phase has the goal to generate a BP model, i.e. to define the set of activities and the execution constraints between them,[1] by formalizing the informal BP description using a particular BP modeling notation. The Process Design & Analysis phase plays an important role in the BPM life cycle for any improvement initiative, since it greatly influences the remaining phases of this cycle. In addition, also run-time aspects are important for BP improvement, e.g. resource allocations and scheduling may significantly impact BP performance.

1.1. *Problem statement*

Traditionally, two steps are considered in the BP Design & Analysis phase to create a BP model.[5] The first step consists of analyzing the BP, e.g. by interviewing stakeholders (people involved in the process), in order to draw an initial BP model (as-is model). Second, in order to improve this initial model, different techniques can be employed like simulation[6] or BP redesign,[7] resulting in the generation of a to-be model. Typically, different quality dimensions like time, cost, flexibility and quality can be differentiated[7] between which trade-off decisions have to be made when creating a BP design. Once a certain process design has been chosen and implemented, BPs are executed according to this design.[a] During process execution, scheduling decisions are then typically made by the BPM systems (BPMSs), by automatically assigning activities to resources.[8]

[a]In this work, we make the assumption that there is a BPM system executing the BPs.

In most cases, the overall process of creating a BP model is carried out manually by business analysts, who specify the BP information through an imperative language by choosing between several different alternative designs the one which best meets the performance goals of the organization. Therefore, analysts must deal with several aspects in order to generate a suitable BP model, such as: (1) the activity properties, e.g. activity duration, resource or role which is required for activity execution (i.e. the role-based allocation pattern is considered[8]), (2) the relations between the activities, i.e. control-flow of the BP, and (3) the optimization of several objectives, e.g. minimization of completion time. The manual specification of imperative BP models can therefore form a very complex problem, i.e. it can consume a great quantity of time and human resources, may cause certain failures, and may lead to non-optimized models since the tacit nature of human knowledge is often an obstacle to eliciting accurate process models.[9]

Not only the process design, but also the allocation of resources during process execution has a great influence on process performance. However, scheduling is only considered to a limited degree in existing BPMSs, and is typically done during run-time by assigning work to resources.

The situation is further complicated by the fact that typically multiple instances of a process get concurrently executed within a particular timeframe. In order to ensure that the execution of a process is not only locally optimized for a single instance, the whole set of instances which are executed within a particular timeframe has to be considered.

1.2. *Contribution*

To support process analysts in the definition of optimized BP models we suggest a method for automatically generating imperative BP models using artificial intelligence (AI) planning techniques from constraint-based specifications. Unlike imperative models, the specification of process properties in a declarative way, e.g. using a constraint-based specification, only requires process designers to state what has to be done instead of having to specify how it has to be done.[b] In the proposed approach, the static part of the input declarative model (i.e. control-flow and resource constraints) is expected to be useful on a long-term basis since it embraces information which is not supposed to change often. The base declarative model (i.e. only including the static part) is complemented with information that is less stable and which is potentially unknown until starting the BP execution, i.e. estimates related to (1) number of process instances which are being executed within a particular timeframe, (2) activity durations, and (3) resource availabilities. From this extended model, the proposed approach is in charge of determining how

[b]The advantages of using declarative languages for BP modeling instead of imperative languages, e.g. facilitating the human work involved in the BP modeling, are discussed in several studies, e.g. Refs. 10–15.

to satisfy the constraints imposed by the declarative specification and at the same time to attain an optimization of certain objective functions (e.g. minimization of completion time). For this optimization, scheduling is done on a short-term basis by considering the optimization of a set of instances.

Unlike conventional proposals, in our approach each generated model is created and deployed for a specific planning period, considering changing information such as the number of process instances which are being executed within a specific timeframe. For the next executions of the declarative model, new models will be generated considering the specific values which are given for the changing information. Since planning is done on a short-term basis, the generated models are less prone to change.

Figure 1 provides an overview of our approach. Taking the constraint-based specifications as a starting point (cf. Fig. 1(1)), enactment plans can automatically be generated (cf. Fig. 1(2)). For this, activities to be executed have to be selected and ordered (planning problem[16]) considering the control-flow imposed by the constraint-based specification. Moreover, to automatically propose execution plans which meet the performance goals best (e.g. minimizing the overall completion time (OCT), i.e. time needed to complete all process instances which were planned for a certain period), the constraint-based model is complemented with information related to estimates regarding the number of instances, activity durations, and resource availabilities (scheduling problem[17]). For planning and scheduling (P&S) the activities such that the process objective function is optimized, a constraint-based approach is proposed since constraint programming[18] supplies a suitable framework for modeling and solving problems involving P&S aspects.[19] The generated enactment plans are then automatically translated into a Business Process Model and Notation (BPMN) model[20] (cf. Fig. 1(3)), which can be then further improved by a business analyst, where necessary. In most cases, BPMN models can be translated into an execution language,[21] such as BPEL,[22] which enables BP designs to be deployed into BPMS and let their instances be executed by a BPM engine. To provide for an increased flexibility the BPMN model can be dynamically adapted during run-time by using replanning (cf. Fig. 1(4)).

Note that the BPMN model is generated with the goal of making the declarative model automatically executable by a BPMS by considering the specific values of the changing information which are given just before starting the execution the

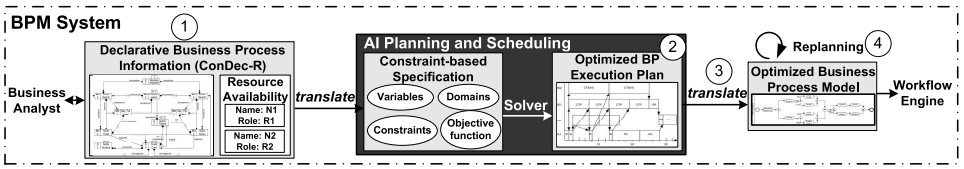


Fig. 1. AI P&S techniques for the generation of optimized BP models.

process. In this way, application of decision deferral patterns is automated,[23] i.e. the role of the BPMS is rather focused on enabling control and ensuring compliance (decisions are automatically made by the BPMS). Regarding decision deferral patterns, our approach belongs to the late modeling and composition pattern, i.e. allowing for modeling and automatic composition of a process model just before starting the execution of a branch of process instances. Therefore, our approach can be framed within dynamic process-based composition (i.e. completely creating the executable process model dynamically at run-time), which constitutes an example of the automated variant of the late modeling and composition pattern.

The main contributions of this paper can be summarized as follows:

(i) The definition of a language for the constraint-based specification of BPs which extends ConDec,[11,24] named ConDec-R (cf. Sec. 3, Step 1 in Fig. 1), to enable the reasoning about resources.
(ii) Automatic planning and scheduling of the BP activities for the generation of optimized BP enactment plans from the ConDec-R specifications, through a constraint-based approach (cf. Sec. 4, Step 2 in Fig. 1).
(iii) Automatic generation of optimized BP models in BPMN from optimized BP enactment plans (cf. Sec. 5, Step 3 in Fig. 1).
(iv) Providing for run-time flexibility by allowing decisions to be deferred at run-time and the BPMN model to be dynamically adapted during run-time (cf. Sec. 6, Step 4 in Fig. 1).
(v) Validation of the proposed approach through the analysis of different performance measures related to a range of test models of varying complexity (cf. Sec. 8).

In this way, the automatic generation of BP models simplifies the BP design phase by facilitating the human work in most cases, preventing failures in the developed BP models, and enabling better optimization to be attained in the enactment phase. Furthermore, imperative BP models can dynamically be generated from static constraint-based specifications just before starting the BP enactment, once some values for the enactment parameters, e.g. resource availabilities, are known. Moreover, the automatic generation of BP models can deal with complex problems of great size in a simple way (as will be demonstrated in Sec. 8). Therefore, a wide study of several aspects can be carried out, such as those related to the requirement of resources of different roles, or the estimated completion time for the BP enactment, by generating several kinds of alternative specifications. In addition, in order to address run-time flexibility the proposed approach allows decisions to be deferred at run-time by using complex late-planning activities, and the BPMN model to be dynamically adapted during run-time using replanning.

The remainder of this paper is organized as follows: Sec. 2 introduces backgrounds needed for the further understanding of the paper, Secs. 3–6 detail the proposals of this work, Sec. 7 explains an example, Sec. 8 deals with the evaluation

of the proposed approach, Sec. 9 summarizes related work, Sec. 10 presents a critical discussion of the advantages and limitations of our proposal, and finally, Sec. 11 includes some conclusions and future work.

2. Background

Our work combines aspects of scheduling, planning, and constraint programming in order to automatically generate optimized BP enactment plans from constraint-based specifications. The optimized enactment plans are then translated into BPMN models. Section 2.1 provides backgrounds regarding constraint-based processes. Section 2.2 gives an overview of planning, scheduling, and constraint programming. Section 2.3 summarizes the BPMN standard.

2.1. *Constraint-based BP models*

Different paradigms for process modeling exist, e.g. imperative and declarative. Irrespective of the chosen approach, desired behavior must be supported by the process model, while forbidden behavior must be prohibited.[11,25,26] While imperative process models specify exactly *how* things have to be done, declarative process models focus on *what* should be done. In literature, several rule-based and constraint-based languages for declarative BP modeling are proposed (e.g. Refs. 11, 24, 27–29). In our proposal we use ConDec[11,24] for the BP control-flow specification. We consider ConDec to be a suitable language, since it allows the specification of BP activities together with the constraints which must be satisfied for correct BP enactment and for the goal to be achieved. Moreover, ConDec allows to specify a wide set of BP models in a simple and flexible way. ConDec is based on constraint-based BP models (cf. Definition 2.1), i.e. including information about (1) activities that can be performed as well as (2) constraints prohibiting undesired process behavior.

Definition 2.1. A *constraint-based process model* $S = (A, C_{\mathrm{BP}})$ consists of a set of activities A, and a set of constraints C_{BP} prohibiting undesired execution behavior. For each activity $a \in A$ resource constraints can be specified by associating a role with that activity. The activities of a constraint-based process model can be executed arbitrarily often if not restricted by any constraints.

Constraints can be added to a ConDec model to specify forbidden behavior, restricting the desired behavior. For this, ConDec proposes an open set of templates, i.e. parametrized graphical representations of constraints over the BP activities, which can be divided into three groups (for a description of the complete set of templates, cf. Ref. 30):

(i) *Existence* templates: Unary relationships concerning the number of times one activity is executed. As an example, Exactly(N, A) specifies that A must be executed exactly N times.

(ii) Relation templates: Positive binary relationships used to establish what should be executed. As an example, Precedence(A, B) specifies that to execute activity B, activity A needs to be executed before.

(iii) Negation templates: Negative binary relationships used to forbid the execution of activities in specific situations. As an example, NotCoexistence(A, B) specifies that if B is executed, then A cannot be executed, and vice versa.

While unary relationships describe constraints related to one activity (e.g. existence constraints), binary constraints describe relationships between activities (e.g. precedence constraints). Binary templates are composed by a source activity (cf. Definition 2.2) and a sink activity (cf. Definition 2.3).

Definition 2.2. A source activity of a binary template is an activity which appears in the first parameter of the template. For templates which state precedence relations between activities, a source activity is a predecessor activity.

Definition 2.3. A sink activity of a binary template is an activity which appears in the second parameter of the template. For templates which state precedence relations between activities, a sink activity is a successor activity.

Figure 2 shows a simple constraint-based model which is composed by activities A, B, and C, and constraints $C1$ (Exactly(2, A)), $C2$ (Precedence(A, B)), $C3$ (Precedence(A, C)), and $C4$ (NotCoexistence(B, C)).

Furthermore, binary templates can be extended by defining branched templates, as described in Ref. 26. The branched templates for the binary templates can be established between several BP activities in the following way:

- The branched template is established between several source activities and one sink activity, so that the relation is given between `at least` one of the sources and the sink.

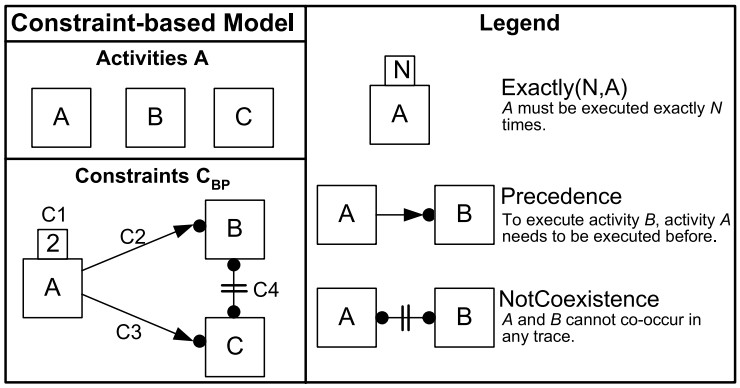


Fig. 2. Simple constraint-based model.

- The branched template is established between one source activity and several sink activities, so that the relation is given between the source and `at least` one of the sinks.

As the execution of a constraint-based model proceeds, information regarding the executed activities is recorded in an execution trace (cf. Definition 2.4).

Definition 2.4. Let $S = (A, C_{\mathrm{BP}})$ be a constraint-based process model with activity set A and constraint set C_{BP}. Then: A **trace** σ is composed by a sequence of starting and completing events $\langle e_1, e_2, \ldots, e_n \rangle$ regarding activity executions a_i, $a \in A$, i.e. events can be:

(i) $start(a_i, r_{jk}, t)$, i.e. the i-th execution of activity a using k-th resource with role j is started at time t.
(ii) $comp(a_i, t)$, i.e. the i-th execution of activity a is completed at time t.

Due to their flexible nature, frequently several ways to execute constraint-based process models exist, i.e. everything which is enabled can be executed. The decisions related to which of the enabled activities to execute and in what order can be made using several mechanisms[23]:

- Goal-based: Decisions between alternatives (selection of a particular process fragment, actor, or activity implementation) are made considering the overall goals (cf. Definition 2.5) of the process.

Definition 2.5. The *goal* of a BP is specified through the constraints which must be satisfied in the BP enactment.

For example, traces[c] $\langle AAB \rangle$ and $\langle AAC \rangle$ are two valid ways of executing the constraint-based model of Fig. 2, while trace $\langle AABC \rangle$ is invalid due to $C4$. The different valid execution alternatives, however, can vary greatly in respect to their quality, i.e. how well different performance objective functions (cf. Definition 2.6) like minimizing cycle time can be achieved. Optimization decisions can be framed as goal-based decisions (i.e. there alternatives to reach a specific goal and the optimization of given objective functions should be considered to select one of these alternatives).

Definition 2.6. The *objective function* of a BP is the function to be optimized during the BP enactment, e.g. minimization of OCT.

- Rule-based: Decisions between different alternatives are made based on a set of rules.

[c] For the sake of clarity, traces represent sequences of activities so that no parallelism is considered in the examples. Moreover, only completed events for activity executions are included in the trace representation.

- Experienced-based: Decisions between different alternatives are made by relying on past experiences made in similar context.
- User-based: Decisions between different alternatives are made by leaving decision making to the human expert.

In our base proposal, we consider all the choice-like constructs of ConDec as optimization or goal decisions, i.e. decision making is goal-based. Assuming that all decisions are goal-based, the base approach might be valid for some environments like for example certain web service settings. If more flexibility is required, in Sec. 6.1 we propose an approach that allows decisions which are not goal-based to be deferred to run-time and enables the optimization within decision fragments which frame non-goal-based decisions. Moreover, to provide for an increased run-time flexibility, there is the possibility to use replanning (cf. Sec. 6.2).

2.2. *Planning, scheduling and constraint programming*

For generating optimized BP enactment plans optimizing the performance objective functions (cf. Definition 2.6) of constraint-based process models, activities to be executed have to be planned[16] and scheduled[17] by considering the constraint-based specification. To do this, a constraint-based approach is proposed.[19]

The area of scheduling[17] includes problems in which it is necessary to determine an enactment plan for a set of activities related by temporal constraints (in our context the control-flow constraints together with the resource constraints, i.e. required resources, introduced in Sec. 2.1). Moreover, since the execution of activities may require the same resources, they may compete for limited resources. In general, the objective in scheduling is to find a feasible plan which satisfies both temporal and resource constraints. Several objective functions are usually considered to be optimized, in most cases related to temporal measures (e.g. minimization of completion time), or considering the optimal use of resources.

In a wider perspective, in AI planning,[16] the activities to be executed are not established *a priori*, hence it is necessary to select them from a set of alternatives and to establish an ordering. In most cases, the specification of planning problems includes the initial state of the world, the goal (a predicate representing a set of possible final states) that must be reached, and a set of operators (actions) which can be applied to one state in order to reach another state. Furthermore, in planning problems, usually the optimization of certain objective functions is considered.

Constraint programming (CP)[18] (cf. Fig. 3) can be used, among others, for planning and scheduling purposes.[19] In order to solve a problem through constraint programming, it needs to be modeled as a constraint satisfaction problem (CSP) (cf. Definition 2.7).

Definition 2.7. A *CSP* $P = (V, D, C_{\text{CSP}})$ is composed by a set of variables V, a set of domains D, which is composed of the domain of values dom_i for each variable $var_i \in V$, and a set of constraints C_{CSP} between variables, so that each

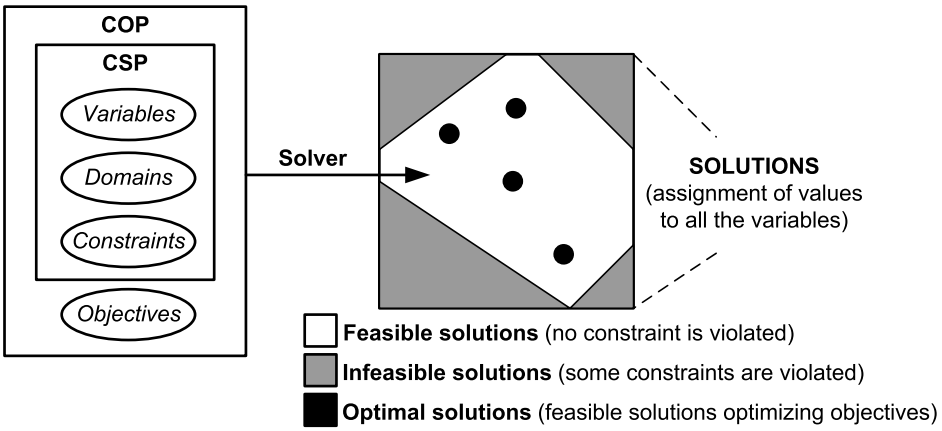


Fig. 3. Constraint programming.

constraint represents a relation between a subset of variables and specifies the allowed combinations of values for these variables.

A solution to a CSP (cf. Definition 2.8) consists of assigning values to CSP variables.

Definition 2.8. *A solution* $S = \langle (var_1, val_1), (var_2, val_2), \ldots, (var_n, val_n) \rangle$ for a CSP $P = (V, D, C_{\text{CSP}})$ is an assignment of a value $val_i \in dom_i$ to each variable $var_i \in V$. A solution is *partial* if one or more CSP variables exist which are not instantiated. A solution is *feasible* when the assignments variable-value satisfy all the constraints, i.e. if a goal state is reached.

In a similar way, a CSP is feasible if at least one feasible solution for this CSP exists. From now on, S^{var} refers to the value assigned to variable var in the (partial) solution S.

Similar to CSPs, constraint optimization problems (COPs, cf. Definition 2.9) require solutions that optimize objective functions.

Definition 2.9. A *COP* $P_o = (V, D, C_{\text{CSP}}, o)$ is a CSP which also includes an objective function o to be optimized.

A feasible solution S for a COP is *optimal* when no other feasible solution exists with a better value for the objective function o.

Constraint programming allows to separate the models from the algorithms, so that once a problem is modeled 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. Several mechanisms are available for solving CSPs and COPs,[18] which can be classified as *search algorithms* (i.e. for exploring the solution space to

find a solution or to prove that none exists) or *consistency algorithms* (i.e. filtering rules for removing inconsistent values from the domain of the variables). In turn, search algorithms can be classified as *complete search algorithms* (i.e. performing a complete exploration of a search space which is based on all possible combinations of assignments of values to the CSP variables), and *incomplete search algorithms* (i.e. performing an incomplete exploration of the search space so that, in general, to get a feasible or an optimal solution is not guaranteed).

2.3. *Business process model and notation*

The optimized enactment plans generated using constraint programming are then translated to optimized imperative models (cf. Sec. 5), i.e. BPMN models which can be translated into an execution language,[21] such as BPEL,[22] which enables BP designs to be deployed into BPM systems and let their instances be executed by a BPM engine. The BP model and notation (BPMN)[20] is a standard for modeling BP flows and web services, and provides a graphical notation for the specification of BP models. A BPMN model is composed of events, gateways, activities and swim lanes, among other elements. An event represents something that happens during the enactment of a BP and affects its execution flow, specifically the start event initiates the flow of the process, while the end event finishes this flow. Gateways are in charge of controlling how sequence flows interact as they converge or diverge within a process. Specifically, the *exclusive data-based gateway* can either be used as a decision point where several outgoing sequence flows are possible, but only one sequence will be selected for the execution, or as a way to merge several sequence flows into one; the *parallel gateway*, in turn, provides a mechanism to both fork and synchronize the flows. Swim lanes are graphical ways of organizing and categorizing the BP activities, whereby pools represent the participants in a BP, and lanes are used to organize the activities within a pool according to roles and resources.

3. ConDec-R: Constraint-Based Specification of Business Processes

In order to plan and schedule the BP activities (cf. Sec. 2.2), ConDec is used as a basis. To make ConDec usable for our concrete application an extension of ConDec (named ConDec-R) is defined in this work, as detailed in Secs. 3.1 and 3.2. ConDec-R supports all templates described in Ref. 30 and additionally provides extended support for branched templates, as described in Ref. 26. The ConDec-R templates, extending the ConDec templates,[30] are listed in Ref. 67.

3.1. *Extending ConDec with estimates and resource availabilities*

To support the direct reasoning of resources (which is not possible in ConDec) we extended ConDec with estimates of activity durations and the specification of

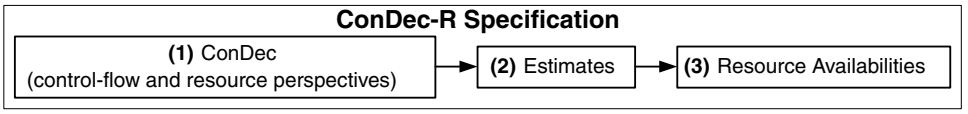


Fig. 4. ConDec-R process model specification.

resource availabilities. In short, a ConDec-R process model specification (cf. Definition 3.1, Fig. 4) extends a ConDec model (cf. Fig. 4(1)) as specified in Definition 2.1 by including:

- The *estimated duration* of each activity (cf. Fig. 4(2)). Estimates can be obtained by interviewing business experts or by analyzing past process executions (e.g. by calculating the average values of the parameters to be estimated from event logs). Moreover, both approaches can be combined to get more reliable estimates.
- The *available resources* of each role (cf. Fig. 4(3)). This information is independent of the ConDec-R activities, and hence it can be changed without affecting the specification of the activities, and vice versa.

Notice that the resource availabilities can be unknown until starting BP enactment. This is not a problem for our proposal since the most static information, i.e. the control-flow and resource constraints (cf. Fig. 4(1)), is complemented with more changing information, i.e. the estimates (cf. Fig. 4(2)), and finally the most dynamic data, i.e. information on resource availabilities (cf. Fig. 4(3)), is included. In this way, the imperative BP model can be generated just before starting the BP enactment by considering the actual values of the resource availabilities (Secs. 4 and 5).

Definition 3.1. A *ConDec-R process model* $CR = (Acts, C_{\text{BP}}, Res)$ related to a constraint-based process model $S = (A, C_{\text{BP}})$ is composed by a set of BP activities $Acts$, which contains tuples $(a, role, dur)$ which includes for each BP activity $a \in A$ the role of the required resource (i.e. $role$) and the estimated duration (i.e. dur), i.e. $Acts \subseteq A \times Roles \times \mathbb{N}$, being $Roles$ the set of names of all the considered roles; a set of ConDec constraints C_{BP}; and a set of available resources Res, composed by tuples $(role, \#role)$ which includes for each role (i.e. $role$) $\#role$ available resources.

An example of a ConDec-R process model is depicted in Fig. 5(1), where $Acts = \{(A, R_1, 5), (B, R_2, 3), (C, R_1, 4)\}$, $C_{\text{BP}} = \{Exactly(2, A),\ Precedence(A, B), Precedence(A, C),\ NotCoexistence(B, C)\}$, and $Res = \{(R_1, 2),\ (R_2, 1)\}$.

3.2. *Extending ConDec with parallel execution of activities*

In ConDec no parallelism is considered in the execution of activities which are related by ordering constraints since ConDec activities are atomic. In this work, non-atomic activities, i.e. durative activities, are considered, and hence the ConDec

templates are extended so that the relations which are stated in Allen's interval algebra[33] are allowed in order to deal with temporal reasoning. In ConDec-R, the relation *activity b must be executed after a* can imply four different meanings:

- $st(b) \geq et(a)$ (default option)
- $st(b) \geq st(a)$
- $et(b) \geq et(a)$
- $et(b) \geq st(a)$

In this way, in ConDec-R some of the ConDec templates[11,24] are adapted and extended by considering the possible parallelism in the execution of those activities that are related by ordering constraints. This leads to four variants for the same temporal relation between two activities a and b, which is represented by an additional label at the end of the template name. This label represents: first, the time related to a which is constrained (start, S, or end, E), and the time related to b which is constrained (start, S, or end, E) through the inclusion of the template.[d] Therefore, the four variants for the same template are: SS, ES, SE, EE. Some examples of this kind of extension can be seen in Sec. 7.

4. From ConDec-R to Optimized Enactment Plans

In this section, the complete process which is proposed to generate BP enactment plans from a ConDec-R specification through CP is detailed (cf. Fig. 5). As stated, BP activities and constraints are specified in a ConDec-R model (cf. Step 1 in

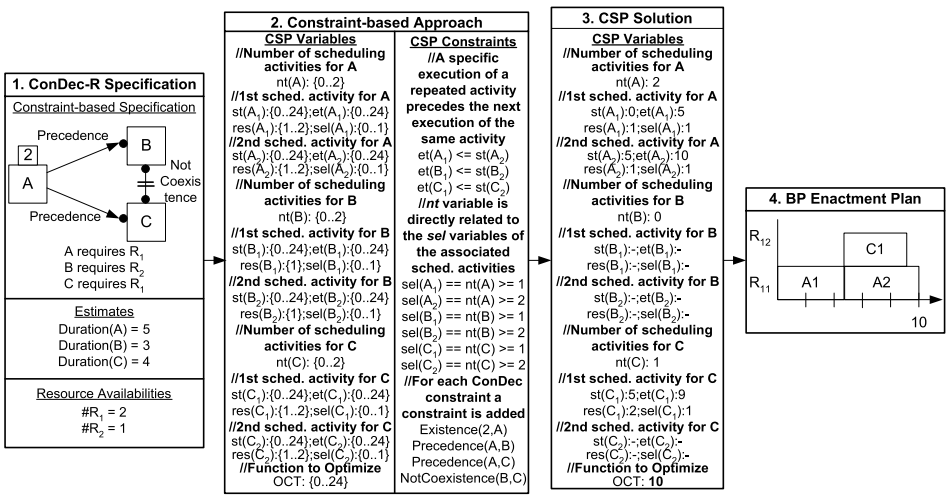


Fig. 5. From ConDec-R specification to BP enactment plan.

[d]In a similar way, ConDec++[26] also considers constraints imposed on the start and the completion times of non-atomic activities.

Fig. 5, Sec. 3) so that frequently several feasible ways to execute this model exist. Each specific feasible execution of a ConDec-R model leads to a specific value for the function to optimize. In general, even multiple optimal executions, i.e. feasible solutions leading to a minimal completion time,[e] may exist. In order to generate optimal (or optimized) execution plans for a specific ConDec-R model, we propose a constraint-based approach for P&S the BP activities (cf. Step 2 in Fig. 5). The obtained plans, i.e. solutions to the COP (cf. Step 3 in Fig. 5), optimize the specified objective function (cf. Definition 2.6) and satisfy all the constraints which are stated in the specification of the problem, i.e. reaches the specified goal (cf. Definition 2.5). Lastly, the generated plans are visualized as Gantt chart (cf. Step 4 in Fig. 5) which illustrates the start and the end times of the activities together with the assigned resource.

4.1. *Representing the ConDec-R model as CSP model*

As first step of the constraint-based approach the Condec-R model needs to be represented as a CSP. Regarding the CSP model of the proposed constraint-based approach, BP activities (repeated activities, cf. Definition 4.1), which can be executed arbitrarily often if not restricted by any constraints, are modeled as a sequence of optional scheduling activities (cf. Definition 4.2). This is required since each execution of a BP activity is considered as one single activity which needs to be allocated to a specific resource and temporarily placed in the enactment plan, i.e. stating values for its start and end times.

Definition 4.1. A *repeated activity* $ra = (a, role, dur, nt)$ represents a ConDec-R activity $(a, role, dur) \in Acts$ of a ConDec-R process model $CR = (Acts, C_{\mathrm{BP}}, Res)$ (cf. Definition 3.1) which can be executed several times.[f] The properties of a repeated activity are: the role of the required resource for the execution of activity a,[g] role, the estimated duration,[h] dur, and a CSP variable nt which represents the number of times activity a can be executed (cf. Fig. 5(1)).

Lower and upper bounds are related to the domain of each integer CSP variable var, representing minimum and maximum values which can be given to var in a feasible solution, respectively. Thereby, $LB(var)$ and $UB(var)$ refer to the lower and upper bounds of the domain of var.

[e] In this paper, we consider the OCT as objective function. However, note that the proposal can be easily extended to support further objective functions like cost.

[f] From now on, $role(a)$, $dur(a)$, and $nt(a)$ refer to the properties of the repeated activity related to the ConDec-R activity $(a, role, dur)$.

[g] For sake of simplicity, the same required resource is considered for all the executions of a BP activity. Note that the proposed constraint-based approach can also deal with BP activities which require several resources of various kinds of roles, since ConDec-R problems are modeled as scheduling problems with optional activities, where the resources have a discrete capacity (cumulative scheduling). Therefore, optimized BP enactment plans with activities which require more than one role can be generated by using the proposed constraint-based approach.

[h] The same estimated duration is considered for all the executions of a BP activity.

Given a ConDec-R process model $CR = (Acts, C_{\text{BP}}, Res)$ (cf. Definition 3.1), the set of related repeated activities is composed by $\{(role, dur, nt), (a, role, dur) \in Acts\}$. Note that all the activities which are included in the ConDec-R model are modeled as repeated activities (cf. Definition 4.1) in the proposed constraint-based approach regardless of the number of times that the BP activities are going to be executed in valid enactment plans, i.e. regardless of the ConDec-R constraints. The repetitiveness of a repeated activity is determined by considering all the ConDec-R constraints of the model and the optimization of the objective function when generating the optimized BP enactment plans. For example, activities A, B and C of the constraint-based model of Fig. 5(1) are repeated activities (cf. Definition 4.1). Activity A, for example, requires resource of role R_1 and has an estimated duration of 5. Moreover, the number of activity executions nt for A is equal to 2.

For each repeated activity, nt scheduling activities exist, which are added to the CSP problem specification, apart from including a variable nt (cf. Fig. 5(2)). For example, for repeated activity A in Fig. 5 two scheduling activities exist (referred to as A_1 and A_2).

Definition 4.2. A *scheduling activity* $sa = (a, st, et, res, sel)$ represents a specific execution of a repeated activity $ra = (a, role, dur, nt)$ (cf. Definition 4.1) related to a ConDec-R process model $CR = (Acts, C_{\text{BP}}, Res)$ (cf. Definition 3.1), where st is a CSP variable indicating the start time of the activity execution, et is a CSP variable indicating the end time of the activity execution ($et - st = dur$) in a way that each execution of a BP activity is temporarily placed in the enactment plan, res is a CSP variable representing the resource used for the execution (identified by a number between 1 and $\#role$, $(role, \#role) \in Res$), and sel is a CSP variable indicating whether or not the scheduling activity is selected to be executed (cf. Fig. 5(2)).

Moreover, an additional CSP variable representing the objective function to be optimized (in the context of our proposal the overall completion time), named OCT, is also included in the CSP model: $\text{OCT} = \max_{(a,role,dur) \in Acts} (et(a_{nt(a)}))$.[i]

In addition, in order to ensure the consistency between the CSP variables several constraints have to be added to the CSP (cf. Fig. 5(2)): (1) $\forall i : 1 \leq i < UB(nt(a)) : et(a_i) \leq st(a_{i+1})$, i.e. a specific execution of a repeated activity precedes the next execution of the same activity, and (2) $\forall i : 1 \leq i \leq UB(nt(a)) : sel(a_i) == nt(a) >= i$, i.e. the nt variable of the repeated activity is directly related to the sel variables of the associated scheduling activities, e.g. if $nt(a) = 2$, then $sel(a_1) = 1$, $sel(a_2) = 1$ and $\forall i : nt(a) < i \leq UB(nt(a)), sel(a_i) = 0$. Moreover, for each ConDec template a global

[i] a_i refers to the scheduling activity related to the i-th execution of the repeated activity associated to the ConDec-R activity $(a, role, dur)$, while $st(a_i)$, $et(a_i)$, $res(a_i)$ and $sel(a_i)$ refer to its related properties.

constraint is added to the CSP model, i.e. $Existence(2, A)$, $Precedence(A, B)$, $Precedence(A, C)$, and $NotCoexistence(B, C)$ for the constraint-based model depicted in Fig. 5.

Definition 4.3. A *CSP-ConDec problem* related to a ConDec-R process model $CR = (Acts, C_{\text{BP}}, Res)$ (cf. Definition 3.1) is a COP $P_o = (V, D, C_{\text{CSP}}, o)$ (cf. Definition 2.9) where:

(i) The set of variables V is composed by all the CSP variables included in the presented CSP model plus the CSP variable related to the OCT, i.e. $V = \{nt(a), (a, role, dur) \in Acts\} \cup \{st(a_i), et(a_i), res(a_i), sel(a_i), (a, role, dur) \in Acts, i \in [1 \dots UB(nt(a))]\} \cup \{OCT\}$.

(ii) The set of domains D is composed by the domains of each CSP variable v, $Dom(v)$, i.e. $D = \{Dom(nt(a)) = \{0 \dots MC\}, (a, role, dur) \in Acts\} \cup \{Dom\ (st(a_i)) = Dom(et(a_i)) = \{0 \dots MC \times \sum_{(a,role,dur) \in Acts} dur(a)\}, (a, role, dur) \in Acts, i \in [1 \dots UB(nt(a))]\} \cup \{Dom(res(a_i)) = \{1 \dots \#role, (role, \#role) \in Res\}, (a, role, dur) \in Acts, i \in [1 \dots UB(nt(a))]\} \cup \{Dom\ (sel(a_i)) = \{0 \dots 1\}, (a, role, dur) \in Acts, i \in [1 \dots UB\ (nt(a))]\}$, where MC is the maximum cardinality for the BP activities, i.e. nt (established by existence relations in the constraint-based model). In this way, MC is used for establishing initial upper bounds (i.e. UB) for the domain of several variables (including nt variables).

(iii) The set of constraints C_{CSP} is composed by the global constraints (implemented by the filtering rules, cf. Sec. 4.2) related to the ConDec-R constraints included in C_{BP} together with the constraints from the proposed CSP model, i.e. $\forall\, i : 1 \leq i < nt(a) : et(a_i) \leq st(a_{i+1})$, $\forall\, i : 1 \leq i \leq UB(nt(a)) : sel(a_i) == nt(a) >= i$ for each repeated activity $(a, role, dur) \in Acts$.

(iv) The objective function o is minimizing the OCT variable.

In the proposed constraint-based approach resources are implicitly constrained since COMET provides a high-level constraint modeling specific to scheduling which includes an efficient management of shared resources. Note that, besides the role-based allocation pattern, the CSP variables which are included in the model can be also used for specifying further resource constraints.[8] As an example, separation of duties (i.e. the ability to specify that two activities a and b must be allocated to different resources in a given workflow case) can be specified by including the next constraint in the proposed CSP model: $\forall\, i : 1 \leq i \leq nt(a)$, $\forall\, j : 1 \leq j \leq nt(b) : a_i \cdot res \neq b_j \cdot res$.

Figure 5 includes the translation from a ConDec-R specification into a CSP so that the CSP variables and constraints are stated as explained in Definition 4.3 (cf. Step 2). In general, for each repeated activity a, a CSP variable nt is added to the CSP model. Thereby, the value for $LB(nt(a))$ is initially set to 0 (it will be automatically updated during the solving process if an existence constraint is added through the corresponding filtering rule, cf. Sec. 4.2), and for $UB(nt(a))$

a rough initial estimate is made by considering the maximum obligatory cardinality MC of all repeated activities which is stated by existence constraints. For the constraint-based model depicted in Fig. 5, for example, the upper bound for all repeated activities is initially set to 2 (due to the existence constraint related to activity A). This value is high enough to ensure a feasible solution (the optimal solution, however, in general includes lower values of nt for several activities).

Moreover, $LB(\text{OCT})$ is initially set to 0, and $UB(\text{OCT})$ is estimated as the maximum cardinality times the sum of the duration of all the BP activities, i.e. $2 \times (5 + 3 + 4)$ in the example of Fig. 5. This is since the worst solution which can be obtained results in a plan which includes the execution of each BP activity the maximum number of times when all activities are sequentially executed.

For similar reasons, for each scheduling activity a_i, lower and upper bounds for st and et are set to lower and upper bounds of OCT. Furthermore, in general, $Dom(res(a_i)) = \{1 \dots \#role(role(a))\}$, i.e. $Dom(res(A_i)) = \{1 \dots 2\}$ and $Dom(res(C_i)) = \{1 \dots 2\}$ for any i since $\#R_1 = 2$, and $Dom(res(B_i)) = \{1\}$ for any i since $\#R_2 = 1$ for the constraint-based model depicted in Fig. 5. In addition, for each scheduling activity a_i, $Dom(sel(a_i)) = \{0 \dots 1\}$, since sel is a binary variable indicating whether or not the scheduling activity is selected to be executed.

4.2. *Filtering rules*

To improve the modeling of the problems and to efficiently handle the constraints in the search for solutions, our constraint-based proposal includes for each ConDec template a related global constraint implemented through a filtering rule (responsible for removing values which do not belong to any solution) for the definition of the high-level relations between the BP activities. In this way, the constraints stated in the ConDec-R specification (cf. Definition 3.1) are included in the CSP model through the related global constraints. These global constraints facilitate the specification of the problem. At the same time, the related filtering rules enable the efficiency in the search for solutions to increase. This is since during the search process these filtering rules remove inconsistent values from the domains of the variables. In the CSP model specification, initial estimates are made for upper and lower bounds of variable domains (cf. Sec. 4.1), and these values are refined during the search process. The developed filtering rules (cf. Ref. 34) are considered in the search algorithms.

4.3. *Solving the COP*

Once the problem is modeled, several constraint-based mechanisms can be used to obtain the solution for the COP, i.e. optimized enactment plans (cf. Definition 4.4). Since the generation of optimized plans presents NP-complexity,[35] it is not possible to ensure the optimality of the generated plans for all the cases. The developed

constraint-based approach, however, allows solving the considered problems in an efficient way.

Definition 4.4. A *BP enactment plan* is composed by: (i) the number of times each BP activity is executed, (ii) the start and the completion times for each activity execution, and (iii) the resource which is used for each activity execution.

By taking into account the NP-complexity of the considered problems, we adapted the existing solver COMET[42] by applying an *incomplete search* for solving the specific considered problems. We also evaluate its suitability for the generation of BPMN models from constraint-based specifications (cf. Sec. 8). This incomplete search includes *randomized components* in order to diversify the search. By means of this approach, a first feasible solution is quickly found by a randomized greedy algorithm. The same greedy algorithm is used for iteratively improving the best solution found until a time limit is reached. Through this incomplete search, all the solutions can be reached and the search procedure efficiently explores a wide range of solutions from diversified areas of the search space.

In general, when optimizing a CSP variable, if a feasible solution which is known exists, the value of the variable to optimize in the known solution can be used for discarding large subsets of fruitless candidates by using upper and lower estimated bounds of the quantity being optimized during the search process. Thus, if a known feasible solution S for the problem to solve exists, the objective value for this solution (S^{OCT}) is a valuable information which can be added to the constraint model through the constraint $\mathrm{OCT} < S^{\mathrm{OCT}}$. Thus, some non-optimal candidates, i.e. candidates whose OCT value cannot be less than S^{OCT} in any case, are discarded during the search, increasing the efficiency in the search for solutions.

Moreover, in our proposal, during the search process, some of the values which only lead to non-feasible solutions, i.e. inconsistent values, are removed from the domains of the CSP variables through the developed filtering rules (cf. Sec. 4.2) in order to reduce the search space by maintaining arc consistency (cf. Definition 4.5).

Definition 4.5. A $\mathrm{CSP} = (V, D, C_{\mathrm{CSP}})$ presents *arc consistency* iff for all pairs of CSP variables $(var_1, var_2) \,|\, var_1, var_2 \in V$, for each value of var_1 in the domain of var_1 there is some value in the domain of var_2 that satisfy all the constraints stated in C_{CSP} between var_1 and var_2, and vice versa.

In the proposed approach, the developed filtering rules and CSP modeling (cf. Sec. 4.1) are implemented such that they maintain the arc consistency for all pairs of CSP variables during all the search process.

5. From Optimized Enactment Plans to Optimized Business Process Models

Section 4 has described how optimized BP enactment plans can be generated from ConDec-R specifications. This section describes how a BPMN model which includes

the same activities to be executed in the same ordering and also using the same resources can be generated from the optimized enactment plan.

For each role in the BP enactment plan, a BPMN pool (cf. Definition 5.1) is created, which contains as many lanes as number of available resources for that role.

Definition 5.1. A *BPMN pool* $BPMNPool = (role, \#role)$ is a pool of a BPMN model, which is composed of $\#role$ lanes.

Moreover, for each scheduling activity in the BP enactment plan a BPMN activity (cf. Definition 5.2) is created. Additionally, one start activity and one end activity are included in the BPMN model.

Definition 5.2. A *BPMN activity* $BPMNAct = (pool, lane, dur, st)$ is an activity of a BPMN model placed in the lane named *lane* of the pool named *pool*, with duration *dur* and start time *st*.

One of the most important aspects to be considered for the generation of optimized BPMN models are the precedence relations between the BPMN activities (scheduling activities). For establishing these precedence relations the values for the start and the end times of the scheduling activities in the enactment plan are considered. These precedence relations are then used as a basis for generating BPMN models (cf. Definition 5.6) from BP enactment plans. Some related definitions are given below:

Definition 5.3. In a BP enactment plan regarding a CSP solution S, a scheduling activity a_i is a *predecessor* of another scheduling activity b_j, $a_i \in predecessors(b_j)$, if the relation $S^{et(a_i)} \leq S^{st(b_j)}$ holds due to resource or template relations.

Definition 5.4. In a BP enactment plan, a scheduling activity a_i is a *direct predecessor* of another scheduling activity b_j, $a_i \in DP(b_j)$, if $a_i \in predecessors(b_j) \wedge \nexists c_k \in predecessors(b_j) \,|\, a_i \in predecessors(c_k)$.

Definition 5.5. In a BP enactment plan, a scheduling activity a_i is an *indirect predecessor* of another scheduling activity b_j, $a_i \in IP(b_j)$, if $a_i \in predecessors(b_j) \wedge \exists\, c_k \in predecessors(b_j) \,|\, a_i \in predecessors(c_k)$.

Definition 5.6. A *BPMN model* $BPMN = (Pools, Activities, SequenceFlows, ParallelM)$ related to a ConDec-R process model $CR = (Acts, C_{\text{BP}}, Res)$ (cf. Definition 3.1) and to a solution S (cf. Definition 2.8) of the related CSP-ConDec problem (cf. Definition 4.3) is a BP model specified through the BPMN language, where:

(1) $Pools = \{BPMNPool(role, \#role), (role, \#role) \in Res\}$.
(2) $Activities = \{BPMNAct(role(a),\ S^{res(a_i)}, dur(a),\ S^{st(a_i)}),\ (a, role, dur) \in Acts,\ i \in [1 \ldots S^{nt(a)}]\} \cup \{start = BPMNAct(P_0, L_0, 0, 0)\} \cup \{end = BPMNAct(P_0, L_0, 0, \max_{(a,role,dur) \in Acts, i \in [1 \ldots S^{nt(a)}]} S^{et(a_i)})\}$.

(3) Let the set $Predecessors$ be:

(i) $\{(start, a_i) \,|\, (a, role, dur) \in Acts, i \in [1 \ldots S^{nt(a)}], S^{st(a_i)} = 0\} \cup$

(ii) $\{(a_{S^{nt(a)}}, end) \,|\, (a, role, dur) \in Acts,\ \nexists b_i,\ i \in [1 \ldots S^{nt(b)}],\ (b, role_b, dur_b) \in Acts,\ a_{S^{nt(a)}} \in predecessors(b_i)\} \cup$

(iii) $\{(b_i, c_j) \,|\, i \in [1 \ldots S^{nt(b)}],\ (b, role_b, dur_b) \in Acts,\ j \in [1 \ldots S^{nt(c)}],\ (c, role_c, dur_c) \in Acts, b_i \in DP(c_j)\}$,

Then:

(a) $SequenceFlows = \{(b_i, c_j) \,|\, (((b, role_b, dur_b) \in Acts \wedge i \in [1 \ldots S^{nt(b)}]) \vee b_i = start) \wedge (((c, role_c, dur_c) \in Acts \wedge j \in [1 \ldots S^{nt(c)}]) \vee c_j = end) \wedge (b_i, c_j) \in Predecessors \wedge |\{d_k, (((d, role_d, dur_d) \in Acts \wedge k \in [1 \ldots S^{nt(d)}]) \vee d_k = start), (d_k, c_j) \in Predecessors\}| = 1)\}$.

(b) $ParallelM = \{(Sources, c_j) \,|\, (((c, role_c, dur_c) \in Acts \wedge j \in [1 \ldots S^{nt(c)}]) \vee c_j = end) \wedge Sources = \{b_i, (((b, role_b, dur_b) \in Acts \wedge i \in [1 \ldots nt(b)]) \vee b_i = start) \wedge (b_i, c_j) \in Predecessors\} \wedge |Sources| > 1\}$.

In this way, through the set $Predecessors$, the precedence relations between activities are stated so that (i) the start activity is predecessor of all scheduling activities whose st value is equal to 0, (ii) the activities which are not predecessors of any other activity, are predecessor of the end activity, and (iii) in general, one activity b_i is predecessor of another activity c_j iff b_i is direct predecessor of c_j (cf. (3) in Definition 5.6). The set $Predecessors$ is represented in the BPMN model by BPMN sequence flows between a source activity b_i and a sink activity c_j, in the case that b_i is the only predecessor of c_j (cf. (3)(a) in Definition 5.6), or by a parallel merging gateway between a set of source activities $Sources$ and a sink activity c_j in the case that c_j has more than one predecessor (cf. (3)(b) in Definition 5.6). Note that parallel merging gateways (i.e. parallel gateways which have several sources and only one sink) need to be explicitly included in the resulting BPMN model, since they do not have the same meaning as several binary sequence flows from several sources and one sink. However, parallel splitting gateways (i.e. parallel gateways which have several sinks and only one source) do not need to be explicitly included in the resulting BPMN model since several binary sequence flows between one source activity and several sink activities have the same meaning as a parallel splitting gateway in the BPMN language.

The pseudocode and complexity analysis of the algorithms which were developed for generating BPMN models from optimized BP enactment plans are included in Appendix A. As a brief summary of this appendix, there is a main algorithm, Algorithm A.1, which constructs a BPMN model from a ConDec-R model and from a related optimized BP enactment plan (cf. Definition 5.6). As stated before, one of the most important aspects to be considered for this model generation are the precedence relations between the scheduling activities of the plan, which are managed by Algorithm A.2. These precedence relations are due to (1) resource constraints, i.e. the activities are allocated in the resources in a specific order in the generated enactment plan, and (2) ConDec-R constraints related to precedence between

activities. Typically, unlike resource precedence relations, precedence relations due to ConDec-R constraints cannot be easily obtained. To this end, each ConDec-R template presents a method which is in charge of determining the precedence relations which are given between the scheduling activities related to the repeated activities which are involved in that ConDec-R template. The mentioned method for some representative ConDec-R templates is detailed in Algorithms A.3–A.5.

6. Allowing for Run-Time Flexibility

The execution plans generated in Sec. 4 provide an optimal way for executing the source ConDec model assuming certain estimated values and all decision to be goal-based. Even though these assumptions are valid for certain environments (e.g. certain web service settings) estimates might not always be accurate or some decisions might depend on run-time information. For this, the approach described in Secs. 3–5 is extended in this section to allow decisions to be deferred at run-time (cf. Sec. 6.1), and to allow the BPMN model to be dynamically adapted during run-time (cf. Sec. 6.2).

6.1. *Late-planning activities*

Executing a ConDec model usually entails dealing with decisions related to (1) how many times one activity is being executed, and (2) the order of execution of the activities. We assume that at least the decisions related to the order of execution of the activities are goal-based. However, we consider non-goal-based decisions (e.g. user-based decisions), if needed, regarding the number of executions of a particular activity. Related to these decisions, in turn, in ConDec one activity can be executed arbitrarily often if not restricted by any constraint. However, there are some ConDec templates which constrain the number of executions of the activities, resulting either in a specific value (e.g. A must be executed exactly twice), or in a range (e.g. A must be executed either once or twice). The number of times one activity should be executed can be stated by one specific constraint (e.g. Exactly(A, 2)), or by the combination of several constraints (e.g. the combination of Exactly(A, 2) together with ChainSuccession(A, B) implies that B should be executed exactly twice). To be able to deal with decisions related to the number of times certain activities are being executed which are not goal-based, we propose to encapsulate these activities (together with the relations in which they are involved) in a complex declarative late-planning activity when specifying the ConDec-R model, i.e. we propose the use of hierarchical models. In declarative models the activities included in a complex activity should be such that they can be executed in isolation from the top-level process.[37]

Encapsulating decisions which are not goal-based in a fragment allows dealing with each sub-process (i.e. complex activity) as if it were a black box, and therefore, our approach can be directly applied (even enabling multiple instance

optimization). Therefore, when creating the optimized enactment plans from the ConDec-R specification (cf. Sec. 4), each late-planning activity is treated as an atomic activity, and it is managed as a repeated activity (cf. Definition 4.1). In this way, when generating the BPMN model (cf. Sec. 5) the complex activities are then integrated into the BPMN model by substituting the BPMN activity related to the complex activity by the associated imperative fragment. The following sectiosn describes the generation process. For sake of clarity we describe in different subsections how constraints (cf. Sec. 6.1.1), resources (cf. Sec. 6.1.2), and durations (cf. Sec. 6.1.3) are managed.

6.1.1. *Constraints*

The BPMN fragment associated to a specific complex activity is generated as follows:

(1) Generating all possible combinations of declarative models in such a way that all different possibilities for nt (i.e. number of times) for each activity are covered. This is done by stating Exactly constraints for all the possible values for the number of executions for all the activities which belong to the complex activity. Specifically, for each activity A whose number of executions should be in a range [Min ... Max], the generated models should cover all the possibilities (i.e. Exactly(A, nt), $\forall\, nt \in$ [Min ... Max]) in combination with all the possibilities for the other activities. Note that the maximum number of execution times for each activity belonging to a complex activity needs to be established, otherwise, the possibilities are not finite.
(2) For each declarative model which is generated, related optimized enactment plans are created (i.e. local optimization for each possible feasible declarative model is addressed) through the proposed constraint-based approach (cf. Sec. 4).
(3) These optimized plans are then translated to BPMN fragments (cf. Sec. 5).
(4) These fragments are then linked by using existing merging algorithms (e.g. Ref. 38). Note that the resulting fragment will include XOR gateways when necessary.

When generating the different combinations of declarative models (i.e. step (1)) it is possible that some unfeasible combinations exist. In these cases, no related optimized enactment plan is generated, and therefore, the related BPMN fragment is not considered when merging.

Figure 6 shows an example of the complete process over a fragment which includes five BP activities (A, B, C, D and E) and five existence relations (i.e. all activities should be executed at most once) together with five binary relations (i.e. (1) $ExChoice(A, C)$, implying that either A or C (but not both) must be executed, (2) $ExChoice(B, D)$, implying that either B or D (but not both) must be executed, (3) $Response(A, B)$, implying that after the execution of A, B should

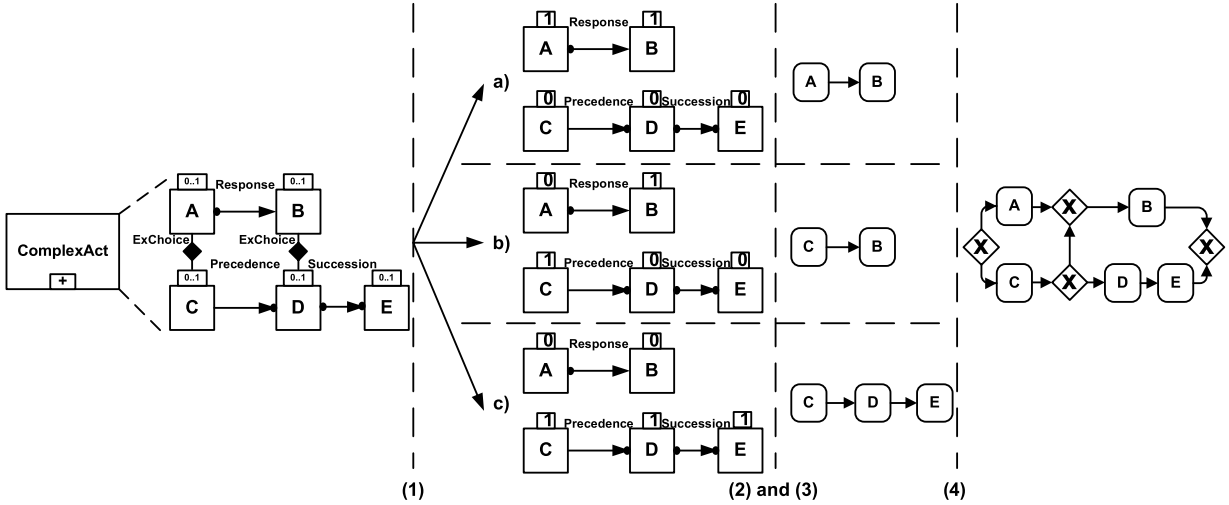


Fig. 6. Generating BPMN fragments from declarative complex activities.

be eventually executed, (4) $Precedence(C, D)$, implying that before the execution of D, C should be executed, and (5) $Succession(D, E)$, implying that after the execution of D, E should be executed and before the execution of E, D should be executed). Given that declarative specification, there are three feasible scenarios, i.e. three possible ways to execute the specification ensuring that all constraints are satisfied:

(a) A is executed once; C is not executed due to $ExChoice(A, C)$; B is executed once after A due to the $Response(A, B)$ constraint; D is not executed due to $ExChoice(B, D)$, therefore also E cannot be executed due to $Succession(D, E)$.
(b) C is executed once; A is not executed due to $ExChoice(A, C)$; B is executed once; D is not executed due to $ExChoice(B, D)$, therefore also E cannot be executed due to $Succession(D, E)$. In the related optimized enactment plan, both options (B succeeding C or C succeeding B) are feasible. For this example, we consider that the option B succeeding C is more optimized than C succeeding B (note that for each feasible scenario only the most optimized plan is selected for the merging, as explained in the step (2) of the process).
(c) C is executed once; A is not executed due to $ExChoice(A, C)$); D is executed once; B is not executed due to $ExChoice(B, D)$. Since D is executed, E should be also executed due to $Succession(D, E)$. In the related optimized enactment plan, C should precede D due to $Precedence(C, D)$, and D should precede E due to $Succession(D, E)$.

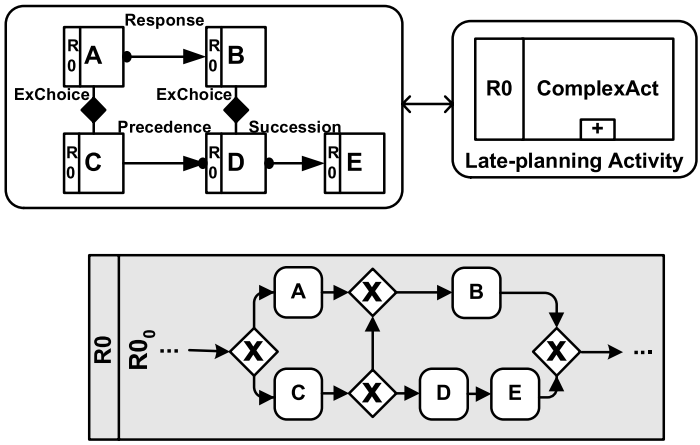
In this example, some unfeasible combinations for nt exist. For example, the scenario in which A is executed once and D is executed once is unfeasible since two relations (i.e. $Response(A, B)$ and $Precedence(C, D)$) are violated.

In Fig. 6, the different BPMN fragments (related to the optimized enactment plans) which are obtained from the three feasible scenarios have been merged using the tool presented in Ref. 38. For the sake of clarity, in Fig. 6 information related to resources and durations of activities has been omitted.

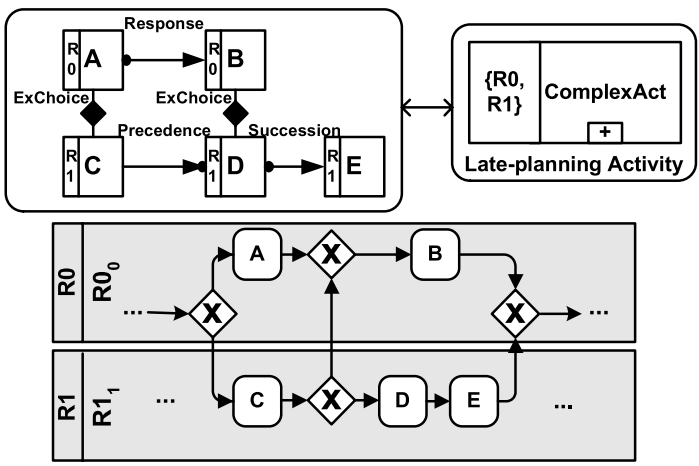
Note that optimization is locally applied within each complex activity since for each declarative model which is generated (i.e. for each possibility) optimized enactment plans are generated.

6.1.2. *Resources*

For each complex activity, required resources need to be stated when including this activity in the ConDec-R model. When all the activities which belong to the same complex activity require resources related to the same role, the complex activity will also require that role, and the proposed approach can be directly applied (cf. Fig. 7(a), where all the activities require a resource of role R0). However, when the activities which belong to the same complex activity require resources related to different roles, some adjustments are required, e.g. encapsulating the declarative



(a)



(b)

Fig. 7. Complex activities: Dealing with resources. (a) Activities requiring resources related to the same role, (b) activities requiring resources related to different roles.

sub-process in a complex activity which requires as many resources as different roles are included in the sub-process (cf. Fig. 7(b)), i.e. the constraint-based approach needs to be adapted to allow for activities which require multiple resources, resulting in a cumulative scheduling problem.[39] This extension can be easily achieved since most constraint-based systems provide a high-level constraint modeling specific to scheduling which includes an efficient management of shared resources for well-known scheduling problems, which is the case of the cumulative scheduling problem. When generating the BPMN model, each activity of the sub-process needs to be associated to the suitable lane (cf. Fig. 7). Note that, in the proposed approach,

the required resource is considered to be used throughout the duration of the activity.

6.1.3. *Durations*

Moreover, for each complex activity, estimated durations need to be stated when including this activity in the ConDec-R model. The estimated durations of the complex activities can be calculated in different ways, e.g. as (1) the average duration of these complex activities in past process executions (i.e. by analyzing event logs), and hence, trying to optimize the resulting plan as much as possible although usually more replanning will be required, or (2) the maximum duration of these complex activities in past process executions, and hence, the plan is probably less optimized but less replanning will be required.

6.2. *Replanning*

Since estimates might not always be accurate and resource availabilities might unexpectedly change, the generated BPMN model is dynamically adapted during runtime by using replanning, and hence allowing for an increased flexibility (cf. Fig. 8). As can be seen, as execution proceeds, the BP enactment and the resource availabilities are monitored (`a` in Fig. 8). All new events, i.e. activities get started/completed or resources become available/unavailable, are stored in an Event Log (`b` in Fig. 8). Whenever events are updated the Replanning Module (`c` in Fig. 8) analyzes the optimized plan (`d` in Fig. 8) as well as the events. In particular, it checks if the current execution matches with the optimized enactment plan or whether updates of both the enactment plan (`f` in Fig. 8) and the BPMN model (`g` in Fig. 8) are required. In general, updates can become necessary due to (1) deviations, i.e.

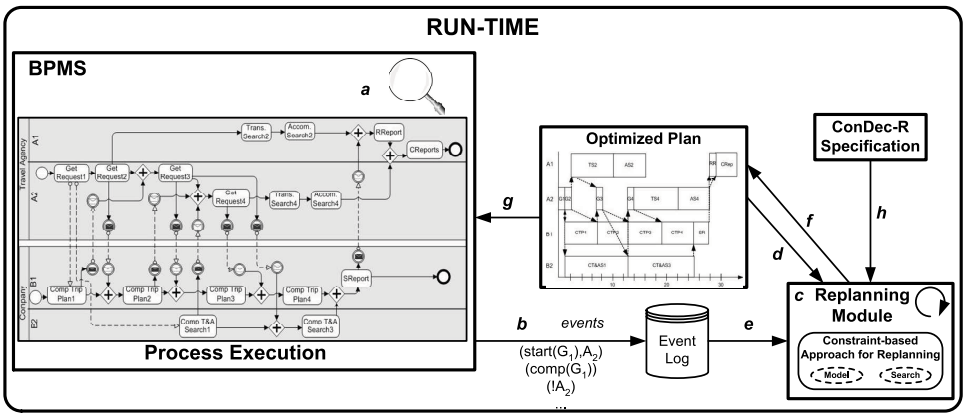


Fig. 8. Flexible execution of BPMN models.

estimates are incorrect (e.g. when activity executions take longer/shorter than estimated), or (2) resource availabilities change (e.g. resources become unavailable). Note that not every deviation requires replanning due to the slack of some activities in the enactment plan. If plan updates are required, the Replanning Module needs to access the ConDec-R specification of the process (h in Fig. 8) to generate a new optimized plan which considers the actual partial execution of the process by using the proposed constraint-based approach (cf. Sec. 4). The generated optimized plan is, in turn, translated to an optimized BPMN model which is used for updating the current BPMN model in a way that the part which has been already executed remains unchanged, and the part which remains to be executed is replaced. Despite the NP-complexity of the considered problems, in general, replanning is less time consuming than initial planning, since most of the information about previous generated plans can usually be reused, and CSP variable values become known as execution proceeds.

Note that changing a deployed BPMN model and migrating running instances to a new schema can be quite challenging since respective changes must not violate process model correctness and proper instance execution.[23] However, in our approach, the proposed model adaptation and instance migration can be handled properly as detailed in the following.

On the one hand, in process model evolution it is necessary to check that the new model is (1) correct, i.e. it meets the structural properties required by the process modeling language used and (2) sound, i.e. it obeys proper completion and absence of dead activities.[23] In our approach, the generated BPMN model is correct since the automated generation guarantees that the new model meets the structural properties required by BPMN. Moreover, it is sound since the model is automatically generated from a feasible enactment plan which meets all the constraints imposed by the declarative specification and reaches the specified goal. Since the generation of the new models is not manual but completely automated, no errors can be unintentionally introduced.

On the other hand, once a new correct and sound model is deployed, the BPMS must properly deal with corresponding process instances, i.e. process instances which were started and partially executed on the previous model, but have not yet been completed.[23] In this way, in addition to structural properties, the BPMS needs to consider the state of a process instance when adapting its process model,[23] i.e. depending on the current state of a process instance, certain changes should be allowed while others must be prohibited (e.g. it must not be possible to change the past of a process instance). Specifically, the running process instance should be state compliant with the new process model. A process instance is state compliant with an updated process model (i.e. can therefore be migrated to it) if the execution trace of the instance is producible on the new model as well. In our case, there is only one running instance (which comprises the execution of all instances which were planned within a specific timeframe) which has to be migrated to the new model version. This is not problematic in our approach

since the new model is generated through replanning from the partial execution trace of this instance. Therefore, this trace will be always producible on the new model, i.e. everything which has been done before can be done in the new model. However, for migrating this instance to the new process model version, activity states might have to be adapted to enable proper continuation of instance execution afterwards. As an example of instance state adaptation, it might become necessary to immediately enable or disable certain activities before continuing with the execution of the process instance.[23] Using selected commercially available state-of-the art BPMSs (e.g. AristaFlow BPM Suite[40]) respective changes can be accomplished.

7. An Example

In this section, an example, the travel agency problem, is developed in order to clarify the overall proposed approach. First, the proposed problem is detailed (cf. Sec. 7.1), together with its ConDec-R specification (cf. Sec. 7.2). The generated optimized enactment plan and the corresponding BP model representation are then shown and explained (cf. Sec. 7.3). The example deals with a set of representative templates in order to illustrate various kinds of relations which can be given between activities of BPs.

7.1. *The travel agency problem*

The analyzed example represents an agency which manages holiday bookings by offering clients the following three services: transport, accommodation, and guided excursions. For some of the services the travel agency can ask a travel company to help in managing some client requests. After all the client requests are carried out, the agency must write a report which contains the information in answer to the requests, which will then be sent to the clients. For efficiency reasons, the agency creates only one report a day, hence it must be written after all the requests are carried out.

The activities which can be executed in order to deal with the client requests are detailed in Table 1.

Assume that the travel agency wants to minimize the response time for the clients (end time of the client-report activity). For this the agency not only considers the number of client requests ($\#P$), which is known at the beginning of each working day, but also the number of resources available in the agency ($role\ A, \#A$) and the number of available resources in the company ($role\ B, \#B$).

7.2. *ConDec-R specification for the travel agency problem*

In order to solve this problem through the proposed approach, the first step is the creation of the related ConDec-R specification (cf. Fig. 9). Since a ConDec-R

Table 1. Activities of the travel agency problem.

Id	Description	Constraints	Role	Dur.
G	The client request is received by the agency	The following client request cannot be received until the current request has *started* to be processed (both trip plan and transport search have started)	A	1
TS	The agency searches for a suitable transportation	Can only be executed after G has completed, and if the agency and not the company deals with the search of transport for this request	A	8
AS	The agency searches for suitable accommodation	Must be executed after each TS	A	6
TP	The agency organizes a trip plan	Can only be executed after G has completed, if the agency and not the company deals with the creation of the trip plan for this request	A	5
$CT\&AS$	The company searches for transport and accommodation	Can only be executed after G has completed, if the company and not the agency deals with the search of transport for this request	B	12
CTP	The company creates a trip plan	Can only be executed after G has completed, if the company and not the agency deals with the creation of the trip plan for this request	B	6
$SReport$	The company sends a report to the agency which includes information about all the trip requests	Must be executed after all the activities related to the client request which are carried out by the company have finished, when there is at least one client request	B	3
$RReport$	The agency receives the report with all the trip requests	Must be executed after each $SReport$	A	1
$CReports$	The agency writes a report which includes information about all handled requests	Must be executed after having completed all activities. It is executed only once	A	4

specification contains two parts which are independent, these parts can be specified separately fostering their reuse:

- Information about the BP activities (required resources, durations, as well as unary constraints) and the high-level relations which are given between the BP activities (cf. Fig. 9(a); Table 2). For representing the BP activities as well as unary constraints and high-level relations, ConDec graphical notation[30] is used: (1) each BP activity is represented by a box which contains the name of the activity (e.g. Get Request), (2) an unary constraint over a BP activity is represented by a little box above that activity which contains information regarding

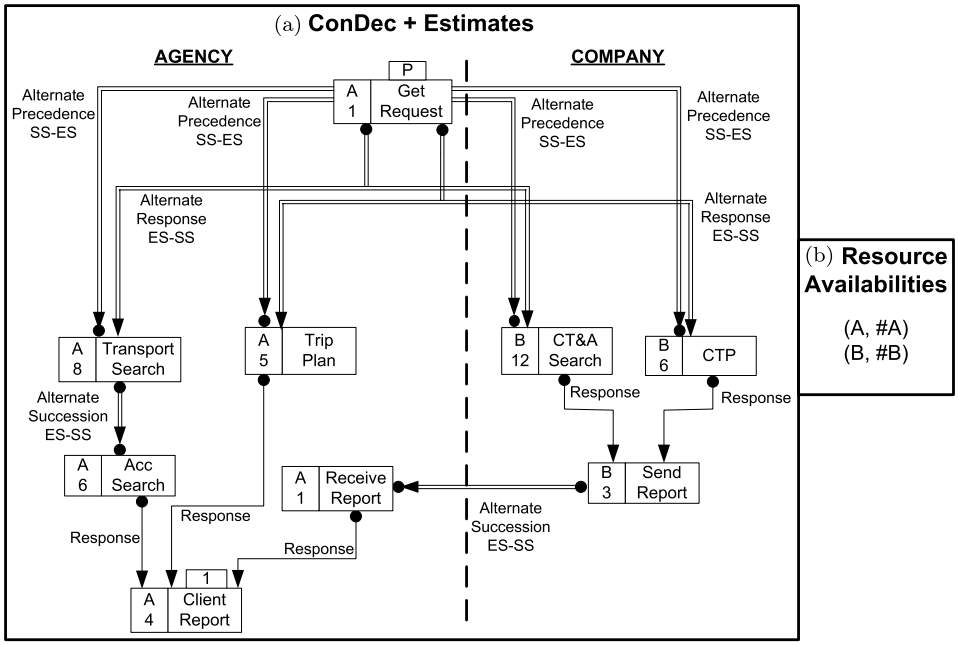


Fig. 9. ConDec-R specification for the travel agency problem.

the number of times that activity needs to be executed (e.g. exactly P over Get Request), and (3) high-level relations between BP activities are depicted by special arrows whose shape depends on the specific relation which is represented (e.g. Alternate Precedence SS-ES between Get Request and Transport Search). Since ConDec activities have been extended to ConDec-R activities by including the role of the required resource and the estimated duration, the graphical representation of ConDec-R activities includes this new information in a box at the left side of the activities (e.g. Get Request requires a resource with role A, and has an estimated duration of 1).

- Information about the roles and available resources (cf. Fig. 9(b)). This information is represented by a box which contains a tuple $(role, \#role)$ for each role which is considered in the process definition, which represents that there are $\#role$ available resources with role $role$.

7.3. *Optimized enactment plan and optimized BP model for the travel agency problem*

Using the ConDec-R specification of Fig. 9, the related constraint problem is generated and solved through the constraint-based proposal which is described in Sec. 4, resulting in an optimized enactment plan for the travel agency problem. This plan is used for the generation of an optimized BP model.

Table 2. Constraints of the travel agency problem.

Relation	Description
$Exactly(\#P, G)$	G must be executed each time a client request is received.
$Exactly(1, CReports)$	$CReports$ must be executed exactly once.
$AltPrecSS - ES(G, TS)$	Before the first execution of TS, G must be executed, and between two executions of TS (TS_{i-1} and TS_i), G must also be executed. Formally: $st(G_j) \geq st(TS_{i-1})$, i.e. the next client request can be received at the same moment the previous request *starts* to be processed. $st(TS_i) \geq et(G_j)$, i.e. a transport search cannot start until the client request is completely received (*finishes*).
$AltPrecSS - ES(G, CT\&AS)$	Exactly the same than $AltPrecSS - ES(G, TS)$ by replacing TS by $CT\&AS$.
$AltPrecSS - ES(G, TP)$	Exactly the same than $AltPrecSS - ES(G, TS)$ by replacing TS by TP.
$AltPrecSS - ES(G, CTP)$	Exactly the same than $AltPrecSS - ES(G, TS)$ by replacing TS by CTP.
$AltRespES - SS(G, \{TS, CT\&AS\})$	After the last execution of G, at least one of TS or $CT\&AS$ must be executed, and between two executions of G, at least one of TS or $CT\&AS$ must also be executed.
$AltRespES - SS(G, \{TP, CTP\})$	Exactly the same than $AltRespES - SS(G, \{TS, CT\&AS\})$ by replacing $\{TS, CT\&AS\}$ by $\{TP, CTP\}$.
$AltSucES - SS(TS, AS)$	Before the first execution of AS, TS must be executed, and between two executions of TS (TS_{i-1} and TS_i), AS must also be executed. Furthermore, after the last execution of TS, AS must be executed, and between two executions of AS (AS_{i-1} and AS_i), TS must be executed. Formally: $st(TS_j) \geq st(AS_{i-1})$, i.e. the next transport search can be received at the same moment the previous one *starts* to be processed. $st(AS_i) \geq et(TS_j)$, i.e. the accommodation search cannot start until the transport search is *completed*.
$Resp(CT\&AS, SReport)$	After the last execution of $CT\&AS$, $SReport$ must be executed.
$Resp(CTP, SReport)$	After the last execution of CTP, $SReport$ must be executed.
$AltSucES - SS(SReport, RReport)$	Exactly the same than $AltSucES - SS(TS, AS)$ by replacing TS by $SReport$, and AS by $RReport$.
$Resp(RReport, CReports)$	After the last execution of $RReport$, $CReports$ must be executed.
$Resp(AS, CReports)$	After the last execution of AS, $CReports$ must be executed.
$Resp(TS, CReports)$	After the last execution of TS, $CReports$ must be executed.

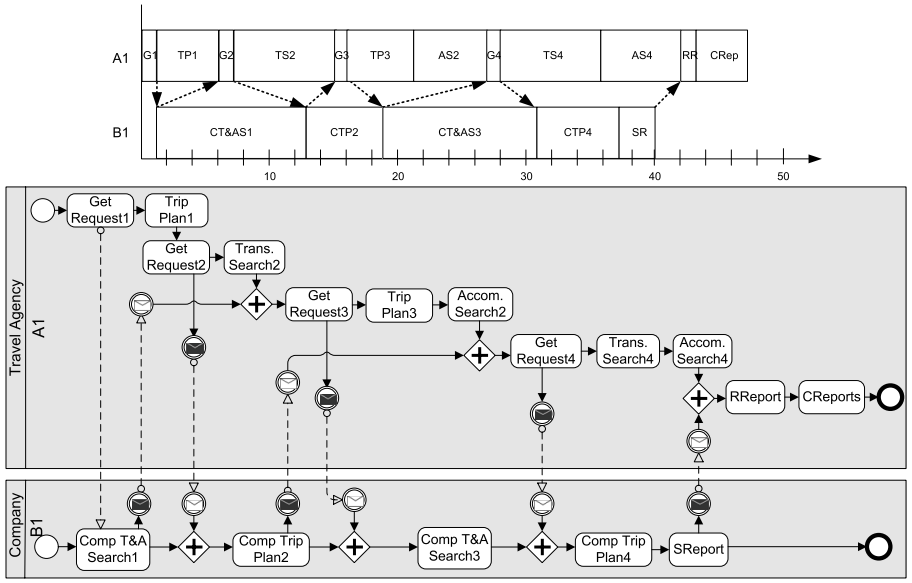


Fig. 10. Optimized Gantt chart and BPMN for the travel agency problem for $\#P = 4$, $\#A = 1$ and $\#B = 1$.

As commented, in order to define an instance $(\#P, \#A, \#B)$ for the travel agency problem, the following parameters must be stated: number of client requests $(\#P)$, number of resources available in the agency $(\#A)$, and number of resources available in the company $(\#B)$.

In this section, two instances are studied as illustrations, specifically Problem 1 defined by $(\#P = 4, \#A = 1, \#B = 1)$, and Problem 2 defined by $(\#P = 4, \#A = 2, \#B = 2)$. For Problem 1, Fig. 10 shows both the optimized Gantt chart (OCT = 47) and the BP model which are obtained through our proposal. It can be seen that regarding the first client request, depicted by G1 in the Gantt diagram), the trip plan is created by the agency (TP1 activity), while the transport and accommodation search are carried out by the company (CT&AS1). Once both activities TP1 and CT&AS1 *start*, the second client request can be received (G2). The fact that an activity B can only start after another activity A has *started* is stated by considering the predecessors of A as predecessors of B. In this case, the predecessor of TP1 and CT&AS1 (i.e. G1) must (directly o indirectly) precede G2. Regarding the second Get Request activity (G2), the trip plan is organized by the company (CTP2 activity), while the transport and accommodation search are carried out by the agency (TS2 and AS2 activities). In this case, activity AS2, which is related to the second request, is postponed until after the end of the execution of other activities related to the third request (G3, TP3), for efficiency reasons (notice that there is no constraint between the repeated activities TP and AS). Once both activities CTP2 and TS2 start, the third client request can be received (G3). Regarding the third Get request (G3), the trip plan is created by the agency

(TP3 activity), while the transport and accommodation search are carried out by the company (CT&AS3 activity). Once both activities TP3 and CT&AS3 start, the fourth Get request can be received (G4). Regarding the fourth request (G4), the trip plan is created by the company (CTP4 activity), while the transport and accommodation search are carried out by the agency (TS4 and AS4 activities). After all the client requests which are carried out by the company are finished, the Send Report (SR) activity can be executed. After this, the Receive Report (RR) activity can be executed. Finally, after all activity executions, the Client Reports (CR) activity is executed.

In a similar way, for Problem 2, Fig. 11 shows the optimized Gantt chart (OCT = 33) and the BP model which are obtained with our proposal.

7.4. *Dynamic programming for the travel agency problem*

A feasible solution to a model of a number of instances can be obtained by concatenating known solutions for the same model with a smaller number of instances. The

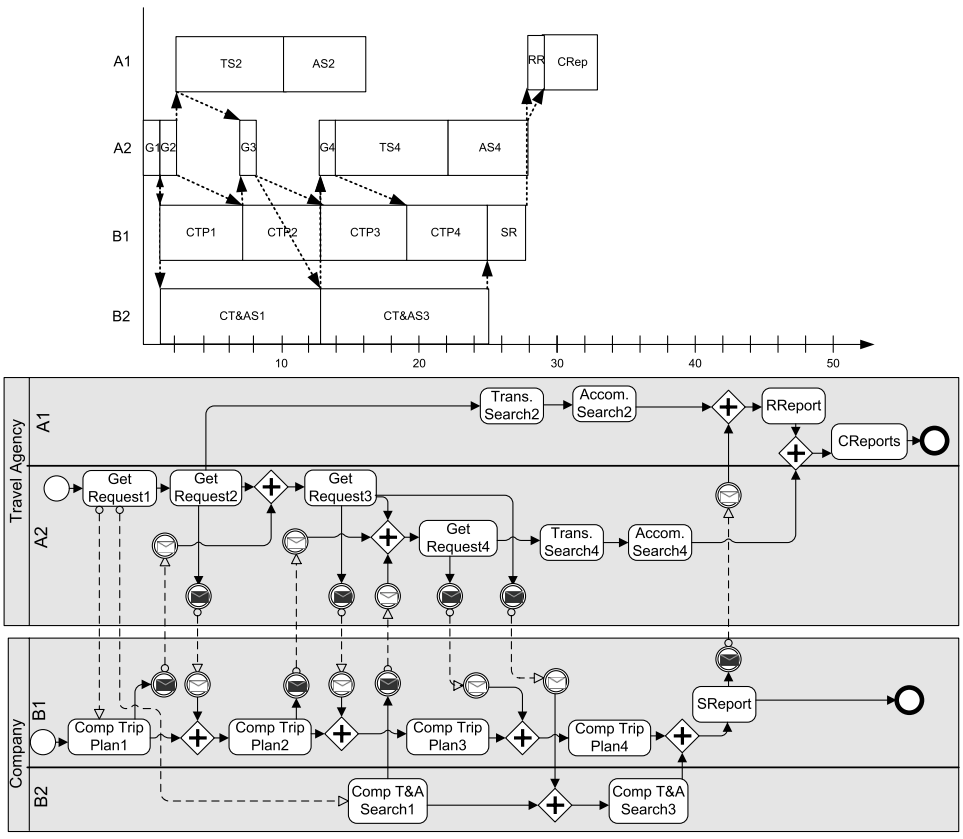


Fig. 11. Optimized Gantt chart and BPMN for the travel agency problem for $\#P = 4$, $\#A = 2$ and $\#B = 2$.

optimal way to perform this concatenation can be achieved by *dynamic programming* (DP).[41] For the travel agency problem, DP can be applied for obtaining good solutions by joining optimal solutions to smaller problems. Let $\mathrm{OCT}_{a,b}(p)$ be the best OCT which is known for an instance `(p, a, b)` of the travel agency problem. DP can be applied to the travel agency problem so that the best OCT for `(p, a, b)` obtained through DP, $\mathrm{OCT\_}DP_{a,b}(p)$, can be defined by:

$$\mathrm{OCT\_}DP_{a,b}(p) = \min_{1 \leq i \leq p/2} (\mathrm{OCT}_{a,b}(i) + \mathrm{OCT}_{a,b}(p-i) - dur(CReports))^{\mathrm{j}}$$

i.e. the best combination of two optimal/optimized solutions to smaller problems is chosen.

8. Empirical Evaluation

In order to evaluate the efficiency of the proposal, a controlled experiment is conducted. Section 8.1 describes the design underlying the experiment, and Sec. 8.2 shows the experimental results and the data analysis.

8.1. *Experimental design*

Purpose: The purpose of the empirical evaluation is to analyze our proposal in the generation of optimal enactment plans from ConDec-R specifications, specifically, the goals are: (1) the comparison of our constraint-based proposal with DP (cf. Sec. 8.2.1) and (2) the demonstration of its use for simulation purposes (cf. Sec. 8.2.2).

Objects: We used the travel agency problem as example for our evaluation, since it includes various and representative relations of several types and complexity from the set of all the ConDec-R templates.[k]

Independent Variables: For the empirical evaluation, the number of client requests, $\#P$, the number of resources of *roleA*, $\#A$, and the number of resources of *role B*, $\#B$, are taken as independent variables.

Response Variables: Some performance measures (cf. Table 3) related to the best generated plan are reported for the generated problems (Figs. 12; Tables 4–6).

Experimental Design: Based on the travel agency problem we generated a wide set of problem instances by varying the different independent variables: $\#P$, $\#A$ and $\#B$. For variable $\#P$, the values $1 \ldots 100$ are considered, for $\#A$, the values $1 \ldots 5$ are considered, and for $\#B$, the values $1 \ldots 5$ are considered.

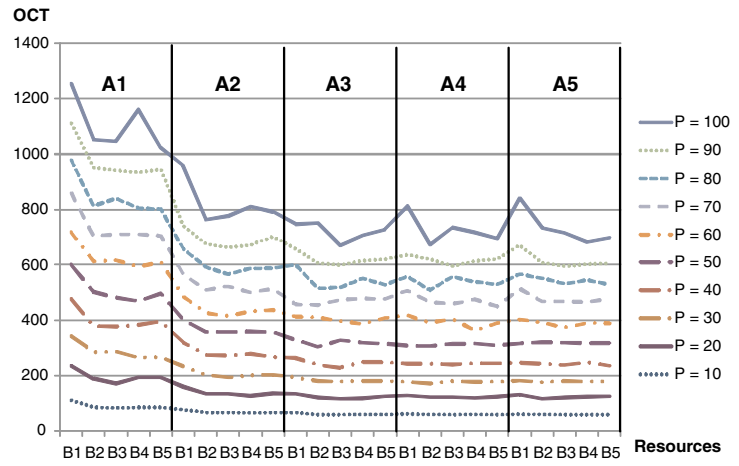
Experimental Execution: For the experiments, the constraint-based search algorithm is run until a 10-min CPU time limit is reached. The machine for all experiments is an Intel Core2, 2.13 GHz, 1.97 GB memory, running on Windows XP.

[j] `CReports` activity must be executed only once, and must be allocated after the execution of all other activities.

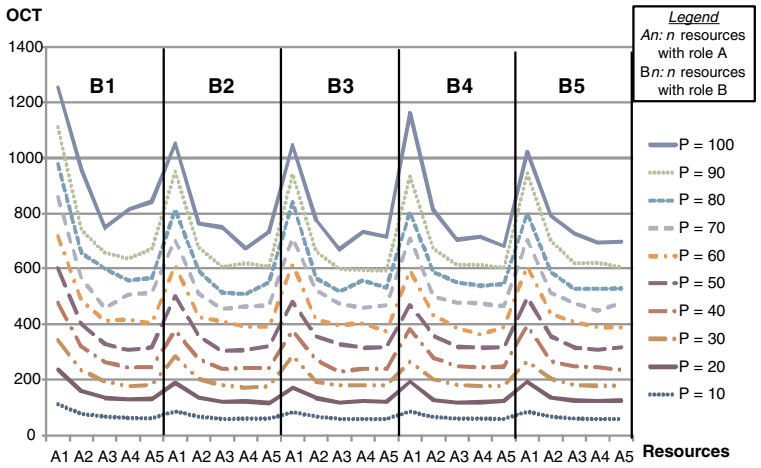
[k] A tool for generating optimized BP models for the travel agency problem can be found at http://regula.lsi.us.es/AgenciesOptimizedModels/, where some tests can be carried out.

Table 3. Response variables.

Id	Description
CP/DP($s1 + s2$)	The way in which the best solution is found, which can be by means of the proposed constraint-based approach, CP, or by DP through combining $s1$ and $s2$, DP($s1 + s2$).
OCT	OCT for the generated optimized enactment plan.
%BusyA	Average percentage of use of resources of role A, regarding the OCT.
%BusyB	Average percentage of use of resources of role B, regarding the OCT.



(a)



(b)

Fig. 12. OCT depending on $\#A$, $\#B$. (a) Resources are grouped by $\#A$, (b) resources are grouped by $\#B$.

Table 4. %OCT of the best solution found and method (CP or DP) that reaches it.

P	$(\#A, \#B)$	OCT	CP/DP$(s1+s2)$	P	$(\#A, \#B)$	OCT	CP/DP$(s1+s2)$
1	(1, 1)	20	CP	11	(1, 1)	119	DP$(5+6)$
1	(2, 2)	19	CP	11	(2, 2)	72	CP
1	(3, 3)	19	CP	11	(3, 3)	68	CP
1	(4, 4)	19	CP	11	(4, 4)	68	CP
1	(5, 5)	19	CP	11	(5, 5)	67	CP
2	(1, 1)	27	CP	12	(1, 1)	128	DP$(6+6)$
2	(2, 2)	21	CP	12	(2, 2)	80	DP$(5+7)$
2	(3, 3)	21	CP	12	(3, 3)	71	CP
2	(4, 4)	21	CP	12	(4, 4)	70	CP
2	(5, 5)	21	CP	12	(5, 5)	70	CP
3	(1, 1)	38	CP	13	(1, 1)	139	DP$(6+7)$
3	(2, 2)	28	CP	13	(2, 2)	85	CP
3	(3, 3)	28	CP	13	(3, 3)	78	CP
3	(4, 4)	28	CP	13	(4, 4)	75	CP
3	(5, 5)	28	CP	13	(5, 5)	75	CP
4	(1, 1)	47	CP	14	(1, 1)	149	DP$(6+8)$
4	(2, 2)	33	CP	14	(2, 2)	92	DP$(7+7)$
4	(3, 3)	33	CP	14	(3, 3)	84	CP
4	(4, 4)	33	CP	14	(4, 4)	84	CP
4	(5, 5)	33	CP	14	(5, 5)	84	CP
5	(1, 1)	57	CP	15	(1, 1)	160	DP$(7+8)$
5	(2, 2)	36	CP	15	(2, 2)	98	DP$(7+8)$
5	(3, 3)	35	CP	15	(3, 3)	91	DP$(5+10)$
5	(4, 4)	35	CP	15	(4, 4)	88	CP
5	(5, 5)	35	CP	15	(5, 5)	88	CP
6	(1, 1)	66	CP	16	(1, 1)	170	DP$(8+8)$
6	(2, 2)	44	CP	16	(2, 2)	103	CP
6	(3, 3)	43	CP	16	(3, 3)	97	CP
6	(4, 4)	43	CP	16	(4, 4)	93	CP
6	(5, 5)	43	CP	16	(5, 5)	93	CP
7	(1, 1)	77	CP	17	(1, 1)	181	DP$(8+9)$
7	(2, 2)	48	CP	17	(2, 2)	110	DP$(8+9)$
7	(3, 3)	45	CP	17	(3, 3)	101	DP$(7+10)$
7	(4, 4)	45	CP	17	(4, 4)	99	CP
7	(5, 5)	45	CP	17	(5, 5)	99	CP
8	(1, 1)	87	CP	18	(1, 1)	190	DP$(6+12)$
8	(2, 2)	54	CP	18	(2, 2)	116	DP$(9+9)$
8	(3, 3)	52	CP	18	(3, 3)	104	CP
8	(4, 4)	52	CP	18	(4, 4)	104	CP
8	(5, 5)	52	CP	18	(5, 5)	104	CP
9	(1, 1)	98	CP	19	(1, 1)	201	DP$(7+12)$
9	(2, 2)	60	CP	19	(2, 2)	122	DP$(8+11)$
9	(3, 3)	57	CP	19	(3, 3)	112	DP$(7+12)$
9	(4, 4)	57	CP	19	(4, 4)	111	DP$(7+12)$
9	(5, 5)	57	CP	19	(5, 5)	111	DP$(7+12)$
10	(1, 1)	109	DP(4+6)	20	(1, 1)	211	DP$(8+12)$
10	(2, 2)	67	CP	20	(2, 2)	128	DP$(9+11)$
10	(3, 3)	60	CP	20	(3, 3)	116	DP$(10+10)$
10	(4, 4)	60	CP	20	(4, 4)	116	DP$(10+10)$
10	(5, 5)	60	CP	20	(5, 5)	116	DP$(10+10)$

Table 5. %Busy A versus $\#P$.

$\#A$	$\#P$									
	10	20	30	40	50	60	70	80	90	100
1	95.2	95.1	95.0	95.0	95.0	95.0	95.0	95.0	95.0	95.0
2	79.8	82.6	81.4	83.1	82.2	81.7	82.6	83.3	82.8	82.4
3	73.5	74.8	74.3	75.5	75.1	74.8	75.4	75.9	75.6	75.4
4	57.8	59.0	59.1	59.6	59.6	59.5	59.7	59.9	59.8	59.8
5	43.9	46.0	45.9	46.5	46.3	46.2	46.5	46.7	46.6	46.5

Table 6. %Busy B versus $\#P$.

$\#B$	$\#P$									
	10	20	30	40	50	60	70	80	90	100
1	78.6	80.8	80.3	82.0	81.4	81.0	81.9	82.6	82.2	81.9
2	72.7	71.1	75.3	72.2	74.6	76.2	74.3	72.8	74.1	75.1
3	46.7	48.1	49.1	48.9	49.3	49.6	49.4	49.3	49.5	49.7
4	30.5	30.7	31.3	31.2	31.4	31.6	31.5	31.4	31.5	31.6
5	26.4	27.2	27.5	27.7	27.7	27.8	27.8	27.9	27.9	27.9

In order to solve the constraint-based problems (cf. Sec. 4), the developed algorithms have been integrated with the system COMET,[42] which is able to generate high-quality solutions for highly constrained problems in an efficient way.

8.2. *Experimental results and data analysis*

As commented, the purpose of the empirical evaluation is two-fold, i.e. analyzing the suitability of our proposal through a comparison with DP (cf. Sec. 8.2.1), and through the use for simulation (cf. Sec. 8.2.2).

8.2.1. *Comparison with DP*

DP (cf. Sec. 4.3) is a widely used technique in solving optimization problems, leading to solutions of high quality in most cases. Specifically, for the travel agency problem, DP can be applied (cf. Sec. 7.4). We would like to evaluate whether our constraint-based proposal usually improves the solution of good quality which can be obtained by DP, i.e. works efficiently.

Table 4 shows the OCT for the best solutions which are found for some representative instances, together with the method (either DP or CP) which finds the best solution (column CP/DP($s1 + s2$) in Table 4). Thereby DP($s1 + s2$) means that the best solution is found by DP through combining $s1$ and $s2$. For instances in which both techniques reach solutions which present the same values for OCT, DP($s1 + s2$) is depicted. It can be observed that for $1 \leq \#P \leq 14$, the constraint-based approach obtains better solutions than DP for almost all of the instances. Moreover, for $15 \leq \#P \leq 18$, in some cases DP obtains solutions that are better than or equal to those obtained through CP, and in other cases CP obtains the

best solution. Furthermore, for $19 \leq \#P \leq 20$, DP obtains solutions that are better than or equal to those obtained through CP for all the instances. Furthermore, it seems that the solutions for $\#P = \{6, 7, 8, 9, 10 \text{ and } 12\}$ are largely optimized since they widely appear in the DP solutions. The results (cf. Table 4) show that our constraint-based approach, i.e. CP, is able to improve the solution obtained by DP in most of the cases. This shows that our constraint-based approach works efficiently in the generation of optimized enactment plans, and hence, for the automatic generation of optimized BPMN models.

In short, our data indicates that for the generation of optimized BP models, CP enables complex problems to be solved in a more efficient way than they would be through other alternative methods, such as DP or the manual specification of BP models.

Additionally, when $\#P$ increases, the complexity of the problem rises sharply, and hence the manual treatment of the problem would become almost inextricable. In contrast, when using our approach an optimized solution for large problems, such as for $\#P = 100$ can be obtained in only 10 min.

Threats to validity: There are several factors which may threaten the validity of our experiments for the attainment of generalizable conclusions:

- The specific characteristics of the considered example, i.e. the empirical evaluation only considers a concrete problem with a specific number of BP activities and specific relations between the BP activities.
- The way in which the optimal/optimized solutions for problems of a certain size are combined in order to obtain solutions for larger problems is specific for the considered example. In most cases, feasible solutions for larger problems can be generated by concatenating solutions to smaller problems through DP. However, the way in which solutions to problems of a given size can be combined to provide a solution to a larger problem depends on the type of problem considered.

8.2.2. *Use for simulation*

Our approach can be used for simulation purposes in the BP design and analysis phase in order to study the relevance of several parameters in the quality of the generated plans, e.g. resource availability. As an example, the relevance of the number of available resources for the travel agency problem is analyzed as follows.

Figure 12 shows the completion time of the best BP enactment plan (OCT) which is generated through our approach. In both graphics of Fig. 12, the OCT is shown depending on the number of resources of roles A and B. First, the considered resources are grouped according to $\#A$ (Fig. 12(a)). Secondly, the considered resources are grouped according to $\#B$ (Fig. 12(b)). It can be seen, in most cases, that the OCT greatly decreases as $\#A$ increases. Additionally, in most cases, the OCT remains almost the same when $\#B$ increases. Therefore, $\#A$ seems to be

much more influential than $\#B$ for the OCT, i.e. A is a more critical resource for the considered travel agency problem.

In Tables 5 and 6, the average percentages of use of the resources of roles A and role B, respectively, regarding the OCT, are shown. In all cases, for the same value of $\#P$, these percentages decrease as the number of resources of the associated role increases. Moreover, as expected, the average percentage of use of resources of role A is greater than the average percentage of use of resources of role B for the same value of $\#P$.

9. Related Work

Our approach makes a proposal for integrating P&S with BPMSs. Most related work on such an integration focuses on the enactment phase in order to make dispatching decisions as to which activity should be executed using a resource when it becomes free (dynamic scheduling),[43–48] while very few integrations are carried out during the modeling phase, as presented here.

Also related to our proposal is research on the generation of BP models, e.g. Refs. 9, 49–53. While the proposals of Refs. 49 and 50 provide the BP information through an execution/interchange language, XPDL, our approach, in turn, uses a declarative modeling language based on a formal logic (LTL). As stated, the usage of declarative specifications allows the user to specify `what` has to be done instead of `how`, thereby facilitating the human work involved and avoiding failures. In contrast to XPDL, where the user has to specify the model in an imperative way, in our proposal the generation of imperative BP models is automatically done by the system. In a related way, in Ref. 51, planning tools are used for the semiautomatic generation of BP models, by considering the knowledge introduced through BP Reengineering languages. In Ref. 51, they propose an object-oriented structure modeling tool that follows their own rule-based approach, while we propose the use of an extension of ConDec, a widely referenced language in the context of BPM (e.g. Refs. 26 and 55), which also allows a higher level of abstraction. Additionally, Ref. 52 proposes a planning formalism for the modeling of BPs through an SAP specification (Status and Action Management, SAM), which is a variant of PDDL. Unlike our work, neither the resource perspective nor the optimization of several instances are considered since in Ref. 52 each non-deterministic action (i.e. activity) cannot be repeated in the generated solution. Moreover, Ref. 53 presents a service-oriented approach which transforms high-level BP models into web services composition models. This approach uses UML to specify the BP models from an MDA point of view, which lacks an implementation view of BP models,[56] in contrast to ConDec, which is a graphical and specific language for the modeling of BPs. Furthermore, Ref. 9 proposes to refine BP models by combining learning and planning techniques, starting from processes which are not fully described. Unlike our work, Ref. 9 needs past process executions and examples provided by the user to apply learning techniques. Moreover,[9] does not consider the optimization of any

objective function in the generation of the plans. Furthermore, in Ref. 9 executable plans are generated, while we propose the generation of BP models which are specified in a standard language, i.e. BPMN, which can also be improved by business analysts if necessary.

Related to the combined used of declarative and imperative models, Ref. 57 proposes the use of declarative BP models for specifying processes independently of a particular environment in order to align optional process customizations. This information is complemented with imperative BP specifications which contain information related to the control flow of the processes, often specific to a given environment. In Ref. 57 both declarative and imperative BP models need to be (manually) specified, while in our proposal the optimized imperative models are automatically generated. Lastly, Ref. 58 analyzes the need of transitions between different BP modeling paradigms, i.e. declarative, imperative and hybrid proposals, supporting our proposal.

Additionally, there exist some proposals which could be used to generate optimized enactment plans for BPs from constraint-based process specifications. Specifically, Ref. 59 proposes the generation of a non-deterministic finite state automaton from constraint-based specifications based on linear temporal logic (LTL) which represents exactly all traces that satisfy the LTL formulas. When extending this approach by including estimates, the OCT of all the traces could then be calculated (e.g. Ref. 60). However, the big disadvantage following such an approach would be that it can lead to performance problems when confronted with large constraint-based models since the automaton generated for the concatenated LTL formulas is exponential with respect to the size of the formula,[61] and, unlike the proposed approach, no heuristic has been used. In a similar way, CLIMB[26] could be used to generate feasible traces, i.e. traces which meet all the constraints imposed by the declarative specification, and calculate its completion time. Then, the best traces could be selected. Unlike our approach, Ref. 26 does neither consider optimality nor resource availabilities. Therefore, this would only cover the planning part of our proposal, but not the scheduling aspects addressed by our approach.

There is some related work (e.g. Refs. 62–65) which is focused on the automated composition of web services using planning techniques, i.e. given a set of services which are published on the web and a goal, the aim is to generate a composition of the available services which satisfies the goal. Like our approach, dynamic process-based composition approaches also start from a declarative specification. However, while they only consider activities with preconditions and effects, we allow for an increased expressiveness through the ConDec language. Supporting the suitability of ConDec for specifying web services, the work[66] proposes the language DecSerFlow (which is a sister language of ConDec, i.e. both share the same concepts and tools) as a declarative service flow language. Moreover, unlike,[62–65] our approach considers multiple instances which are executed within a particular timeframe, which is fundamental for achieving global optimization.

10. Discussion and Limitations

In BP most environments, the Process Design & Analysis phase is manually carried out by business analysts, who must deal with several aspects, such as resource allocation, the activity properties and the relations between them, and may even have to handle the optimization of several objectives. Therefore, in some cases, the manual specification of BP models can consume great quantity of resources, cause failures, and lead to non-optimized models, resulting in a very complex problem.[9]

Hence, it should be emphasized that the automatic generation of BP models facilitates the human work in most cases, prevents failures in the developed BP models, and enables better optimization to be attained in the enactment phase.

Additionally, the specification of process properties in a declarative way allows the user to specify what is to be done, and the proposed AI-based tool is in charge of determining how it is to be done in order to satisfy the problem specifications, and to attain the optimization of certain objective functions.

Unlike conventional BPMN models, in our approach each generated model comprises the execution of a set of instances. Therefore, our approach always addresses, at least, global inter-instance optimization (even when optimization within each instance is not completely addressed). In this way, optimization over a set of instances is always addressed (e.g. the resources which are shared by the different instances are allocated in an optimized way by considering all instances to be executed). Additionally, in most cases, intra-instance optimization is also (completely or partially) addressed, as explained in Sec. 6.1.

Moreover, the automatic generation of BP models can deal with complex problems of great size in a simple way, as demonstrated in Sec. 8. Therefore, a wide study of several aspects can be carried out by simulation, such as those related to the requirement of resources of different roles, or the estimated completion time for the BP enactment, by generating several kinds of problems.

Furthermore, the proposed constraint-based approach can be used to efficiently solve further planning and scheduling problems which include similar relations between repeated activities, and which are unrelated to BP environments.

It should also be clarified that the BP models are generated for execution purposes, and hence clarity of meaning for the users of the generated models is not considered relevant in the current proposal.

Note that the generation of optimized enactment plans from constraint-based specifications is, from our point of view, the most challenging task of the proposed approach. Existing constraint-based BPMSs like Declare[32] could be used (in combination with planning) to decide which activities have to be done in the current state. In a related way, optimized enactment plans can be used to help users to find good/optimal ways to execute a declarative process by suggesting recommendations, as explained in one of our previous works (cf. Ref. 67). However, in the current work, the motivation for generating BPMN models from these enactment plans is two-fold: (1) visualize the generated enactment plan in a standard BP modeling language the users are familiar with, with the goal of allowing the

business analysts further improve the BPMN model when necessary, and (2) enable BP designs to be deployed into BPM systems and let their instances be automatically executed by existing (commercially available) state-of-the-art BPMSs (e.g. AristaFlow).

However, the proposed approach also presents a few limitations. First, the business analysts must deal with a new language for the constraint-based specification of BPs, therefore a period of training is required in order to let the business analysts become familiar with ConDec-R specifications. Secondly, the optimized BP models are generated by considering estimated values for the activity duration and resource availability, hence our proposal is only appropriate for processes in which the duration of the activities and the resource availability can be estimated. Nevertheless, in our approach, the BPMN model can be dynamically adapted during run-time by using replanning, and hence allowing for an increased flexibility (cf. Sec. 6.2). Moreover, ConDec-R specifications deal with both control-flow and resource perspectives, and also temporal data. Incorporating the non-temporal data perspective is subject to future work. Notice that already without non-temporal data many problems can be solved.

There are several objectives which can be considered in BPMS. In this work, we have considered minimizing the OCT only. However, this proposal can be extended in order to consider further objectives, such as cost or other temporal measures.

11. Conclusions and Future Work

This work presents a proposal for the automatic generation of BP models from constraint-based specifications which consider both control-flow and resource perspectives. To this end, several steps are developed: first, the definition of a suitable language, ConDec-R, which allows the constraint-based specification of BPs to be defined in a suitable form (both control-flow and resource perspectives are considered); secondly, a constraint-based proposal for planning and scheduling the BP activities in an optimized way in order to obtain optimized BP enactment plans; and third, an algorithm for generating the optimized BP model in BPMN from the optimized BP enactment plan. An example, the travel agency problem, is given in order to clarify the proposed approach. To validate its quality compared to existing methods like DP, different performance measures related to a range of test models of varying complexity are analyzed. The results indicate that, although the optimization of process execution is a highly constrained problem, the proposed approach produces a satisfactory number of suitable solutions.

As for future work, the proposed approach will be extended through the incorporation of further objective functions. Moreover, we will explore various constraint-based solving techniques and analyze their suitability for the generation of optimized BP plans. Furthermore, in the enactment phase, we intend to apply P&S techniques in order to replan the activities by considering the actual values of the parameters to validate the run-time flexibility of the approach. Moreover, the

development of a tool for the automatic generation of optimized BP models from ConDec-R specifications is currently under development. In addition, we intend to address an actual process enactment experiment in an organization according to the proposed method.

Acknowledgments

This work has been partially funded by the Spanish Ministerio de Ciencia e Innovación (TIN2009-13714) and the European Regional Development Fund (ERDF/FEDER).

Appendix A. Algorithms for Generating BPMN Models

In order to develop the algorithms to generate the BP models from the optimized enactment plans (cf. Algorithms A.1–A.5), certain related types are stated, as shown in Fig. A.1 (UML diagram). Note that at this point of the process the CSP variables are instantiated, and hence all the information is known (`nt` variable for each BP activity, `st` variable for each scheduling activity, resource in which each scheduling activity is executed, etc.). The types which appear in the UML diagram are as follows:

- $OptimizedPlan(acts, r, t)$: This represents the generated optimized enactment plan. Moreover, it contains the information related to the input problem. Specifically, this type contains properties regarding a set of roles r, a set of repeated activities (ConDec-R activities) $acts$, and a set of constraints which relate the repeated activities t.
- $RepeatedAct(role, dur, acts, nt)$: This represents the ConDec-R activities. Each repeated activity contains information about the required role (i.e. $role$), the estimated duration (i.e. dur), the set of scheduling activities which represent the execution of each BP activity (i.e. $acts$), and the number of times this repeated activity is executed (i.e. nt).
- $Role(resources)$: This represents a role, and it is composed of the set of resources available for this role.
- $Resource(acts)$: This represents a resource. This type contains properties regarding a list of scheduling activities which are executed in that resource, ordered by the start time.
- $Constraint(name)$: This represents the high-level relations which are given between the repeated activities. In order to consider the branched constraints (Sec. 3), two specializations are included to allow the relations between one source and several sinks (ConstraintSinks), and between several sources and one sink (ConstraintSources).[1] The method *includePred* of a template updates the

[1]Note that both ConstraintSinks and ConstraintSources can be used for specifying binary constraints.

information of the BPMN model by including the precedence relations which are implied by that template (more details are given later in this section during the presentation of the algorithms). For the generation of the BPMN model, the constraints are considered for the connection of the BPMN activities.

- $P\&SAct(st, et, res)$: This represents each execution of a repeated activity. This type contains properties regarding the start and the end times of the activity, together with the resource used by the scheduling activity (st, et and res respectively). Since each P&SAct is related to a specific BPMNAct in the resulting BPMN model, the P&SAct type provides the method *toBPMNAct* in order to obtain the related BPMNAct from a P&SAct (cf. Fig. A.1). This method is formalized as follows, where the symbol $\rightarrow$ is used to specify the output parameter: $toBPMNAct(a : P\&SAct) \rightarrow BPMNAct(a.res.lane.pool, a.res.lane, a.dur, a.st)$.
- $BPMNModel(pools, acts, seqFlows, gates)$: This represents the BPMN model that is generated. This model is composed of a set of pools $pools$, a set of BPMN activities $acts$, a set of sequence flows $seqFlows$, and a set of gates $gates$. It contains the function $createBPMN() \rightarrow BPMNModel(\emptyset, \emptyset, \emptyset, \emptyset)$ (i.e. this method returns an object of type BPMNModel in which all properties are empty sets).
- $BPMNAct(pool, lane, dur, st)$: This represents a BPMN activity. This type contains properties regarding the pool and the lane where the activity is allocated (i.e. $pool$ and $lane$ respectively), together with the estimated duration dur and start time st. It contains the following functions (cf. Fig. A.1): (1) $createBPMNAct(a : P\&SAct) \rightarrow BPMNAct$ $(a.res.lane.pool, a.res.lane, a.dur, a.st)$, which creates a BPMNAct from a P&SAct, and (2) $createBPMNAct(p : Pool, l : Lane, dur : int, st : int) \rightarrow BPMNAct(p, l, dur, st)$.
- $Pool(lanes, role)$: This represents a BPMN pool. Each pool is associated to a specific object of type Role $role$, and is composed of a set of objects of type Lane $lanes$. It contains the function $createPool(role : Role) \rightarrow Pool(lanes, role)$, where $lanes = \bigcup_{res \in role.resources} createLane(res)$, i.e. for each resource of that role, a related lane is created and included in the pool.
- $Lane(res)$: This represents a BPMN lane. Each lane is associated to a specific resource res. It contains the function $createLane(res : Resource) \rightarrow Lane(res)$.
- $Gate$: This represents a BPMN gate. In order to consider parallel merging gateways, a specialization, named ParallelM, is developed.
- $ParallelM(sources, sink)$: This represents a parallel merging gateway, together with the related input and output connections of the gateway. This type contains properties regarding a set of inputs $sources$, and one output $sink$. It contains the function $createParallelM(l : Set < BPMNAct >, a : BPMNAct) \rightarrow ParallelM(l, a)$.
- $SequenceFlow(a, b)$: This represents a precedence sequence flow between two BPMN activities, a and b. It contains the function $createSequenceFlow(a : BPMNAct, b : BPMNAct) \rightarrow SequenceFlow(a, b)$. Note that the connections between a BPMN activity and a gateway are stated in ParallelM objects.

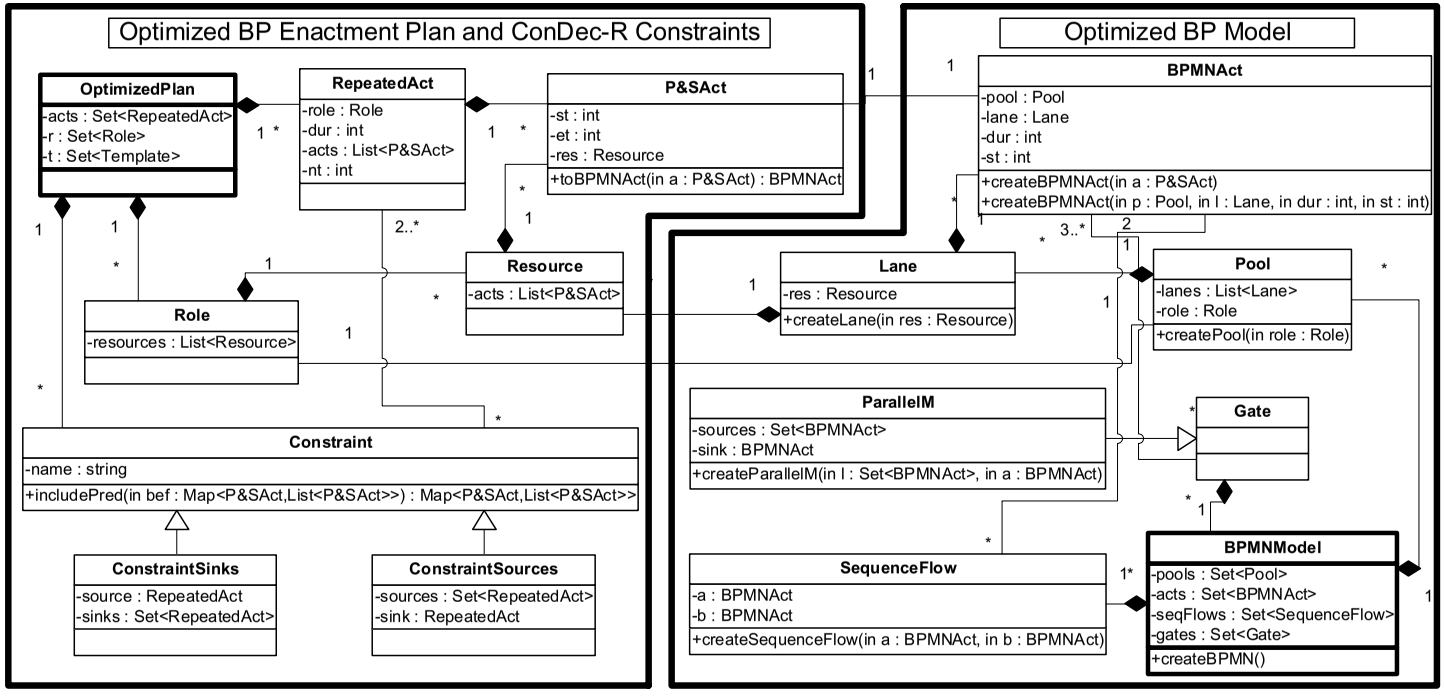


Fig. A.1. UML diagram of types for the optimized BPMN generation.

In Algorithms A.1–A.5, T⟨P⟩ represents the generic type T with the generic parameter instantiated to P. These algorithms are explained below.

The main algorithm, Algorithm A.1, constructs a BPMN model from an optimized BP enactment plan (cf. Definition 4.4) and a ConDec-R model (cf. Definition 3.1). From the enactment plan and the ConDec-R model, the input parameters of Algorithm A.1 can be stated, i.e. a sorted set of scheduling activities ordered by start time (i.e. *acts*); a set of the constraints which relate the repeated activities (i.e. *c*); and a set of the considered roles (i.e. *r*). Algorithm A.1 starts by creating

Algorithm A.1: Construct an Optimized BP Model from an Optimized BP Enactment Plan

input : SortedSet ⟨P&SAct⟩ *acts*, ordered by *st*
Set⟨Constraint⟩ *c*
Set⟨Role⟩ *r*
output: BPMNModel *bp*

1 $bp \leftarrow createBPMN()$;
2 $bp.pools \leftarrow \{createPool(role) \mid role \in r\}$;
3 $bp.acts \leftarrow \{createBPMNAct(a) \mid a \in acts\}$;
4 BPMNAct $start \leftarrow createBPMNAct(P_0, L_0, 0, 0)$;
5 BPMNAct $end \leftarrow createBPMNAct(P_0, L_0, 0, max_{a \in acts} a.et)$;
6 $bp.sequenceFlows \leftarrow \{createSequenceFlow(start, ini) \mid ini \in bp.acts, ini.st = 0\}$;
7 Map⟨P&SAct,Set⟨P&SAct⟩⟩ $pred \leftarrow CreateDependencies(acts, c, r)$;
8 **foreach** *psact in acts* **do**
9 **if** *pred(psact).size == 1* **then**
10 P&SAct $aPred \leftarrow pred(psact).get(0)$;
11 $bp.sequenceFlows \leftarrow bp.sequenceFlows \cup createSequenceFlow(toBPMNAct(aPred), toBPMNAct(psact))$;
12 **else**
13 Set⟨BPMNAct⟩ $inputs \leftarrow \{toBPMNAct(a) \mid a \in pred(psact)\}$;
14 $bp.gates \leftarrow bp.gates \cup createParallelM(inputs, toBPMNAct(psact))$;
15 Set⟨BPMNAct⟩ $finals \leftarrow \{toBPMNAct(a) \mid a \in P\&SAct, \neg\exists b \in P\&SAct, a \in pred(b)\}$;
16 **if** *finals.size == 1* **then**
17 BPMNAct $final \leftarrow finals.get(0)$;
18 $bp.sequenceFlows \leftarrow bp.sequenceFlows \cup createSequenceFlow(final, end)$;
19 **else**
20 $bp.gates \leftarrow bp.gates \cup createParallelM(finals, end)$;
21 **return** *bp*;

an empty BPMN model (cf. line 1). Moreover, a pool associated to each role is created, together with the corresponding lanes (line 2). In a similar way, a BPMN activity associated to each scheduling activity is created (line 3). The start and end activities of the model can be associated to any pool, which is represented by P_0 in Algorithm A.1, and to any lane, which is represented by L_0 in Algorithm A.1 (lines 4 and 5, respectively). In line 6, a sequence flow between the start BPMN activity and each BPMN activity whose estimated start time is equal to 0 is created through the *createSequenceFlow* method (cf. Fig. A.1). As explained, the *createSequenceFlow* method contains the parameters (1) a of type $BPMNAct$, and (2) b of type $BPMNAct$ as input, and creates a $SequenceFlow$ object which states a BPMN binary precedence relation starting in a and ending in b. After that, the map *pred* associates a set of direct predecessors (cf. Definition 5.2) to each scheduling activity by using the method $CreateDependencies$ (cf. Algorithm A.2, explained later in this section) in order to generate the BPMN model (line 7).[m]

Lines 8–14 establish the sequence flows and gateways between the BPMN activities in the following way: if the BPMN activity has only one direct predecessor, a sequence flow is included (lines 9–11); otherwise if the BPMN activity has several direct predecessors, a parallel merging gateway is included through the $createParallelM$ method (lines 12–14). As explained, the $createParallelM$ method contains the parameters (1) l of type $List\langle BPMNAct\rangle$, and (2) a of type $BPMNAct$ as input, and creates a $ParallelM$ object which states a BPMN parallel merging gateway (also including all the related connections) with contains all the BPMN activities of l as input and the BPMN activity a as output. In line 15, all the final activities are selected to be direct predecessors of the end activity. These activities are related by either a sequence flow, in the case that there is only one ending activity (lines 16–18); or by a parallel merging gateway, in the case that there are several ending activities (lines 19 and 20). Note that, as mentioned, parallel merging gateways (i.e. parallel gateways which have several sources and only one sink) need to be explicitly included in the resulting BPMN model, since they do not have the same meaning as several binary sequence flows from several sources and one sink. However, parallel splitting gateways (i.e. parallel gateways which have several sinks and only one source) do not need to be explicitly included in the resulting BPMN model since several binary sequence flows between one source activity and several sink activities have the same meaning as a parallel splitting gateway in the BPMN language.

As stated before, one of the most important aspects to be considered for this model generation are the precedence relations between the scheduling activities of the plan, which are managed by Algorithm A.2. As mentioned, these precedence relations are due to (1) resource constraints, i.e. the activities are allocated in the resources in a specific order in the generated enactment plan, and (2) ConDec-R

[m]The generic type *Map*⟨T1,T2⟩, which associates an object of type T2 to an object of type T1, is used.

Algorithm A.2: CreateDependencies

input : SortedSet⟨P&SAct⟩ *acts* ordered by *st*
Set⟨Constraint⟩ *constraints*
Set⟨Role⟩ *roles*
output: Map⟨P&SAct,Set⟨P&SAct⟩⟩ *directPredecessors*

```
1  Map⟨P&SAct,Set⟨P&SAct⟩⟩ allPredecessors ← ∅;
2  foreach r in roles do
3      foreach res in r.resources do
4          List⟨P&SAct⟩ actsRes ← res.acts;
5          foreach i in i:1..actsRes.size-1 do
6              allPredecessors(actsRes_{i+1}) ← {actsRes_i};
7  foreach c in constraints do
8      c.includePredecessors(allPredecessors);
9  Map⟨P&SAct,Set⟨P&SAct⟩⟩ indirectPredecessors ← ∅;
10 foreach act in acts do
11     directPredecessors(act) ← allPredecessors(act);
12     foreach p in allPredecessors(act) do
13         directPredecessors(act) ←
           directPredecessors(act) \ allPredecessors(p);
14         indirectPredecessors(act) ←
           indirectPredecessors(act) ∪ allPredecessors(p);
15     allPredecessors(act) ←
       allPredecessors(act) ∪ indirectPredecessors(act);
16 return directPredecessors;
```

constraints related to precedence between activities. Algorithm A.2 generates a map in which each scheduling activity is associated to a set of scheduling activities that are its direct predecessors (cf. Definition 5.2). For this, three maps are managed in this algorithm: (1) *directPredecessors*, which associates each scheduling activity to the set of its direct predecessors, (2) *indirectPredecessors*, which associates each scheduling activity to the set of its indirect predecessors (cf. Definition 5.3), and (3) *allPredecessors*, which associates each scheduling activity to the set of all its direct and indirect predecessors. In Algorithm A.2, first, the precedences required due to the use of the same resource are included (lines 2–6). Secondly, the precedences required due to the high-level relations (i.e. ConDec-R constraints) between the repeated activities which are stated in the model are included through the method `includePred` of each constraint (lines 7 and 8). Typically, unlike resource precedence relations, precedence relations due to ConDec-R constraints cannot be easily obtained. To this end, each ConDec-R template presents a method which is in

charge of determining the precedence relations which are given between the scheduling activities related to the repeated activities which are involved in that ConDec-R template. The mentioned method for some representative ConDec-R templates is detailed in Algorithms A.3–A.5. Lastly, the indirect predecessors are removed from the map *directPredecessors* in order to avoid redundant connections, by taking into account that the sorted set *acts* is ordered by *st*, and hence, the scheduling activities are managed from minor to major *st* in the external loop (lines 9–15). The properties of the maps which are used in Algorithm A.2 are demonstrated by Proposition A.1.

Proposition A.1. *This proposition contains two related parts*:

(a) *After executing lines* 1–9 *of Algorithm* A.2, (1) $directPredecessors = \emptyset$, (2) $indirectPredecessors = \emptyset$, *and* (3) $allPredecessors$ *associates each scheduling activity with all its direct predecessors* (*cf. Definition* 5.4) *and a subset of its indirect predecessors* (*cf. Definition* 5.5).

(b) *At the end of Algorithm* A.2, *as a result of executing lines* 10–15, $\forall\, act \in acts$: (1) *the map* $directPredecessors$ *associates act with exactly all its direct predecessors* (*cf. Definition* 5.4), (2) *the map* $indirectPredecessors$ *associates act with exactly all its indirect predecessors* (*cf. Definition* 5.5), *and* (3) *the map* $allPredecessors$ *associates act with exactly all its direct and indirect predecessors.*

Proof.

(a) The statements $directPredecessors = \emptyset$ and $indirectPredecessors = \emptyset$ hold since these maps have been only initialized. On one hand, the map $allPredecessors$ contains all the direct predecessors since all resource and ConDec relations are considered (lines 2–6 and 7–8, respectively). Moreover, some indirect predecessors have probably been included since redundant connections have not been avoided.

(b) (Mathematical Induction):

  (i) The base case: Since *acts* is a sorted set ordered by *st*, $acts_1$ does not have any predecessor (i.e. $allPredecessors(acts_1) = \emptyset$). Therefore, at the end of Algorithm A.2, as a result of executing lines 10–15: (1) $directPredecessors(acts_1) = \emptyset$, (2) $indirectPredecessors(acts_1) = \emptyset$, and (3) $allPredecessors(acts_1) = \emptyset$, i.e. the statement holds for the base case.

  (ii) The inductive step: If the statement holds $\forall_{i \in 1 \ldots n-1} acts_i$, then the statement also holds for $acts_n$.

    (1) $directPredecessors(acts_n)$: in line 11, this set is initialized with all the precedence relations of $acts_n$ which were previously obtained in the previous step until line 8, i.e. all its direct predecessors and a subset of its indirect predecessor (cf. Proposition A.1(a)). After that, for each direct

predecessor p of $acts_n$ (line 12), all the direct and indirect predecessors of p, which are therefore indirect predecessors of $acts_n$, are removed from the set $directPredecessors(acts_n)$ (line 13). Due to the induction hypothesis, the set $allPredecessors(p)$ is assumed to contain all direct and indirect predecessors of p since we are assuming that the statement holds for $\forall_{i \in 1 \ldots n-1} acts_i$ (due to p is a predecessor of $acts_n$ and $acts$ is a sorted set, then $\exists_{i \in 1 \ldots n-1} | acts_i = p$).

(2) $indirectPredecessors(acts_n)$: For each direct predecessor p of $acts_n$ (line 12), all the direct and indirect predecessors of p, which are therefore indirect predecessors of $acts_n$, are included in the set $indirectPredecessors(acts_n)$ (line 14).

(3) $allPredecessors(acts_n)$: After line 9, the set $allPredecessors(acts_n)$ contains all the direct predecessors and a subset of indirect predecessor of $acts_n$ (cf. Proposition A.1(a)). In order to ensure that all indirect predecessors are included, $allPredecessors(acts_n)$ is updated by considering all the indirect predecessors (line 15). □

Algorithm A.3: `includePred` method for the branched `Precedence` template with several source activities and one sink activity

input : Map⟨P&SAct,Set⟨P&SAct⟩⟩ *pred*
output: Map⟨P&SAct,Set⟨P&SAct⟩⟩ *pred*

1 Set⟨P&SAct⟩ $meet \leftarrow \{a_1 \mid a \in this.sources, a_1.et \leq this.sink_1.st\}$;
2 P&SAct $sel \leftarrow argmin_{a \in meet}(a.et)$;
3 $pred(this.sink_1) \leftarrow pred(this.sink_1) \cup sel$;
4 **return** *pred*;

With respect to the `includePred` method, some representative templates are selected for illustration purposes (other templates can be described in a similar way). In Algorithm A.3, the template regarding the branched `Precedence` template with several source activities and one sink activity (i.e. it is modeled by a *ConstraintSources* object, cf. Fig. A.1) is shown. The location of a branched precedence template between several sources and one sink implies that the first execution of at least one of the sources must finished before the start of the first execution of the sink. In line 1, the set of scheduling activities which comply with the Precedence template (i.e. the first executions of the sources which end before the start of the first execution of the sink) are included in the set *meet*. At least one scheduling activity will be included in this set since the Precedence template is satisfied, however it may be possible to find more than one. In order to generate a BPMN model which is compatible with both the optimized enactment plan and the ConDec-R specification, as is the purpose of our approach, any scheduling activity of the set *meet* can be selected to be the predecessor of the sink in the

BPMN model. One scheduling activity of the set *meet* is then selected to be the predecessor of the sink. Specifically, the scheduling activity which presents more slack is selected (line 2) in order to construct a robust BPMN model. In line 3, the selected predecessor is included in the map, and is associated to the predecessors of the first execution of the sink. The fact that an activity B can start after another activity A has finished (ES, default option), is stated by including A in the set *pred* of B (line 3) of Algorithm A.4.

Algorithm A.4: `includePred` method for the branched `Alternate Precedence` Template with several source activities and one sink activity

input : Map⟨P&SAct,Set⟨P&SAct⟩⟩ *pred*
output: Map⟨P&SAct,Set⟨P&SAct⟩⟩ *pred*

1 Set⟨P&SAct⟩ $meet \leftarrow \{a_1 \mid a \in this.sources, a_1.et \leq this.sink_1.st\}$;
2 P&SAct $sel \leftarrow argmin_{a \in meet}(a.et)$;
3 $pred(this.sink_1) \leftarrow pred(this.sink_1) \cup sel$;
4 **foreach** *i in 2..this.sink.nt* **do**
5 Set⟨P&SAct⟩ $meet \leftarrow \{a_j \mid a \in this.sources, j \in 1..a.nt, this.sink_{i-1}.et \leq a_j.st \wedge a_j.et \leq this.sink_i.st\}$;
6 P&SAct $sel \leftarrow argmax_{a \in meet}((a.st - this.sink_{i-1}.et) + (this.sink_i.st - a.et))$;
7 $pred(sel) \leftarrow pred(sel) \cup this.sink_{i-1}$;
8 $pred(this.sink_i) \leftarrow pred(this.sink_i) \cup sel$;
9 **return** *pred*;

The branched `AlternatePrecedence` template between several sources and one sink implies that before the execution of the sink, at least one of the sources must be executed, and between each two executions of the sink, at least one of the sources must be executed. As discussed, there exist two variants for the same temporal relation, which are represented by adding SS or ES at the end of the name of the template. In the `AlternatePrecedence` template, two temporal relations must be indicated: first, what "sink before source" means, and secondly, what "source before sink" means. Therefore, the branched template `AlternatePrecedenceES-ES` (default option) specifies that "sink before source" means that the **end** time of the sink must be less than or equal to the **start** time of the source, and "source before sink" means that the **end** time of the source must be less than or equal to the **start** time of the sink. The `includePred` method for the branched template `AlternatePrecedenceES-ES` with several source activities and one sink activity (i.e. it is modeled by a *ConstraintSources* object, cf. Fig. A.1) is shown in Algorithm A.4. For lines 1–3, the idea is the same as that in Algorithm A.3. Moreover, between each two successive executions of the sink, $sink_{i-1}$ and $sink_i$, one scheduling activity must be executed. Several scheduling activities related to the sources

can meet this condition (line 5). As before, the scheduling activity which presents more slack is selected (line 6) to be the predecessor of $sink_i$ (line 8), and at the same time $sink_{i-1}$ is selected as the predecessor of the selected scheduling activity.

Algorithm A.5: `includePred` method for the branched `Alternate PrecedenceSS-ES` Template with several source activities and one sink activity

input : Map⟨P&SAct,Set⟨P&SAct⟩⟩ *pred*
output: Map⟨P&SAct,Set⟨P&SAct⟩⟩ *pred*

1 Set⟨P&SAct⟩ $meet \leftarrow \{a_1 \mid a \in this.sources, a_1.et \leq this.sink_1.st\}$;
2 P&SAct $sel \leftarrow argmin_{a \in meet}(a.et)$;
3 $pred(this.sink_1) \leftarrow pred(this.sink_1) \cup sel$;
4 **foreach** *i in 2..this.sink.nt* **do**
5 Set⟨P&SAct⟩ $meet \leftarrow \{a_j \mid a \in this.sources, j \in 1..a.nt, this.sink_{i-1}.\mathbf{st} \leq a_j.st \wedge a_j.et \leq this.sink_i.st\}$;
6 P&SAct $sel \leftarrow argmax_{a \in meet}((a.st - this.sink_{i-1}.et) + (this.sink_i.st - a.et))$;
7 $pred(sel) \leftarrow pred(sel) \cup \mathbf{pred}(this.sink_{i-1})$;
8 $pred(this.sink_i) \leftarrow pred(this.sink_i) \cup sel$;
9 **return** *pred*;

In a similar way, the branched template `AlternatePrecedenceSS-ES` specifies that "sink before source" means that the **start** time of the sink must be less than or equal to the **start** time of the source, and "source before sink" means that the **end** time of the source must be less than or equal to the **start** time of the sink. The `includePred` method for the branched template `AlternatePrecedenceSS-ES` with several source activities and one sink activity (i.e. it is modelled by a *ConstraintSources* object, cf. Fig. A.1) is shown in Algorithm A.5. This algorithm is identical to Algorithm A.4, except for line 7. As mentioned earlier, the fact that an activity B can start after another activity A has finished (ES, default option), is stated by including A in the set *pred* of B. Additionally, the fact that an activity B can only start after another activity A has **started**, label SS, is stated by including the set $pred(A)$ in the set *pred* of B, as can be seen in line 7 of Algorithm A.5.

The complexity analysis of all the algorithms previously described is included in A.1, and the equivalence between the definitions given in Sec. 5 and the algorithms included in this section is detailed in A.2.

A.1. *Complexity analysis*

This section presents the complexity analysis of the algorithms previously described.

Proposition A.2. *If implemented properly, the worst-case time complexity of Algorithm* A.3 *is* $O(n)$, *where* n *is the number of Repeated Activities of the problem.*

Proof. The worst-case time complexity of line 1 is $O(n)$, since $\#sources \leq n$. The worst-case time complexity of line 2 is also $O(n)$, since $\#meet \leq n$. The time complexity of line 3 is constant. Therefore, the worst-case time complexity of Algorithm A.3 is $O(n) + O(n) + \Theta(1)$, equal to $O(n)$. □

Proposition A.3. *If implemented properly, the worst-case time complexity of Algorithms* A.4 *and* A.5 *is* $O(n \times nt)$*, where* n *is the number of Repeated Activities of the problem, and* nt *is the maximum number of times that a repeated activity is executed.*

Proof. The worst-case time complexity of line 1 is $O(n)$, since $\#sources \leq n$. The worst-case time complexity of line 2 is also $O(n)$, since $\#meet \leq n$. The time complexity of line 3 is constant. The worst-case time complexity of lines 4–8 is $O(n \times nt)$ since lines 5–7 (with complexity $O(n)$ from the proof of Proposition A.1) are executed at most nt times. Therefore, the worst-case time complexity of Algorithms A.4 and A.5 is $O(n) + O(n) + \Theta(1) + O(n \times nt)$ equal to $O(n \times nt)$. □

Proposition A.4. *If implemented properly, the worst-case time complexity of Algorithm* A.2 *is* $O(c \times n \times nt + nps^2)$*, where* n *is the number of Repeated Activities of the problem,* nt *is the maximum number of times that a repeated activity is executed,* c *is the number of constraints that appear in the definition of the problem, and* nps *is the number of scheduling activities in the optimized plan.*

Proof. The time complexity of lines 1–5 is $\Theta(nps)$, since each scheduling activity is considered exactly once (each activity uses a specific resource of a specific role). The worst-case time complexity of lines 6 and 7 is $O(c \times n \times nt)$, since the worst-case time complexity of the method *includePred* is $O(n \times nt)$ (Proposition A.2), and this method is invoked c times. The worst-case time complexity of lines 9–12 is $O(nps^2)$, since for each scheduling activity, its predecessors (at most, nps) are considered. Therefore, the worst-case time complexity of Algorithm A.2 is $O(c \times n \times nt + nps^2)$.

□

Proposition A.5. *If implemented properly, the worst-case time complexity of Algorithm* A.1 *is* $O(c \times n \times nt + nps^2)$*, where* n *is the number of Repeated Activities of the problem,* nt *is the maximum number of times that a repeated activity is executed,* c *is the number of constraints that appear in the definition of the problem, and* nps *is the number of scheduling activities in the optimized plan.*

Proof. The worst-case time complexity of line 1 is $O(c \times n \times nt + nps^2)$, by Proposition A.3. The worst-case time complexity of line 3 is $O(n)$, since $\#role \leq n$. The time complexity of line 4 is $\Theta(nps)$. The worst-case time complexity of lines 6 and 15 is $O(nps)$. The time complexity of lines 7–13 is $\Theta(nps)$. Therefore, the worst-case time complexity of Algorithm A.1 is $O(c \times n \times nt + nps^2)$. □

A.2. *Equivalence between definitions in Sec.* 5 *and algorithms in Appendix A*

The definitions which appear in Sec. 5 define the generation of BPMN models from a formal point of view, while the algorithms in the appendix describe the same generation from an implementation point of view (e.g. some information has been duplicated and ordered in a different way to facilitate the implementation). Therefore, the types used in the appendix are not exactly the same than the definitions given throughout the paper. However, both Definition 5.6 and Algorithm A.1 generate the same BPMN model from the same input ConDec-R model and from the same solution, as explained in the following.

Regarding the input parameters: (1) Definition 5.6 includes (i) a ConDec-R process model $CR = (Acts, CBP, Res)$ and (ii) a solution S to a CSP-ConDec problem as input parameters, while (2) Algorithm A.1 includes (i) a sorted set of objects of type P&SAct, $acts$, ordered by st; (ii) a set of objects of type Constraint, c; (iii) and a set of objects of type Role, r. The input parameters of Algorithm A.1 can be obtained from the input parameters of Definition 5.6 as follows:

- the information for each P&SAct a_i in $acts$ (i.e. st, et and res, cf. UML diagram of Fig. 12) is taken so that $a_i \cdot st = S^{st(a_i)}$, $a_i \cdot et = S^{et(a_i)}$, $a_i \cdot res = S^{res(a_i)}$ (i.e. from the CSP variables which are related to the scheduling activity a_i in the solution to the CSP-ConDec problem, cf. Definition 4.3).
- $c = C_{\text{BP}}$, i.e. the set of objects of type Constraint, c, contains the same constraints which are included in the ConDec-R model.
- $r = \{role, (role, \#role) \in Res, role.resources = \{res_k, k \in [1 \dots \#role], res_k \cdot acts = \{a_i \in acts, a_i \cdot res = res_k\}\}\}$, i.e. there is one object of type Role in r for each role in Res. Moreover, for each object of type Role $role$, there are $\#role$ objects of type Resource in the list $resources$ of $role$. Each object of type Resource res_k contains, in turn, an ordered list of objects of type P&SAct which are related to the P&S activities which are allocated in that resource.

Furthermore, the equivalence between Definition 5.6 and Algorithm A.1 is detailed as follows:

- (1) in Definition 5.6 is equivalent to line 2 in Algorithm A.1, i.e. both include a pool for each role in the resulting BPMN model.
- (2) in Definition 5.6 is equivalent to lines 3–5 in Algorithm A.1, i.e. both include a BPMN activity for each BP activity, plus one activity related to the start BPMN activity, plus one activity related to the end BPMN activity.
- (3) in Definition 5.6 is equivalent to lines 6–20 in Algorithm A.1. The set $predecessors$ in Definition 5.6 (cf. (3)) states the direct precedences between all the activities, including: the precedences between the P&S activities, and the precedences in which the BPMN *start* and *end* activities are involved. However, in Algorithm A.1, the precedences in which the BPMN *start* and *end* activities

are involved are handled differently from the rest of activities due to implementation reasons. The equivalence between (3) in Definition 5.6 and lines 6–20 in Algorithm A.1 is explained as follows:

— (3)(i) in combination with (3)(a) in Definition 5.6 is equivalent to line 6 in Algorithm A.1, i.e. both include sequence flows between the BPMN start activity and all the activities whose start time is equal to 0 (i.e. are direct successors of the start activity). Specifically, in (3)(i) all the precedence relations in which the start activity is involved are included in the set $Predecessors$, and in (3)(a) sequence flows between the start activity and all its direct successors are included. However, in Algorithm A.1, these sequence flows are directly included in line 6. Note that the start activity is never involved in parallel merging gateways since this activity does not have any predecessor.
— (3)(ii) in combination with (3)(a) and 3(b) in Definition 5.6 is equivalent to lines 15–20 in Algorithm A.1, i.e. both include either a sequence flow or a parallel merging gateway between the BPMN end activity and all its direct predecessors. Specifically, in (3)(ii) all the precedence relations in which the end activity is involved are included in the set $Predecessors$. In a similar way, in line 15 of Algorithm A.1, all the direct predecessors of the end activity are stored in the set finals. Then, if there is only one direct predecessor for the end activity, a sequence flow between this predecessor and the end activity is included (lines 16–18 in Algorithm A.1, (3)(a) in Definition 5.6). However, if there are more than one direct predecessor for the end activity, a parallel merging gateway between all predecessors and the end activity is included (lines 19 and 20 in Algorithm A.1, (3)(b) in Definition 5.6). Note that the end activity may be involved in parallel merging gateways since this activity may have more than one predecessor.
— (3)(iii) in combination with (3)(a) and 3(b) in Definition 5.6 is equivalent to lines 7–14 in Algorithm A.1, i.e. both include either a sequence flow or a parallel merging gateway between each activity and all its direct predecessors. Specifically, in (3)(iii) all the precedence relations in which each activity is involved are included in the set $Predecessors$. In a similar way, in line 7 of Algorithm A.1, all the direct predecessors of each activity are stored in the map *pred*. For this, Algorithm A.2 (i.e. CreateDependencies) is used. Algorithm A.2 is equivalent to (3)(iii) since, as demonstrated in Proposition A.1, Algorithm A.2 returns a map which relates each activity with its direct predecessors (cf. Definition 5.4). Then, for each activity (line 8 in Algorithm A.1) if there is only one direct predecessor for that activity, a sequence flow between this predecessor and the activity is included (lines 9–11 in Algorithm A.1, (3)(a) in Definition 5.6). However, if there are more than one direct predecessor for that activity, a parallel merging gateway between all its predecessors and the activity is included (lines 12–14 in Algorithm A.1,

(3)(b) in Definition 5.6). Note that each activity may be involved in parallel merging gateways, since it may have more than one predecessor.

References

1. M. Weske, *Business Process Management: Concepts, Languages Architectures* (Springer, Berlin, Germany, 2007).
2. M. Dumas, W. van der Aalst and A. ter Hofstede, *Process-Aware Information Systems: Bridging People and Software Through Process Technology* (Wiley-Interscience, Hoboken, NJ, 2005).
3. G. Group, Leading in Times of Transition: The 2010 CIO Agenda (EXP Premier Report No. January 2010), in *Report, Gartner, Inc* (2010).
4. W. van der Aalst, A. ter Hofstede and M. Weske, Business process management: A survey, in *Proc. BPM* (2003), pp. 1–12.
5. R. Aguilar-Savén, Business process modelling: Review and framework, *Int. J. Prod. Econom.* **90**(2) (2004) 129–149.
6. J. Barjis and A. Verbraeck, The relevance of modeling and simulation in enterprise and organizational study, in *Proc. EOMAS* (2010), pp. 15–26.
7. H. Reijers, *Design and Control of Workflow Processes* (Springer-Verlag Berlin, Heidelberg, 2003).
8. N. Russell, W. van der Aalst, A. ter Hofstede and D. Edmond, Workflow resource patterns: Identification, representation and tool support, in *Proc. CAiSE* (2005), pp. 216–232.
9. H. M. Ferreira and D. R. Ferreira, An integrated life cycle for workflow management based on learning and planning, *Int. J. Coop. Inform. Syst.* **15**(4) (2006) 485–505.
10. J. Wainer, F. Bezerra and P. Barthelmess, Tucupi: A flexible workflow system based on overridable constraints, in *Proc. SAC* (2004), pp. 498–502.
11. M. Pesic, M. Schonenberg, N. Sidorova and W. van der Aalst, Constraint-based workflow models: Change made easy, in *OTM Conferences (1)* (2007), pp. 77–94.
12. I. Rychkova, G. Regev and A. Wegmann, High-level design and analysis of business processes: The advantages of declarative specifications, in *Proc. RCIS* (2008), pp. 99–110.
13. D. Fahland, D. Lübke, J. Mendling, H. Reijers, B. Weber, M. Weidlich and S. Zugal, Declarative versus imperative process modeling languages: The issue of understandability, in *Proc. BPMDS and EMMSAD* (2009), pp. 353–366.
14. D. Fahland, J. Mendling, H. Reijers, B. Weber, M. Weidlich and S. Zugal, Declarative versus imperative process modeling languages: The issue of maintainability, in *Proc. BPM Workshops* (2010), pp. 477–488.
15. P. Pichler, B. Weber, S. Zugal, J. Pinggera, J. Mendling and H. Reijers, Imperative versus declarative process modeling languages: An empirical investigation, in *Proc. ER-BPM* (2011), pp. 383–394.
16. M. Ghallab, D. Nau and P. Traverso, *Automated Planning: Theory and Practice* (Morgan Kaufmann, Amsterdam, 2004).
17. P. Brucker and S. Knust, *Complex Scheduling (GOR-Publications)* (Springer-Verlag, New York, Inc., Secaucus, NJ, USA, 2006).
18. F. Rossi, P. van Beek and T. Walsh (eds.), *Handbook of Constraint Programming* (Elsevier, New York, USA, 2006).
19. M. Salido, Introduction to planning, scheduling and constraint satisfaction, *J. Intell. Manuf.* **21**(1) (2010) 1–4.

20. Business Process Model and Notation (BPMN), Version 2.0., http://www.omg.org/spec/BPMN/2.0/ (2011) [Online; accessed 3-October-2011].
21. C. Ouyang, W. van der Aalst, M. Dumas and A. ter Hofstede, Translating BPMN to BPEL (2006).
22. Web Services Business Process Execution Language Version 2.0: OASIS Standard, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html (2007) [Online; accessed 3-October-2011].
23. M. Reichert and B. Weber, *Enabling Flexibility in Process-Aware Information Systems* (Springer, 2012).
24. M. Pesic and W. van der Aalst, A declarative approach for flexible business processes management, in *Proc. BPM Workshops* (2006), pp. 169–180.
25. W. van der Aalst, M. Pesic and H. Schonenberg, Declarative workflows: Balancing between flexibility and support, *Comput. Sci. — Res. Develop.* **23**(2) (2009) 99–113.
26. M. Montali, Specification and verification of declarative open interaction models: A logic-based approach, Ph.D. thesis, Department of Electronics, Computer Science and Telecommunications Engineering, University of Bologna, 2009.
27. P. Dourish, J. Holmes, A. MacLean, P. Marqvardsen and A. Zbyslaw, Freeflow: Mediating between representation and action in workflow systems, in *Proc. CSCW* (1996), pp. 190–198.
28. J. Wainer and F. De Lima Bezerra, Constraint-based flexible workflows, in *Proc. CRIWG* (2003), pp. 151–158.
29. R. Lu, S. Sadiq, V. Padmanabhan and G. Governatori, Using a temporal constraint network for business process execution, in *Proc. ADC* (2006), pp. 157–166.
30. W. M. P. van der Aalst and M. Pesic, DecSerFlow: Towards a Truly Declarative Service Flow Language, in *LNCS 4184* (2006), pp. 1–23.
31. I. Barba and C. Del Valle, A constraint-based approach for planning and scheduling repeated activities, in *Proc. COPLAS* (2011), pp. 55–62 [Online; http://icaps11.icaps-conference.org/proceedings/coplas/coplas2011-proceedings.pdf; accessed 3-October-2011].
32. Declare, http://www.win.tue.nl/declare/ (2013) [Online; accessed 11-February-2013].
33. J. F. Allen, Maintaining knowledge about temporal intervals, in *Communications of the ACM* (1983), pp. 832–843.
34. I. Barba and C. Del Valle, Filtering Rules for ConDec Templates — Pseudocode and Complexity, http://www.lsi.us.es/∼quivir/irene/FilteringRulesforConDec Templates.pdf (2011) [Online; accessed 3-October-2011].
35. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (W. H. Freeman & Co., New York, NY, USA, 1979).
36. C. Courbis and A. Finkelstein, Weaving aspects into web service orchestrations, in *Proc. ICWS 2005* (2005), pp. 219–226.
37. S. Zugal, P. Soffer, J. Pinggera and B. Weber, Expressiveness and understandability considerations of hierarchy in declarative business process models, in *Proc. BMMDS/EMMSAD 2012* (2012), pp. 167–181.
38. M. La Rosa, M. Dumas, R. Uba and R. Dijkman, Merging business process models, in *Proc. OTM*, Vol. 6426(1) (2010), pp. 96–113.
39. W. Nuijten and E. Aarts, Sequencing with earliness and tardiness penalties: A review, *Eur. J. Oper. Res.* **90**(2) (1996) 269–284.
40. AristaFlow BPM Suite BPM09-Demo, http://www.uni-ulm.de/einrichtungen/arista flow-forum/screencasts.html (2009) [Online; accessed 05-February-2013].
41. R. Bellman, *Dynamic Programming* (Princeton University Press, Princeton, NJ, 1957).

42. Comet Downloads, http://dynadec.com/support/downloads/ (2010) [Online; accessed 3-October-2011].
43. J. Zhao and E. Stohr, Temporal workflow management in a claim handling system, *SIGSOFT: Software Engineering Notes* **24**(2) (1999) 187–195.
44. J. Son and M. Kim, Improving the performance of time-constrained workflow processing, *J. Syst. Software* **58**(3) (2001) 211–219.
45. B. Ha, J. Bae, Y. Park and S. Kang, Development of process execution rules for workload balancing on agents, *Data Knowledge Eng.* **56**(1) (2006) 64–84.
46. I. Barba and C. Del Valle, Planning and scheduling of business processes in run-time: A repair planning example, *Information Systems Development* (2011), pp. 75–87.
47. S. Rhee, N. Cho and H. Bae, Increasing the efficiency of business processes using a theory of constraints, *Inform. Syst. Frontiers* **12**(4) (2010) 443–455.
48. C. Tsai, K. Huang, F. Wang and C. Chen, A distributed server architecture supporting dynamic resource provisioning for BPM-oriented workflow management systems, *J. Syst. Software* **83**(8) (2010) 1538–1552.
49. F. S. R. Alves, K. F. Guimares and M. A. Fernandes, Integrating planning and scheduling based on genetic algorithms to an workflow system, in *Proc. CEC* (2008), pp. 3766–3775.
50. A. González-Ferrer, J. Fernández-Olivares and L. Castillo, JABBAH: A Java application framework for the translation between business process models and HTN, in *Proc. ICKEPS* (2009), pp. 28–37.
51. M. R-Moreno, D. Borrajo, A. Cesta and A. Oddi, Integrating planning and scheduling in workflow domains, *Expert Syst. Appl.* **33**(2) (2007) 389–406.
52. J. Hoffmann, I. Weber and F. Kraft, SAP speaks PDDL, in *Proc. AAAI* (2010), pp. 1096–1101.
53. V. De Castro and E. Marcos, Towards a service-oriented MDA- based approach to the alignment of business processes with it systems: From the business model to a web service composition model, *Int. J. Coop. Inform. Syst.* **18**(2) (2009) 225–260.
54. W. van der Aalst, Patterns and XPDL: A critical evaluation of the XML process definition language, in *QUT Technical Report FIT-TR-2003-06* (2003), pp. 1–30.
55. L. Ly, S. Rinderle and P. Dadam, Integration and verification of semantic constraints in adaptive process management systems, *Data Knowledge Eng.* **64(1)** (2008) 3–23.
56. M. Owen and J. Raj, BPMN and business process management introduction to the new business process modeling standard, http://www.omg.org/bpmn/Documents/6AD5D16960.BPMN_and_BPM.pdf (2003) [Online; accessed 3-October-2011].
57. I. Rychkova, G. Regev and A. Wegmann, Using declarative specifications in business process design, *Int. J. Comput. Sci. Appl.* **5**(3b) (2008) 45–68.
58. F. Caron, J. Vanthienen, An exploratory approach to process lifecycle transitions from a paradigm-based perspective, in *Proc. BPMDS and EMMSAD* (2011), pp. 178–185.
59. M. Pesic, Constraint-based workflow management systems: Shifting control to users, Ph.D. thesis, Technische Universiteit Eindhoven, Eindhoven, 2008.
60. W. van der Aalst, M. Schonenberg and M. Song, Time prediction based on process mining, *Inform. Syst.* **36**(2) (2011) 450–475.
61. M. Montali, M. Pesic, W. van der Aalst, F. Chesani, P. Mello and S. Storari, Declarative specification and verification of service choreographiess, *ACM Trans. Web* **4**(1) Number 3 (2010).
62. H. Schuschel and M. Weske, Triggering replanning in an integrated workflow planning and enactment system, in *Proc. ADBIS*, Vol. 3255 (2004), pp. 322–335.
63. H. Meyer and M. Weske, Automated service composition using heuristic search, in *Proc. BPM 2006*, Vol. 4102 (2006), pp. 81–96.

64. L. Zeng, A. Ngu, B. Benatallah, R. Podorozhny and H. Lei, Dynamic composition and optimization of web services, *Distr. Parallel Databases.* **24** (2008) 45–72.
65. Y. Syu, S. P. Ma, J. Y. Kuo and Y. Y. Fan Jiang, A survey on automated service composition methods and related techniques, in *Proc. SCC 2012* (2012), pp. 290–297.
66. W. M. P. van der Aalst and M. Pesic, DecSerFlow: Towards a truly declarative service flow language, in *Proc. WS-FM 2006* (2006), pp. 1–23.
67. I. Barba, B. Weber and C. Del Valle, DecSerFlow: Supporting the optimized execution of business processes through recommendations, in *Proc. BPI 2011*, Vol. 99 (2012), pp. 135–140.