A Formal model for configurable business process with optimal cloud resource allocation

Abderrahim Ait Wakrime, Souha Boubaker, Slim Kallel, Emna Guermazi, Walid Gaaloul

Journal of Universal Computer Science, vol. 27, no. 7 (2021), 693-713
submitted: 19/5/2020, accepted: 26/5/2021, appeared: 28/7/2021

A Formal Model for Configurable Business Process with Optimal Cloud Resource Allocation

Abderrahim Ait Wakrime
(Computer Science Department, Faculty of Sciences, Mohammed V University in Rabat, Morocco
https://orcid.org/0000-0001-9215-6309, abderrahim.aitwakrime@fsr.um5.ac.ma)

Souha Boubaker
(Samovar, Télécom SudParis, Institut Polytechnique de Paris, Paris, France
souha.boubaker@gmail.com)

Slim Kallel
(ReDCAD, University of Sfax, Sfax, Tunisia
https://orcid.org/0000-0002-2824-167X, slim.kallel@fsegs.usf.tn)

Emna Guermazi
(University of Sfax, Sfax, Tunisia
emna.guermazi@usf.tn)

Walid Gaaloul
(Samovar, Télécom SudParis, Institut Polytechnique de Paris, Paris, France
https://orcid.org/0000-0003-0451-532X, walid.gaaloul@telecom-sudparis.eu)

Abstract: In today's competitive business environments, organizations increasingly need to model and deploy flexible and cost effective business processes. In this context, configurable process models are used to offer flexibility by representing process variants in a generic manner. Hence, the behavior of similar variants is grouped in a single model holding configurable elements. Such elements are then customized and configured depending on specific needs. However, the decision to configure an element may be incorrect leading to critical behavioral errors. Recently, process configuration has been extended to include Cloud resources allocation, to meet the need of business scalability by allowing access to on-demand IT resources. In this work, we propose a formal model based on propositional satisfiability formula allowing to find correct elements configuration including resources allocation ones. In addition, we propose to select optimal configurations based on Cloud resources cost. This approach allows to provide the designers with correct and cost-effective configuration decisions.

Keywords: Configurable Business Process, Formal Methods, Cloud Resources, Propositional Satisfiability.
Categories: D.2.1, D.2.4
DOI: 10.3897/jucs.70978

1 Introduction

Configurable business process models offer the possibility of representing similar processes with common and variable components. Thanks to this flexibility, companies

dispose increasingly of a wide range of design options. These options are easily created by altering values of variable components. These variable components, namely configurable elements, can be configured as per the organizational specific requirements within each company. For instance, to produce variants of the same group of business process models, the process designer has the possibility to select components to integrate in the model and skip components that are deemed irrelevant.

Running a business process within a company implies taking into consideration underlying exploitation costs mainly virtual resources. On-demand cloud computing solutions were conceived to offer companies flexible and highly available and scalable infrastructures, therefore allowing control over induced business turnover. In this context, two issues arise: (i) the configurable elements may have many configuration options with complex interdependencies between them. Undertaking the task of correctly deciding and applying manually the correct configuration is a tedious and a highly error-prone exercise. Correct configuration is defined here as the set of options selected that generate a process variant that runs without structural and execution errors.(ii) Deciding which cloud resource description is optimal for a certain business process relies on a number of considerations amongst which : the particular need of the process in order to be efficient and available during execution-which depends mainly on the structure of the variant process, the price offer of the cloud solution, and other constraints that would steer the optimization.

Hence, a mathematical model is needed to formulate the optimization problem. Previous research has addressed these two issues from various angles, e.g., [Rosemann and Van der Aalst 2007, Recker et al. 2005, Hallerbach et al. 2010, Kumar and Yao 2012] tackle the complexity of the design phase of a process model, while in [GröNer et al. 2013, Assy and Gaaloul 2016, Asadi et al. 2014, La Rosa et al. 2009], authors suggest guiding configuration and supporting domain-based constraints. Some other approaches are more concerned about ensuring the correctness of the configuration as in [van der Aalst et al. 2010, Hallerbach et al. 2009]. In overall, these approaches suffer from the state space explosion problem and forsake the costs incurred by BP deployment and the configuration of the required resources of this deployment.

In this paper, we use the satisfiability problem (SAT) to address these issues by improving the efficiency of business processes configuration and properly identify the cloud resources requirements. During the last two decades, SAT has undergone a very important development in terms of SAT (propositional satisfiability) solvers. These solvers consist in deciding whether a formula is satisfiable or not. Research in this area has led to the advent of modern SAT solvers capable of solving problems containing millions of variables. This resolution efficiency allowed this technology to be exported to other application areas. Indeed, several problems arising from classic planning, cryptography, product configuration, etc. were encoded to SAT. Thanks to the advance in SAT resolution, SAT solvers are recently becoming the tool for tackling more and more practical problems. SAT considers Boolean function to study truth assignments (assignments of 0 or 1 to variables where the value 1 means a statement is true). Several works employed SAT in a number of applications to solve domain specific problems and hence obtained interesting results.

In our previous work [Ait Wakrime et al. 2019], we proposed a translation rules of a configurable business process into a SAT model to generate all correct configurations. This translation allows to formalize the different configurable and non-configurable connectors of business processes to the corresponding SAT formulas. Thereafter, the minimalistic SAT solver Minisat is used to generate all models that represent all correct configurations. The present work proposes an extension of [Ait Wakrime et al. 2019].

We analyze and verify the configurable business process models and we optimize the cost of deploying these processes in a cloud environment. The proposed optimization yields the best cloud resource configuration that best fits tenants' requirements while minimizing the global cost. To achieve this purpose, we define a model that supports configurable business process (control-flow) and cloud resource allocation (resource-flow). Our model is based on a SAT-based formal approach, exactly on Weighted Partial MinSAT (WPMinSAT), that allows to generate all correct configurations of a configurable process model including the required cloud resources. These correct options help and assist the process designer to easily identify correct process variants. In addition, our approach allows to select the optimal cloud resource configuration. This optimization helps the process designer to select the configuration having the minimum price. Practically, we use WPMinSAT that is an optimized version of a SAT problem that has been proved to efficiently solve many combinatorial optimization problems. We also provide a set of translation rules that translate a configurable business process into SAT and WPMinSAT based models. These rules are implemented as a Java application that takes as input an XML document exported from Signavio Process Manager Tool. We applied our approach on a simplified example of supply chain business process model.

The rest of the paper is structured as follows. In Section 2, an example of configurable process model including Cloud resources is presented as well as some preliminaries about configurable business process and propositional satisfiability. Section 3 illustrates our formalization of process configuration elements. In Section 4, based on an algorithm, our approach is presented to find the correct and optimal process configuration. The approach validation is depicted in Section 5. We present the related work in Section 6. Finally, we conclude and provide insights for future work.

2 Motivating Example and Background

In this section, we present first a motivating example that presents and illustrates our approach. Second, we review the basic notions of propositional satisfiability. Finally, we introduce the Weighted Partial MinSAT problem.

2.1 Motivating Example: Configurable Business Process

In Figure 1, we present a simplified example of a configurable process model designed by a process provider. It is a supply chain process model illustrating different steps from the product purchasing to payment, processing, distributing products and the monitoring of their condition and quality. The process is modeled using the Configurable Business Process Model and Notation (C-BPMN) [Assy 2015, Hallerbach et al. 2010], a configurable extension to BPMN[1].

In a BP, the control-flow perspective describes activities and their execution ordering through different constructors, which permit its execution [Kiepuszewski et al. 2003]. In this work, we consider four main control-flow elements: activity (represented with a rectangle), edge (represented with arrows), event (represented with a circle) and connector (represented with a diamond). Three main connectors are used to model the splits (e.g. s_1) and the joins (e.g. j_1): OR ($\bigcirc$), exclusive OR ($\times$) and AND ($+$). The resource flow perspective describes the different resources required to execute activities in a BP. These resources are offered by cloud providers and they can include computers as virtual

[1] BPMN 2.0 specification: http://www.omg.org/spec/BPMN/2.0

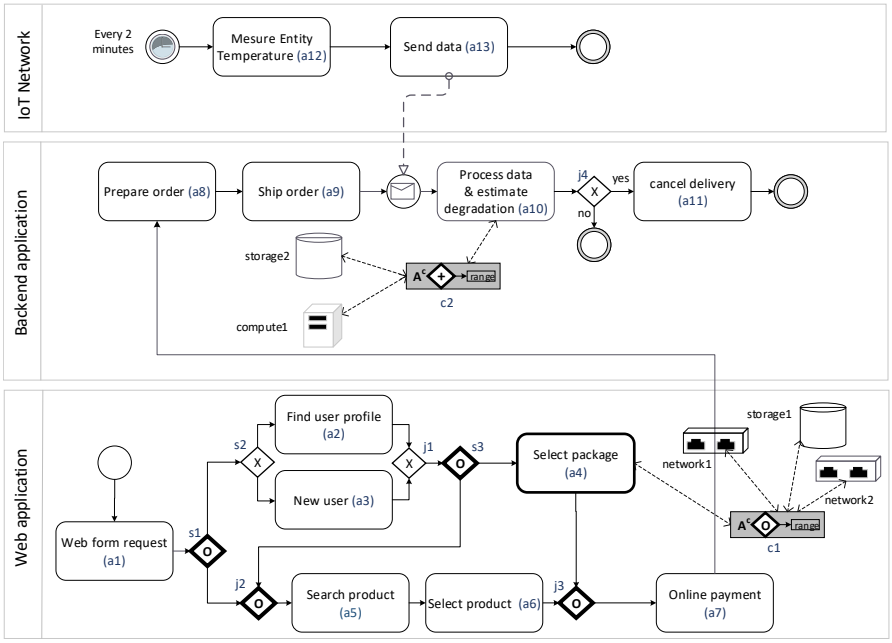

Figure 1: A supply chain configurable process model

machines, block storage, firewalls, load balancers and network devices. Indeed, this is guaranteed by Infrastructure as a Service in Cloud Computing taking into account the Quality of Service such availability, security, reliability, etc., as mentioned in the Service Level Agreement. In the present approach, the cost attributes of QoS is concerned in order to reduce it. In this paper, we consider three main cloud resources elements: storage (e.g. $storage_1$), network (e.g. $network_1$), and compute (e.g. $compute_1$). Configurable process models are proposed to represent in a generic manner similar process models. The control-flow variability can be captured by restricting the behavior of *configurable elements*: connector and activity. The non-configurable ones represent the commonalities in the configurable model. Since a configurable process can not be executed, all configurable elements should be configured and customized in order to obtain a variant that can be instantiated. An activity is configurable if it may be included (i.e. configured to ON) or excluded (i.e. configured to OFF) from the resulting variant. A connector may be configurable to restrict its behavior. It can be configured by (i) changing its type (e.g. from OR to AND), or/and (ii) restricting its incoming or outgoing branches. By configuration, a connector may change its type according to a set of configuration constraints [Rosemann and Van der Aalst 2007] (see Table 1).

Each row corresponds to the initial type that can be mapped and configured to one or more types in the columns. For example, a configurable connector having the OR type can be configured to any type while an AND type remains unchangeable. It is worth noting that the connector AND should never be configured to a sequence (i.e. only one input or output branch is maintained).

Going back to our example, a supply chain company has a number of branches selling

	OR	XOR	AND	SEQ
OR	✓	✓	✓	✓
XOR		✓		✓
AND			✓	

Table 1: Constraints for connectors configuration [Rosemann and Van der Aalst 2007]

different products in different countries. Depending on specific needs of a country, each branch performs a different variant of the configurable process model of Figure 1 in terms of structure and behavior. This example presents 7 configurable elements (6 connectors and one activity) which are highlighted with a thicker border. For instance, activities a_1 and a_6 are non-configurable, which means that they should be included in every configured variant. Whereas, the activity a_4 and the connector s_1 may vary from one process to another, as they are configurable. The resource-flow configuration is recently proposed in [Hachicha et al. 2016] allowing to explicitly model resource allocation variability in multi-tenant process models. This is ensured by linking an activity to allocated resources via a specific resource connector and association arcs. Hence, specific connectors $\mathbf{A}^c$ (called assignment operator) model the association between activities and the needed resources to be executed. It models a variable number of resources allocated to a specific activity. The resource configuration is then obtained by restricting the behavior of these connectors in the same way as control-flow ones. This configurable connector includes two parameters: (i) a configurable type following the same behavior as the control flow configurable connectors and (ii) a range specifies the number of the resources that are recommended to be allocated from each type. In this work, we propose to find the optimal number of resources for a given configuration in order to minimize the cost.

In this example, temperature data is sent from the IoT network to the Backend application via activity *a13*. Then, the activity *a10* processes the received data and estimates the quality degradation of the item based on pre-defined metrics (out of the scope of this work). This information is processed using a compute resource and stored in a storage one. The association between *a10* and the allocated resources is ensured by the connector c_2. This assignment operator has the type AND, this means that the activity *a10* requires both resources in order to be executed. Furthermore, the connector c_1 has the type OR, then the activity *a7* needs either $network_1$, $network_2$ or $storage_1$ or even the three of them. Finally, while the temperature is within the pre-defined range, the process ends with success. Otherwise, the activity *a11* cancels the delivery process.

2.2 Propositional Satisfiability

A CNF (Conjunctive Normal Form) formula Σ is a conjunction (interpreted as a set) of clauses, where a clause is a disjunction (interpreted as a set) of literals. A literal is a positive (x) or negative ($\neg x$) boolean variable. The two literals x and $\neg x$ are called complementary. A unit clause is a clause with only one literal (called unit literal). An empty clause, is interpreted as false, while an empty CNF formula, is interpreted as true. A set of literals is complete if it contains one literal for each variable occurring in Σ and fundamental if it does not contain complementary literals. An interpretation $\mathcal{I}$ of a Boolean formula Σ associates a value $\mathcal{I}(x)$ to some of the variables x appearing in Σ. An interpretation can be represented by a fundamental set of literals, in the obvious way.

A model of a formula Σ is an interpretation $\mathcal{I}$ that satisfies the formula, i.e., that satisfies all clauses of the formula. SAT is the problem of deciding whether a given CNF formula Σ admits a model or not. $\models_*$ denotes the logical consequence modulo unit propagation. Any propositional formula can be translated into an equi-satisfiable formula in CNF using Tseitin's linear encoding [Tseitin 1986].

Let us now briefly describe the basic components of CDCL (Conflict-Driven Clause Learning)-based SAT solvers [Moskewicz et al. 2001, Eén and Sörensson. 2003]. To be exhaustive, these solvers incorporate unit propagation (enhanced by efficient and lazy data structures), variable activity-based heuristic, literal polarity phase, clause learning, restarts and a learnt clauses database reduction policy. Typically, a SAT solver can be assimilated to a sequence of decision and unit propagation literals. Each literal chosen as a decision variable is affected to a new decision. If all literals are assigned, then $\mathcal{I}$ is a model of the formula and the formula is answered to be satisfiable. If a conflict is reached by unit propagation, a new clause is derived by conflict analysis [Zhang et al. 2001] considered as a logical consequence of the initial problem. If an empty clause is derived, then the formula is answered to be unsatisfiable.

2.3 Weighted Partial MinSAT problem

In propositional logic, as mentioned before, a variable x is a literal, as is its negation $\neg x$. A clause is a disjunction of literals. A weighted clause is a paire (c, w), where c represents a clause and w represents its weight (i.e. a natural number). A weighted CNF formula is a conjunction of weighted clauses. In addition, a truth assignment values to the propositional variables satisfies a literal x if x takes the value true (i.e. 1) and satisfies a literal $\neg x$ if x takes the value false (i.e. 0). WPMinSAT problem is an extension of a SAT problem which aims to satisfy a subset of weighted clauses. In a WPMinSAT problem, the clauses are classified into two categories: *hard clause* and *soft clause*. A clause is hard if in a truth assignment, the clause is evaluated to 1, otherwise it is said to be soft. The soft clauses have an associated weight as a finite number, whereas the hard clauses have an infinite weight. Let Φ be a WPMinSAT instance defined as a set of weighted clauses: $\Phi = \{(c_1, \infty), ..., (c_k, \infty), (c_{k+1}, w_{k+1}), ..., (c_m, w_m)\}$, where the first k clauses are hard and the other clauses are soft. To simplify the formula, infinite weights are omitted as follows: $\Phi = \{(c_1), ..., (c_k), (c_{k+1}, w_{k+1}), ..., (c_m, w_m)\}$. A truth assignment satisfies a clause c if it satisfies at least one literal of this clause, and satisfies a CNF formula if it satisfies all the clauses of that formula. A CNF formula is *satisfiable*, if there exists a truth assignment that satisfies it; otherwise, it is *unsatisfiable*. This could be applied to a given weighted clause (c, w): a truth assignment values to the propositional variables (1) satisfies this weighted clause if it satisfies c, then (2) it satisfies a weighted CNF formula $\{(c_1, w_1), ..., (c_m, w_m)\}$, if it satisfies all its clauses $c_1, ..., c_m$. A WPMinSAT problem for an instance of Φ consists in finding a truth assignment that satisfies all the hard clauses and minimizes the sum of weights of the satisfied soft clauses.

3 SAT-based Business Process Configuration

In this section, we introduce our SAT-based approach for business process configuration. We present the formalism that we use for the behavior modeling of all possible variants of a configurable process model while taking into account the needed cloud resources.

3.1 Approach Overview

Our approach consists of three main steps depicted by Figure 2. Firstly, a configurable business process including cloud resources is transformed into a WPMinSAT formalism using propositional logic. Secondly, we propose to find all correct process configurations. This consists in generating the set of all combinations of elements configurations (i.e. control-flow connectors, activities and resource-flow connectors) leading to correct process variants having the minimum cost. Typically, this means that the obtained process models by applying each obtained combination of elements configurations should satisfy the formulas obtained in the first step. These combinations are generated using a WPMinSAT solver. Thirdly, once the correct configurations are obtained, the process analysts will be able to correctly choose and configure their process variant without any additional checking.

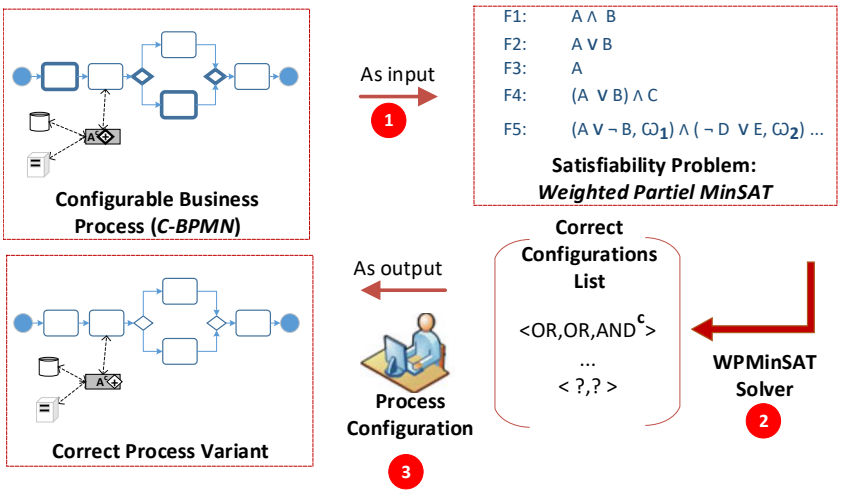

Figure 2: Our approach

In the following two sections, we present our formal approach for representing, first, the control flow elements of a Business Process (BP) as well as a Configurable Business Process (CBP) using SAT. Then, the cloud resources are integrated in our formalism using WPMinSAT.

3.2 Formalizing control-flow in a Configurable Process Model

In our SAT-based formal model, each element of a Business Process (BP) is transformed into a propositional formula using propositional variables and logical connectors like: $\neg, \vee, \wedge, \rightarrow$. The selection and generation of the BP variants during the configuration are related to a conditional statement or a conditional expression represented as a simple implication ($p \rightarrow q$) in classical logic. This implication is read as (if p then q). It merely means (if p is true, then q is also true). The statement ($p \rightarrow q$) is false only when p is true and q is false.

The control flow is ensured by edges that indicates the execution direction (to the right). Each node or element have one or more inputs and one or more outputs. In this work, we represent the input element (*InputElement*) for each process element **e** (i.e. here it may be a connector or an activity) as a propositional variable involving the element in question which is in turn represented by a propositional variable. Then, an implication is added between the propositional variable **e** representing this current element and the propositional variable representing the output element (*OutputElement*). Hence, the relation between each element and its input and output elements can be defined as follows:

$$(InputElement \rightarrow \mathbf{e}) \wedge (\mathbf{e} \rightarrow OutputElement) \tag{1}$$

where: **e**, $InputElement$ and $OutputElement \in$ {connector, activity}.

In order to obtain a structurally correct process model [Weske 2007], each activity should have only one input element (connector or another activity) and one output element (connector or another activity). However, for each connector, the Equation (1) is applied as many times as the number of input elements (in case of a join connector) or output elements. In mathematical logic, implication is one of the binary connectors of proposition calculus language. A formula in implicative form $a \rightarrow b$ equals $\neg a \vee b$ because they describe the same truth table. The Equation (1) can be easily translated into the following clause in order to obtain a CNF formula:

$$\psi \; = \; (\neg InputElement \vee \mathbf{e}) \wedge (\neg \mathbf{e} \vee OutputElement) \tag{2}$$

For instance, the Equation (1) can be applied on the activity a_2 that will give: $(s_2 \rightarrow a_2) \wedge (a_2 \rightarrow j_1)$. Thereafter, the formula ψ of Equation (2) $(\neg s_2 \vee a_2) \wedge (\neg a_2 \vee j_1)$.

In the following, we use the formula ψ in order to translate a configurable process to classical logic in a SAT, then to CNF. However, prior to that, we define in Table 2 the formalization of every configurable connector type. Table 2 depicts the mapping between each type of BPMN connector (linked to activities) and the corresponding SAT formulas. The first column contains the BPMN elements, the second one represents the SAT configurable connectors formalization and the third one depicts the SAT non-configurable connectors formalization. A connector is mapped into a disjunction between two implications: (i) the first one represents the relation between the connector and its input activities, and (ii) the second one represents the relation between the connector and its output activities. Each connector and each activity is formalised using a propositional variable.

In the Table 2, Cn refers to the configurable control-flow connector in the second column and to the regular connector (i.e. non-configurable) in the third column. The configurable connectors formulas define the customization behavior that consists of restricting either the incoming (in case of joins) or outgoing branches (in case of splits) for each type (i.e., either OR, XOR or AND based on the Table 1). Whereas for regular connectors, formulas define their runtime behavior. The *k* input elements are depicted by in_k and the *k* output elements by out_k. For instance, the formula: $((\bigvee_{in_k \in in_{Or_{j_x}}} in_k) \rightarrow Cn) \wedge (Cn \rightarrow out)$ means that: (i) during the execution of the regular OR-join connector Cn, either one or several input elements in_k are executed, and

their execution leads to the firing of the output element out, and (ii) during the configuration of the configurable OR-join Cn, one or more input elements are chosen to obtain an executable configured connector. Furthermore, in a CBP, an activity Act_C is configurable (e.g. a_4) if it can be included or excluded from a process variant. The Equation 1 could be directly applied if **e** = Act_C is included. Otherwise, if it is removed, the equation becomes: ($InputElement \rightarrow OutputElement$) where: $InputElement$ and $OutputElement$ $\in$ {connector, activity}. Therefore, a configurable activity is formalized as follows:

$$((InputElement \rightarrow Act_C) \wedge (Act_C \rightarrow OutputElement)) \vee (InputElement \rightarrow OutputElement) \quad (3)$$

where: $InputElement$ and $OutputElement \in$ {connector, activity}.

For instance, we obtain the following formula when we apply the Equation (3) on the configurable activity a_4: $((s_3 \rightarrow a_4) \wedge (a_4 \rightarrow j_3)) \vee (s_3 \rightarrow j_3)$.

3.3 Formalizing Resource-flow in a Configurable Process Model

In this paper, the resource-flow in a configurable process is insured by specific connectors defining the relation between an activity and the different cloud resources needed for its execution. Hence, in propositional logic, a first implication is added between an input activity and the connector $\mathbf{A}^c$. Then, an implication is added between the propositional variable $\mathbf{A}^c$ and the propositional variable representing the output element, which is in this case the allocated resource.

Accordingly, Equations 1 and 2 (i.e. CNF Formula) may be applied in the same way as for the control-flow where: (i) **e** is the specific configurable resource connector $\mathbf{A}^c$, (ii) the $InputElement$ is an activity and (iii) the $OutputElement$ is a resource. The connector $\mathbf{A}^c$ can be either a configurable OR^c, a configurable XOR^c or a configurable AND^c. Like the control-flow connectors, a configurable resource connector can change its type according to Table 1. However, only outgoing branches may be restricted in this case. In fact, these specific connectors are always considered to be split connectors. As one of our primary goals in this paper is to minimize the cost of the consumed cloud resources, we add a new parameter in our formalization: the cost of an allocated resource. In order to represent this parameter, we need to add a weight w to each resource. Hence, we adapt the formula ψ in accordance with WPMinSAT instance. More precisely, we add the weight at the second clause of ψ formula dealing with resource implication. We obtain the Equation 4 which is reformulated into Equation 5.

$$\varphi \; = \; (\neg InputElement \vee \mathbf{A}^c) \wedge (\neg \mathbf{A}^c \vee OutputElement, w) \quad (4)$$

where: $InputElement \in$ {activity} and $OutputElement \in$ {resource}.

Returning to our example, the Equation (4) is applied on the connector c_1 with a range 2 to obtain the following formula: $(\neg a_4 \vee c_1) \wedge (\neg c_1 \vee (network_1, 2) \vee (storage_1, 2) \vee (network_2, 2))$.

$$\varphi \; = \; \varphi_h \wedge \varphi_s \quad (5)$$

Table 3 illustrates the formalisation of configurable and non-configurable resource connectors using WPMinSAT formulas. In the second column, Cn indicated the configurable resource-flow connector, and in the third column, it refers to the regular ones (i.e. non-configurable).

The input element is depicted by Act_{in} (representing the activity) and the output element by Rsr_{out_k} (representing an allocated resource). For example, the configurable XOR^c-split resource connector is defined as follows:

$$(Act_{in} \rightarrow Cn) \wedge (Cn \rightarrow (\bigvee_{Rsr_{out_k} \in Rsr_{out\,Xor_{s_x}}} Rsr_{out_k}), w)$$

4 WPMinSAT-based Method for Correct and Optimal Process Configuration

This section describes our WPMinSAT-based method that is centered around BP configuration including control-flow perspective and resource-flow perspective.

As explained in the previous section, the control-flow is translated into a CNF formula and the resource-flow is formalized as a weighted CNF formula (i.e. a set of weighted clauses). A weighted clause is represented by a pair (c, w), where c is a classical clause and w is a weight that is a natural number representing the cost associated to a resource.

The classical CNF formulas are the hard clauses and the weighted CNF formulas are the soft clauses. Then, the conjunction of the hard clauses and the soft clauses is a weighted partial CNF formula. In our approach, the formula ψ consists in hard clauses and the formula φ consists in the combination of the hard clauses φ_h and the soft clauses φ_s (ref. Equations 4 and 5).

BPMN elements	***SAT configurable connectors*** ($R_{C\_SAT}(C_n)$)	***SAT non-configurable connectors*** ($R_{NC\_SAT}(C_n)$)
	$((\bigvee_{in_k \in in_{Or_{j_x}}} in_k) \rightarrow \mathbf{Cn}) \wedge (\mathbf{Cn} \rightarrow out)$	$((\bigvee_{in_k \in in_{Or_{j_x}}} in_k) \rightarrow \mathbf{Cn}) \wedge (\mathbf{Cn} \rightarrow out)$
	$(in \rightarrow \mathbf{Cn}) \wedge (\mathbf{Cn} \rightarrow (\bigvee_{out_k \in out_{Or_{s_x}}} out_k))$	$(in \rightarrow \mathbf{Cn}) \wedge (\mathbf{Cn} \rightarrow (\bigvee_{out_k \in out_{Or_{s_x}}} out_k))$
	$((\bigvee_{k \in \{1\ldots \mid in_{And_{j_x}} \mid\}} \bigwedge_{i \in \{1\ldots k\}} in_i) \rightarrow \mathbf{Cn}) \wedge (\mathbf{Cn} \rightarrow out)$	$((\bigwedge_{in_k \in in_{And_{j_x}}} in_k) \rightarrow \mathbf{Cn}) \wedge (\mathbf{Cn} \rightarrow out)$
	$(in \rightarrow \mathbf{Cn}) \wedge (\mathbf{Cn} \rightarrow (\bigvee_{j \in \{1\ldots \mid out_{And_{s_x}} \mid\}} \bigwedge_{i \in \{1\ldots j\}} out_i))$	$(in \rightarrow \mathbf{Cn}) \wedge (\mathbf{Cn} \rightarrow (\bigwedge_{out_k \in out_{And_{s_x}}} out_k))$
	$((\bigvee_{in_k \in in_{Xor_{j_x}}} in_k) \rightarrow \mathbf{Cn}) \wedge (\mathbf{Cn} \rightarrow out)$	$((\bigvee_{in_r \in in_{Xor_{j_x}} \wedge in_k \in in_{Xor_{j_x}} \setminus \{in_r\}} in_r \wedge \neg in_k) \rightarrow \mathbf{Cn}) \wedge (\mathbf{Cn} \rightarrow out)$
	$(in \rightarrow \mathbf{Cn}) \wedge (\mathbf{Cn} \rightarrow (\bigvee_{out_k \in out_{Xor_{s_x}}} out_k))$	$(in \rightarrow \mathbf{Cn}) \wedge (\mathbf{Cn} \rightarrow (\bigvee_{out_r \in out_{Xor_{j_x}} \wedge out_k \in out_{Xor_{j_x}} \setminus \{out_r\}} out_r \wedge \neg out_k))$

Table 2: Different (non-)configurable control flow connectors of BP and the corresponding SAT formulas

BPMN elements	***SAT configurable connectors*** $(R_{CR\_SAT}(C_n, w))$	***SAT non-configurable connectors*** $(R_{NCR\_SAT}(C_n, w))$
OR$_s$	$(Act_{in} \rightarrow Cn) \wedge (Cn \rightarrow (\bigvee_{Rsr_{out_k} \in Rsr_{outOr_{s_x}}} Rsr_{out_k}), w)$	$(Act_{in} \rightarrow Cn) \wedge (Cn \rightarrow (\bigvee_{Rsr_{out_k} \in Rsr_{outOr_{s_x}}} Rsr_{out_k}), w)$
AND$_s$	$(Act_{in} \rightarrow Cn) \wedge (Cn \rightarrow (\bigvee_{j \in \{1\ldots \mid Rsr_{outAnd_{s_x}} \mid\}} \bigwedge_{i \in \{1\ldots j\}} Rsr_{out_i}), w)$	$(Act_{in} \rightarrow Cn) \wedge (Cn \rightarrow (\bigwedge_{Rsr_{out_k} \in Rsr_{outAnd_{s_x}}} Rsr_{out_k}), w)$
XOR$_s$	$(Act_{in} \rightarrow Cn) \wedge (Cn \rightarrow (\bigvee_{Rsr_{out_k} \in Rsr_{outXor_{s_x}}} Rsr_{out_k}), w)$	$(Act_{in} \rightarrow Cn) \wedge (Cn \rightarrow (\bigvee_{Rsr_{out_r} \in Rsr_{outXor_{j_x}} \wedge Rsr_{out_k} \in Rsr_{outXor_{j_x}} \setminus \{Rsr_{out_r}\}} Rsr_{out_r} \wedge \neg Rsr_{out_k}), w)$

Table 3: Different (non-)configurable resource allocation connectors of BP and the corresponding SAT formulas

In order to obtain a correct configuration satisfying the cost constraint, we start by defining a variable $Activity_{root}$ that represents the initial activity and we conjunctively add the formulas ψ and φ as follows: $\psi \wedge \varphi \wedge Activity_{root}$. We also define the variable $Activity_{target}$ that represents the final activity in the process. Then, a correct process variant is a configuration that considers the initial activity as a starting point and by applying unit propagation, that is a rule of correct inference, reaches the final activity. In this work, we artificially add initial and final activities to respectively represent the unique initial point of the process model and the unique final one.

Definition 1[Correct configuration] : Let *cbp* a C-BPMN and let Φ be a SAT formula that represents a transformation of *cbp* to a CNF formula. A configuration $conf$ of *cbp* is a process where each element **e** $\in conf$ allows the following formula: $\psi \wedge \varphi \wedge Activity_{root} \models_* Activity_{target}$ which means that:

$$\Phi = \psi \wedge \varphi \wedge Activity_{root} \wedge \neg Activity_{target}$$
$$\Phi = \psi \wedge \varphi_h \wedge \varphi_s \wedge Activity_{root} \wedge \neg Activity_{target}$$

where: $\psi \wedge \varphi_h \wedge Activity_{root} \wedge \neg Activity_{target}$ represents the hard clauses and φ_s represents the soft clauses.

The Definition 1 explains how to extract the correct configuration from configurable BP with configurable resource allocation. Hence, the Definition 1 implies that a configurable process including control-flow and resource-flow have one correct configuration *iff* the formula Φ admits one assignment to the variables such that all hard clauses are satisfied and the total weight of satisfied soft clauses is minimized. This means that a correct process is possible *iff* the Φ formula has an assignment deduced by *unit propagation* (also called *Boolean constraint propagation*). Hence, a correct configuration can be extracted using unit propagation. This later is an automated theorem proving procedure used to simplify a set of clauses. Also, it is one of the key processes and the most used one in SAT resolution algorithms. Its working principle is the following: until that formula contains a unitary clause, assign true to its literals. In the weighted partial MinSAT case, a unit propagation is started, although restricts it to hard clauses as soft clauses need not be definitely satisfied.

Table 2 shows the translation of (non-)configurable control-flow connectors of BP to SAT formulas. Similarly, the resource-flow connectors are formalized using WPMinSAT formulas in Table 3. The formulas in both tables are basically derived from Equations 1 and 4. Based on this formalization of all configurable process model elements, we define Algorithm 1. Hence, this algorithm allows to formalise a configurable process model as follows. First, the formula Φ, ψ and φ are declared and initialised (lines 1). Second, a *for each* loop iterates over all the elements **e** of a configurable business process CBP. Thereafter, the algorithm applies the corresponding formalization (defined in Tables 2 and 3) : $R_{C\_SAT}(e)$, $R_{NC\_SAT}(e)$, $R_{CA\_SAT}(e)$, $R_{NC\_SAT}(e)$, $R_{A\_SAT}(e)$, $R_{CR\_SAT}(e, w)$ and $R_{NCR\_SAT}(e, w)$ (lines 2 to 16). For instance, if the element **e** is a configurable control-flow connector, the relation $R_{C\_SAT}(e)$ (i.e. the second column of Table 2) is applied to obtain the corresponding formula (lines 3 and 4). Similarly, $R_{NC\_SAT}(e)$ is applied for non-configurable control-flow connectors (lines 5 and 6). The configurable activities are formalized using $R_{CA\_SAT}(e)$ relation that is based on Equation 3 (lines 7 and 8). Regarding non-configurable activity, its formalization is carried out by applying the relation $R_{A\_SAT}(e)$ based on Equation 1 having **e** as the non-configurable activity in question (lines 9 and 10). Until now, all the formalization, concerning the control-flow connectors and activities, is assigned to the formula ψ.

On the other hand, a configurable resource-flow connector **e** is formalized using the relation $R_{CR\_SAT}(e_i, w)$ with the weight w (ref. the second column of Table 3).

This weight of the formula φ represents the cost of each resource of current resource-flow connector **e**. This formula $R_{CR\_SAT}(e_i, w)$ is repeated using a *for* loop as many times as the value of $range$ (lines 11, 12 and 13). In the same way, $R_{NCR\_SAT}(e_i, w)$ (ref. the third column of Table 3) is used to formalize all non-configurable resource-flow connectors (lines 14, 15 and 16). The formalisation of resource-flow connectors is affected to the formula φ.

Finally, the formula Φ is defined by the conjunction between ψ, φ, $Activity_{root}$ and $\neg Activity_{target}$, then it is returned as the output of the algorithm (lines 17 and 18). Let us consider the set of activities and connectors as depicted in configurable business process in Figure 1. In this example, we omit the IoT network part, we are only interested in the Web application and Backend application parts to demonstrate the feasibility of our approach. The Algorithm 1 is applied to this configurable business process including control-flow and resource-flow. The control-flow consists of non-configurable connectors like s_2, j_1, s_4, j_4, configurable connectors like s_1, s_3, j_2, j_3, non-configurable activities: $a_1, a_2, a_3, a_4, a_5, a_7, a_8, a_9, a_{10}, a_{11}$ and configurable activity like a_4. In addition, resource-flow contains the connectors c_1 with the resources $network_1, storage_1$ and $network_2$ related to activity a_4, given that each resource here has a cost is equal 1. The range of this connector is equal 2. Also, it contains the connectors c_2 that connects the resources $compute_1$ and $storage_2$ to activity a_{10} with a range which is worth 1.

The following formulas ψ and φ represent the formalization of the different (non-)configurable control-flow connectors and (non-)configurable resource allocation connectors. ψ is obtained by applying the Algorithm 1, in particular lines 3, 5, 7 and/or 9, while Φ is obtained by applying the lines 11 and/or 14. In sequel, Φ is the disjunction between ψ, φ, initial activity and target activity and is obtained by applying the line 17 of Algorithm 1.

$$\psi = (a_1 \rightarrow s_1) \wedge s_1 \rightarrow (s_2 \vee j_2) \wedge (s_1 \rightarrow s_2) \wedge (s_2 \rightarrow (a_2 \wedge \neg a_3) \vee s_2 \rightarrow (a_3 \wedge \neg a_2)) \wedge ((s_1 \vee s_3) \rightarrow j_2) \wedge (j_2 \rightarrow a_5) \wedge ((a_2 \wedge \neg a_3) \rightarrow j_1 \vee (a_3 \wedge \neg a_2) \rightarrow j_1) \wedge (j_1 \rightarrow s_3) \wedge (j_1 \rightarrow s_3) \wedge (s_3 \rightarrow (a_4 \vee j_2)) \wedge ((a_4 \vee a_6) \rightarrow j_3) \wedge (j_3 \rightarrow a_7) \wedge (a_{10} \rightarrow j_4) \wedge ((j_4 \rightarrow a_{11}) \vee (j_4 \rightarrow \neg a_{11})) \wedge (s_3 \rightarrow a_4) \wedge (a_4 \rightarrow j_3) \vee (s_3 \rightarrow j_3) \wedge (a_1 \rightarrow s_1) \wedge (s_2 \rightarrow a_2) \wedge (a_2 \rightarrow j_1) \wedge (s_2 \rightarrow a_3) \wedge (a_3 \rightarrow j_1) \wedge (j_2 \rightarrow a_5) \wedge (a_5 \rightarrow a_6) \wedge (j_3 \rightarrow a_7) \wedge (a_7 \rightarrow a_8) \wedge (a_7 \rightarrow a_8) \wedge (a_8 \rightarrow a_9) \wedge (a_8 \rightarrow a_9) \wedge (a_9 \rightarrow a_{10}) \wedge (a_9 \rightarrow a_{10}) \wedge (a_{10} \rightarrow j_4) \wedge (j_4 \rightarrow a_{11})$$

$$\varphi = (a_4 \rightarrow c_1) \wedge (c_1 \rightarrow (network_{1_1}, 2 \vee storage_{1_1}, 2 \vee network_{2_1}, 2)) \wedge (a_4 \rightarrow c_1) \wedge (c_1 \rightarrow (network_{1_2}, 2 \vee storage_{1_2}, 2 \vee network_{2_2}, 2)) \wedge (a_{10} \rightarrow c_2) \wedge (c_2 \rightarrow (compute_1, 1 \wedge storage_2, 1)$$

Therefore, the formula $\Phi = \psi \wedge \varphi \wedge a_1 \wedge \neg a_{11}$.

5 Evaluation: SAT Problems Induced by Business Process

In this section, we evaluate the quality and the efficiency of our work. The proposed approach was tested and developed using as input the real configurable business process of Figure 1. Using our formal model, the SAT formula Φ is defined and then represented using DIMACS format that is a standard interface to SAT solvers. The DIMACS format is obtained after the execution of the Algorithm 1 described above on our configurable process example. This format is used to define a Boolean expression, written in CNF formulas which is stored using a file having for extension *.dimacs*. This file is used as input of the used solver. Each line in this file is a list of variables separated by spaces and ended with 0. This list represents a clause which is a disjunction of literals and a

Algorithm 1: Configurable Business Processes to SAT

Input: Configurable Business Process CBP
Output: Formula Φ represents a WPMinSAT formalization

Φ, ψ and φ are a CNF formulas;
/* Iterates over all elements of CBP */
for each $e \in CBP$ **do**
  if $type(e) = \{$configurable control-flow connector$\}$ **then**
    $\psi \leftarrow R_{C\_SAT}(e)$;
  if $type(e) = \{$non-configurable control-flow connector$\}$ **then**
    $\psi \leftarrow R_{NC\_SAT}(e)$;
  if $type(e) = \{$configurable activity$\}$ **then**
    $\psi \leftarrow R_{CA\_SAT}(e)$;
  if $type(e) = \{$non-configurable activity$\}$ **then**
    $\psi \leftarrow R_{A\_SAT}(e)$;
  if $type(e) = \{$configurable resource-flow connector$\}$ **then**
    for all i $\in$ range **do**
      $\varphi \leftarrow R_{CR\_SAT}(e_i, w)$;
  if $type(e) = \{$non-configurable resource-flow connector$\}$ **then**
    for all i $\in$ range **do**
      $\varphi \leftarrow R_{NCR\_SAT}(e_i, w)$;
$\Phi \leftarrow \psi \wedge \varphi \wedge Activity_{root} \wedge \neg Activity_{target}$;
return Φ;

literal is either a positive variable x or its negation $\neg x$. In the file *.dimacs*, a variable is represented by an integer between 1 and n and its negation $\neg$ is represented by the sign $-$. The clauses are distinct and may not simultaneously contain opposite literals. Hence, the lines of a file *.dimacs* represent the conjunction of the clauses of the problem. Each line of a file *.dimacs* represents a CNF format knowing that the first integer in the clause is its weight. The weights must be greater than or equal to 1. The first line of a file *.dimacs* is written as following: $p \quad wcnf \quad n \quad c \quad top$. As the hard clauses have ∞ weight, the weight top is ∞. Since the weight of each hard clause must be equal or greater than top and the weight of each soft clause must be smaller than top, we define it as the maximum weight of the used resources in a CBP model multiplied by 2. In addition, n indicates the number of variables that is the number of activities and connectors. Moreover, c is the exact number of clauses contained in the file.

In the sequel, we used the MinSatz[2] solver [Li et al. 2012] that takes the file *.dimacs* as an argument. This solver is based on a branch and bound algorithm for the minimum satisfiability problem. This problem decides if a weighted partial MinSAT formula is evaluated to *true*. In this case, the formula is satisfiable (SAT) and the optimum solution is found. Otherwise, the formula is unsatisfiable (UNSAT). Due to space concerns, an

[2] https://home.mis.u-picardie.fr/ cli/minsatz2013.c

extract of the *.dimacs* file generated from our motivating example is represented in the Listing 1.

```
p    wcnf    25    48    4
4    -1      2     0
4    -2      3     -2    4    0
4    -2      3     0
4    -3      5     -3    6    0

....
```

Listing 1: An extract of .dimacs file with 25 variables and 48 clauses.

In the Listing 1, each line represents a clause that is a sequence of distinct non-null numbers between $-n$ and n, that ends with 0 on the same line. The opposite literals x and $-x$ do not belong to the same line. In addition, positive number denotes the corresponding variable and the negative one denotes the negation of the corresponding variable. The MinSatz solver checks the existence of the correct configurations and generates them if they exist. Else, the MinSatz solver returns UNSAT i..e, the configurable business process does not contain any correct configuration.

✓: including activity a_4							✗: without activity a_4						
s_1	s_3	j_2	j_3	c_1	c_2	a_4	s_1	s_3	j_2	j_3	c_2	a_4
OR	OR	OR	OR	XOR-AND-OR	AND	!	OR	OR	OR	OR	AND	%
XOR	OR	OR	OR	XOR-AND-OR	AND	✓	XOR	OR	OR	OR	AND	✗
AND	OR	OR	OR	XOR-AND-OR	AND	✓	AND	OR	OR	OR	AND	✗
OR	XOR	OR	OR	XOR-AND-OR	AND	✓	OR	XOR	OR	OR	AND	✗
XOR	XOR	OR	OR	XOR-AND-OR	AND	✓	XOR	XOR	OR	OR	AND	✗
AND	XOR	OR	OR	XOR-AND-OR	AND	✓	AND	XOR	OR	OR	AND	✗
OR	AND	OR	OR	XOR-AND-OR	AND	✓	OR	AND	OR	OR	AND	✗
XOR	AND	OR	OR	XOR-AND-OR	AND	✓	XOR	AND	OR	OR	AND	✗
AND	AND	OR	OR	XOR-AND-OR	AND	✓	AND	AND	OR	OR	AND	✗
OR	OR	XOR	OR	XOR-AND-OR	AND	✓	OR	OR	XOR	OR	AND	✗
XOR	OR	XOR	OR	XOR-AND-OR	AND	✓	**XOR**	**OR**	**XOR**	**OR**	**AND**	✗
OR	XOR	XOR	OR	XOR-AND-OR	AND	✓	OR	XOR	XOR	OR	AND	✗
XOR	**XOR**	**XOR**	**OR**	**XOR**-AND-OR	**AND**	✓	XOR	XOR	XOR	OR	AND	✗
OR	AND	AND	OR	XOR-AND-OR	AND	✓	OR	AND	AND	OR	AND	✗
AND	AND	AND	OR	XOR-AND-OR	AND	✓	AND	AND	AND	OR	AND	✗
OR	OR	OR	XOR	XOR-AND-OR	AND	✓	OR	OR	OR	XOR	AND	✗
XOR	OR	OR	XOR	XOR-AND-OR	AND	✓	XOR	OR	OR	XOR	AND	✗
OR	XOR	OR	XOR	XOR-AND-OR	AND	✓	OR	XOR	OR	XOR	AND	✗
XOR	XOR	OR	XOR	XOR-AND-OR	AND	✓	XOR	XOR	OR	XOR	AND	✗
OR	OR	XOR	XOR	XOR-AND-OR	AND	✓	OR	OR	XOR	XOR	AND	✗
XOR	OR	XOR	XOR	XOR-AND-OR	AND	✓	XOR	OR	XOR	XOR	AND	✗
OR	XOR	XOR	XOR	XOR-AND-OR	AND	✓	OR	XOR	XOR	XOR	AND	✗
XOR	XOR	XOR	XOR	XOR-AND-OR	AND	✓	XOR	XOR	XOR	XOR	AND	✗
AND	AND	OR	AND	XOR-AND-OR	AND	✓	AND	AND	OR	AND	AND	✗
AND	AND	AND	AND	XOR-AND-OR	AND	✓	AND	AND	AND	AND	AND	✗

Table 4: All correct configurations

The different correct configurations obtained using our solver are represented in the Table 4. We can distinguish two groups of configurations depending on the activity a_4

configuration. On the left-hand side of this table, all correct configurations including the activity a_4 are presented. The correct configurations without the activity a_4 are depicted by the right-hand side of this table. This later group does not include the connector c_1 as well since it is linked to a_4. We use MinSatz solver as a WPMinSAT solver in order to reason about a configurable process model including resource allocation configuration. WPMinSAT solver is a system used to decide satisfiability. The MinSatz solver was given 3 seconds to complete the satisfiability checks on our motivating example. Then, we propose to find for each correct configuration the optimal cloud resources allocation. For this aim, we select the optimal number of resources for a given configuration in order to minimize the global process cost. The configuration highlighted is the optimal one in case of a configuration containing activity a_4 with the cost 6. On the other hand, the optimal configuration without a_4 is highlighted and has the minimal cost that is equal 2.

Tool support

In order to allow the usability of our proposal, we implemented a Java application [3] that automatically transforms a configurable business process into DIMACS. The input to our tool is a BPMN file (i.e., an XML file exported from Signavio Process Manager Tool) that represents our configurable process model. The output of this tool is the DIMACS file that is used then as an input of the Sat Solver. In practice, the analyst needs to model the configurable process model using the Signavio Tool. He/she specifies the configurable connectors as well as the required Cloud resources for each activity. Then, once the BPMN 2.0 file is generated, our implemented tool is used to generate the DIMACS file. Afterwards, the SAT Solver is executed to obtain the optimal and correct process configurations. Finally, the analyst picks the appropriate process configuration that is suitable to his/her needs.

6 Related Work

Several approaches have been proposed to model variability and to provide a correctness verification of the configurable process models [Gottschalk et al. 2008, Rosemann and Van der Aalst 2007, van der Aalst et al. 2012, Schnieders and Puhlmann 2006, Hallerbach et al. 2010, Kumar and Yao 2012, La Rosa et al. 2009, Asadi et al. 2014, Assy and Gaaloul 2016]. In this context and in [Gottschalk et al. 2008], Gottschalk et al. propose an approach for extending YAWL language, as a common workflow modelling language with opportunities for predefining alternative model versions within a single workflow model. They propose to allow the configuration of workflow models to a relevant variant in a controlled way. Configurable Event-Driven Process Chains (C-EPCs) [Rosemann and Van der Aalst 2007] is an extended reference modelling language which allows capturing the core configuration patterns. The authors define the formalization of C-EPCs as well as examples for typical configurations. In addition, they propose the identification of a comprehensive list of configuration patterns and they test the quality of these extensions in some experiments. Van der Aalst et al. [van der Aalst et al. 2012] propose an approach inspired by the "operating guidelines" used for partner synthesis for verifying that configurations do not lead to behavioral issues like deadlocks and livelocks. They represent the configuration process as an external service, and compute a characterization of all such services which meet particular requirements via the notion

[3] https://github.com/aaitwakrime/CPMtoWPMinSAT

of configuration guideline. [Schnieders and Puhlmann 2006] proposes an approach for process family architecture modeling and implementation. The authors propose a set of variability mechanisms for BPMN and outlined their implementation using HyperSenses program generators. In the approach [Hallerbach et al. 2010], the modeling of a reference process model which represents a base process model is discussed. The necessary adjustments of this process are treated to configure this base process model to different process variants. This is done by introducing the Provop framework.

Other authors focused also on the issue of process models configuration. For example, in [Assy and Gaaloul 2016] the different variants of configurable process models are derived based on domain constraints and business rules. The work presented in [Kumar and Yao 2012] shows how process templates can be combined with business rules to design flexible business processes. This idea is applied to separate the basic process flow from the business policy elements. Another approach to capture variability in process models is represented in [La Rosa et al. 2009]. This approach proposes a formal framework for representing system variability that allows to detect circular dependencies and contradictory constraints in questionnaire models. In [Asadi et al. 2014] an approach including formal representations and algorithms based on logical reasoning is proposed. Moreover, in this work, the validation in the context of customization of process variants is discussed.

Resource allocation in business process management and configuration has been regarded in a number of approaches. In [Kumar and Yao 2012], an approach is proposed to model configurable business processes integrating control flow, resource needs and data by applying business rules to a generic process template. The authors of [Havur et al. 2016] define an approach to achieve an optimal scheduling of work items that have dependencies and resource conflicts in Business Process Management Systems. To the best of our knowledge, so far, surprisingly little effort has been put into the configurable business process models including resource-flow using a formal model with a focus on cost-efficient resource allocation. In [Hachicha et al. 2016], the resource-flow connectors are proposed to select the needed resources by each activity. These connectors are used in our paper, however, this work does not propose a formal model for a resources configuration taking into account an important constraint namely cost.

On the other hand, a number of works emphasize the value of SAT inside, for instance, product line engineering, business process, Cloud Computing, etc. For instance, in [Mendonca 2009, He et al. 2018] propositional logic and SAT are used to analyze feature models which is a popular variability modeling notation used in product line engineering. [Bo et al. 2017] proposes the use of improved separation of duty algebra to describe a satisfiability problem of qualification requirements and quantification requirements. This is being done to provide a separation of duty and binding of duty requirements. And also in the other works [Ait Wakrime 2017, Ait Wakrime et al. 2015, Ait Wakrime and Jabbour 2016], SAT-based approach is used to relax the failed queries through rewriting them in the Cloud Computing exactly in the Software as a Service (SaaS). In addition, SAT is adopted to compute a minimum composition within preserved-privacy of SaaS Services and Data as a Service (DaaS) Services for a given customer's request.

In summary, our approach proposes an extension of [Ait Wakrime et al. 2019] allowing to verify the configurable business process models but also to optimize the cost of their deployment in a Cloud environment using the SAT. The major differences from the above cited approaches are the following points: (1) It generates structurally correct process configuration options that do not contain run-time errors. (2) It generates all correct options from the beginning (at design time) which allows to assist process designer during the configuration time. (3) Since SAT has gained considerable audience

with the advent of a new generation of SAT solvers during these last few years, the application of SAT techniques to configurable business process offers many benefits in terms of their analyses concerning the generation of correct configuration including optimal resource allocation.

7 Conclusion and Further Work

In this work, we propose an approach for ensuring correct process configuration while taking into consideration cloud resource configuration. We use a formal model based on SAT and WPMinSAT in order to find these configurations and to select the optimal resources number for a minimal cost. Hence, we help process analysts in configuring correct processes while optimizing their deployment cost. We showed the applicability of our approach and we validated it using a WPMinSAT solver. As future work, we aim to consider the different pricing strategies proposed by the cloud providers. For example, AWS proposes the following pricing strategies: on-demand, reserved, spot, and savings plans. We also plan to apply our proposed approach on large scale configurable business processes.

References

[Ait Wakrime et al. 2015] Abderrahim Ait Wakrime, Salima Benbernou, and Said Jabbour. Relaxation based SaaS for Repairing Failed Queries over the Cloud Computing. In e-Business Engineering (ICEBE), 2015 IEEE 12th International Conference on. IEEE, 2015.

[Ait Wakrime and Jabbour 2016] Abderrahim Ait Wakrime and Said Jabbour. On repairing queries in cloud computing. In 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA), pages 1–6. IEEE, 2016.

[Ait Wakrime 2017] Abderrahim Ait Wakrime. Satisfiability-based privacy-aware cloud computing. The Computer Journal, 60(12):1760–1769, 2017.

[Ait Wakrime et al. 2019] Abderrahim Ait Wakrime, Souha Boubaker, Slim Kallel, and Walid Gaaloul. A sat-based formal approach for verifying business process configuration. In International Conference on Big Data Innovations and Applications, pages 47–62. Springer, 2019.

[Asadi et al. 2014] Mohsen Asadi, Bardia Mohabbati, Gerd Gröner, and Dragan Gasevic. Development and validation of customized process models. Journal of Systems and Software, 96:73–92, 2014.

[Assy 2015] N. Assy. Automated support of the variability in configurable process models. PhD thesis, University of Paris-Saclay, France, 2015.

[Assy and Gaaloul 2016] Nour Assy and Walid Gaaloul. Extracting configuration guidance models from business process repositories. In International Conference on Business Process Management, pages 198–206. Springer, 2016.

[Bo et al. 2017] Yang Bo, Chunhe Xia, Zhigang Zhang, and Xinzheng Lu. On the satisfiability of authorization requirements in business process. Frontiers of Computer Science, 11(3):528–540, 2017.

[Eén and Sörensson. 2003] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In International conference on theory and applications of satisfiability testing, pages 502–518. Springer, 2003.

[Gottschalk et al. 2008] Florian Gottschalk, Wil MP Van Der Aalst, Monique H Jansen-Vullers, and Marcello La Rosa. Configurable workflow models. International Journal of Cooperative Information Systems, 17(02):177–221, 2008.

[GröNer et al. 2013] Gerd GröNer, Marko BošKović, Fernando Silva Parreiras, and Dragan GašEvić. Modeling and validation of business process families. Information Systems, 38(5):709–726, 2013.

[Hachicha et al. 2016] Emna Hachicha, Nour Assy, Walid Gaaloul, and Jan Mendling. A configurable resource allocation for multi-tenant process development in the cloud. In International Conference on Advanced Information Systems Engineering, pages 558–574. Springer, 2016.

[Hallerbach et al. 2010] Alena Hallerbach, Thomas Bauer, and Manfred Reichert. Capturing variability in business process models: the provop approach. Journal of Software Maintenance and Evolution: Research and Practice, 22(6-7):519–546, 2010.

[Hallerbach et al. 2009] Alena Hallerbach, Thomas Bauer, and Manfred Reichert. Guaranteeing soundness of configurable process variants in provop. In 2009 IEEE Conference on Commerce and Enterprise Computing, pages 98–105. IEEE, 2009.

[Havur et al. 2016] Giray Havur, Cristina Cabanillas, Jan Mendling, and Axel Polleres. Resource allocation with dependencies in business process management systems. In International Conference on Business Process Management, pages 3–19. Springer, 2016.

[He et al. 2018] Fei He, Yuan Gao, and Liangze Yin. Efficient software product-line model checking using induction and a sat solver. Frontiers of Computer Science, 12(2):264–279, 2018.

[Kiepuszewski et al. 2003] Bartek Kiepuszewski, Arthur HM ter Hofstede, and Wil MP van der Aalst. Fundamentals of control flow in workflows. Acta Informatica, 39(3):143–209, 2003.

[Kumar and Yao 2012] Akhil Kumar and Wen Yao. Design and management of flexible process variants using templates and rules. Computers in Industry, 63(2):112–130, 2012.

[La Rosa et al. 2009] Marcello La Rosa, Wil MP van der Aalst, Marlon Dumas, and Arthur HM Ter Hofstede. Questionnaire-based variability modeling for system configuration. Software & Systems Modeling, 8(2):251–274, 2009.

[Li et al. 2012] Chu Min Li, Zhu Zhu, Felip Manyà, and Laurent Simon. Optimizing with minimum satisfiability. Artificial Intelligence, 190:32–44, 2012.

[Marques-Silva and Sakallah 2000] João P Marques-Silva and Karem A Sakallah. Boolean satisfiability in electronic design automation. In Proceedings of the 37th Annual Design Automation Conference, pages 675–680. ACM, 2000.

[Mendonca 2009] Marcilio Mendonca, Andrzej Wkasowski, and Krzysztof Czarnecki. Sat-based analysis of feature models is easy. In Proceedings of the 13th International Software Product Line Conference, pages 231–240. Carnegie Mellon University, 2009.

[Moskewicz et al. 2001] Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In Proceedings of the 38th annual Design Automation Conference, pages 530–535. ACM, 2001.

[Recker et al. 2005] Jan Recker, Michael Rosemann, Wil Van Der Aalst, and Jan Mendling. On the syntax of reference model configuration–transforming the c-epc into lawful epc models. In International Conference on Business Process Management, pages 497–511. Springer, 2005.

[Rosemann and Van der Aalst 2007] Michael Rosemann and Wil MP Van der Aalst. A configurable reference modelling language. Information Systems, 32(1):1–23, 2007.

[Schnieders and Puhlmann 2006] Arnd Schnieders and Frank Puhlmann. Variability mechanisms in e-business process families. Proceedings of the 9th International Conference on Business Information Systems , 85:583–601, 2006.

[Tseitin 1986] G.S. Tseitin. On the complexity of derivations in the propositional calculus. In H.A.O. Slesenko, editor, Structures in Constructives Mathematics and Mathematical Logic, Part II, pages 115–125, 1968.

[van der Aalst et al. 2010] Wil MP van der Aalst, Marlon Dumas, Florian Gottschalk, Arthur HM Ter Hofstede, Marcello La Rosa, and Jan Mendling. Preserving correctness during business process model configuration. Formal Aspects of Computing, 22(3-4):459–482, 2010.

[van der Aalst et al. 2012] Wil M.P.van der Aalst, Niels Lohmann, and Marcello La Rosa. Ensuring correctness during process configuration via partner synthesis. Information Systems, 37(6):574–592, 2012.

[Weske 2007] Mathias Weske. Business Process Management: Concepts, Languages, Architectures. Springer, 2007.

[Xiang et al. 2018] Yi Xiang, Yuren Zhou, Zibin Zheng, and Miqing Li. Configuring software product lines by combining many-objective optimization and sat solvers. ACM Transactions on Software Engineering and Methodology (TOSEM), 26(4):14, 2018.

[Zhang et al. 2001] Lintao Zhang, Conor F. Madigan, Matthew H. Moskewicz, and Sharad Malik. Efficient conflict driven learning in a boolean satisfiability Solver. In Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design, pages 279–285. IEEE Press, 2001.