Peer-reviewed version available at Appl. Sci. 2018, 8, 2008; doi:10.3390/app8102008

Article

Business Process Configuration according to Data Dependency Specification

Luisa Parody [1,†*], María Teresa Gómez-López [2,†], Angel Jesús Varela-Vaca [2,†] and Rafael M. Gasca[2,†]

1 Universidad Loyola Andalucía; mlparody@uloyala.es
2 Universidad de Sevilla; {maytegomez, ajvarela, gasca}@us.es
* Correspondence: mlparody@uloyala.es; Tel.: +34-955-641-659
† These authors contributed equally to this work.

Abstract: Configuration techniques have been used in several fields, such as the design of business process models. Sometimes these models depend on the data dependencies, being easier to describe *what* has to be done instead of *how*. Configuration models enable to use a declarative representation of business processes, deciding the most appropriate work-flow in each case. Unfortunately, data dependencies among the activities and how they can affect the correct execution of the process, has been overlooked in the declarative specifications and configurable systems found in the literature. In order to find the best process configuration for optimizing the execution time of processes according to data dependencies, we propose the use of Constraint Programming paradigm with the aim of obtaining an adaptable imperative model in function of the data dependencies of the activities described declarative.

Keywords: Business Process Modelling; Declarative Model; Imperative Model; Model Configuration; Constraint Programming

1. Introduction

A business process, henceforth referred to as BP, consists of a set of activities that are executed in coordination within an organizational and technical environment. These activities jointly attain a business goal [1]. Several languages propose an imperative representation of business processes, whose specification allows business experts to describe an explicit order of execution between activities, and to transform the process into an executable model (for example, that activities A, B and C are executed sequentially, or activities D and E are executed in parallel) [2], [3], [4], [5], and [6]. Sometimes the order of the activities is therefore derived from the data dependencies, being possible different configurations. In those cases, the imperative languages fail in the consideration of the data dependencies between activities. Configurable work-flows permit to include a degree of freedom about how the activities are related according to their data dependencies.

Configurable models tend to use a declarative representation of the configurable aspects. Although the imperative process models are significantly more understandable than declarative models [7] [8], it is sometimes very difficult or even impossible to describe a problem in an imperative way [9] and [10]. One of the cases in business processes, where this difficulty can be found, is when the data relation can determine different activities order. The variation of the model is the reason why several authors have proposed languages for the definition of BPs as declarative models [11], [12], [13], and [14]. Unfortunately, the data dependencies and its relation with declarative description has not been faced up.

This paper develops two ideas: (1) propose a declarative description of the activities that form a BP, including the data dependencies between them; (2) based on this declarative model, it is automatically transformed into an imperative model where the work-flow is defined to minimize the execution time of the instances when the process is executed, and keeping the data dependencies

Peer-reviewed version available at Appl. Sci. 2018, 8, 2008; doi:10.3390/app8102008

relations. In order to obtain the imperative model, we propose the use of Constraint Programming, an Artificial Intelligence Technique.

The analysis of the data interchanged during the process execution can affect to two aspects to optimize a BP: (i) attainment of a better business product, for example, by reducing the costs or satisfying customers' preferences; and, (ii) the reduction of the execution time of the instances. The problem of determining the best product in terms of the input and output data of the activities is presented in [15]. In these studies only the first type of aspect is solved, specifically, the way in which the input data can influence the successful execution of a process is analyzed in order to find an optimized product. On the other hand, in the proposal presented in this paper, these previous works are extended to include the second aspect, the optimization of the execution of the instances regarding the data dependencies. How to order the activities by considering these data dependencies in an imperative representation is a hard and difficult task. In order to support business experts in the design task, we propose an automatic transformation from the declarative model into an optimal imperative model where the execution time is minimized. The main reason of transforming declarative model into an imperative model is that imperative models are easier to implement in commercial Business Process Management Systems (BPMS).

Therefore, the aim of this article is to present a framework capable of obtaining the model configuration of the imperative description from a declarative specification based on data dependencies using Constraint Programming. This configurable system must also be able to detect errors and omissions in the declarative model, and to reason and validate, through the different stages of the transformations. To the best of our knowledge, there are no proposals of declarative specification focused on data management, that create optimal executable BP using techniques of configuration.

The rest of the paper is organized as follows: Section 2 presents the example used in the paper to set out the proposal. Section 3 describes related work. Section 4 presents the Conf-BP Framework. Each individual component of the framework is then described in detail. Section 5 formalizes the declarative specification and applies it to the example. Section 6 proposes the automatic transformation of the declarative configurable model into a imperative BP. Section 7 defines the configurable system in charge of the transformation from declarative to imperative modelling using Constraint Programming, and applies it to the example. Results are presented by applying the automatic transformation to the example in Section 8. Finally, conclusions are drawn and future work is proposed in Section 9.

2. Detailing an Example

One clear example, where the fundamental aspect is the successful execution of a BP, arises in the form of the organization of a trip. Generally, the customer searches by hand on the Internet for the cheapest combination of flight and hotel, for specific dates and cities. In addition, if the results obtained remain unsatisfactory, then other dates are sought until a convenient combination is found. Moreover, if necessary and also cheaper, a car can be rented in order to drive to another city to take a flight from another airport, thereby expanding the range of cities of departure and arrival.

In order to obviate the search of several combinations by hand, it is possible to combine the activities that represent the three providers (Hotel, Flight, and Car Rental Providers) into a single BP. The question becomes how to configure the activities in a business process since the input and output are related between them. Therefore, the model of the BP implies two difficulties:

- **To find the input of the activities that optimize the trip by means of minimizing the price**: The process has to provide customers with various possibilities of trips with flights, hotel and car rental (if necessary), while taking into account the existing combinations between a set of possible dates and airports. In addition, there is an objective function to optimize which selects only one of all the possible combinations obtained by the process. An evaluation and a proposal of this part of the problem is analyzed in [15].

Peer-reviewed version available at Appl. Sci. 2018, 8, 2008; doi:10.3390/app8102008

- **To obtain an imperative model that minimizes the execution time of the BP taking into account the data dependencies**: Although the activities to enjoy during the trip are: *"take a car to go to the airport"*, *"catch the flight"* and *"arrive to the hotel"*, the process of booking each part of the trip does not have to follow the same sequence. If the input data of each activity were known, all the activities related to the provider could be executed in parallel. The problem arises when certain activity inputs are related to other activity outputs, or when the activities are executed or not depending on the information obtained from activities executed previously. The objective of this paper is the use of artificial intelligence techniques based on Constraint Programming in order to create an imperative model where the data dependencies are taken into account to minimize the execution time of each instance. For example, it is possible for a flight to arrive at its destination on a different day to when it takes off (overseas route), and therefore the check-in date in the hotel cannot be determined until this information is known. The question is: how to configure a business process formed by these activities to minimize the execution time of the instances? This implies the analysis of every combination of activities and to select the control flow gates that relate them.

3. Related Work

The configuration problems have been used in various industrial application fields, such as computer networks, electrical engineering, telecommunication, financial services, or surveillance [16]. Furthermore, configuration has long been part of the field of Artificial Intelligence. Certain attempts to formalize configuration have been proposed in [17] and [18]. Although many studies relate the term "configuration" to the setting of parameters [19] [20], we focus on the definition of configuration which consists of finding sets of specific objects that satisfy the properties of a given model, such as in [21], [22], and [23].

In this paper, we propose the application of the configuration techniques based on artificial intelligence to business process models. These techniques apply declarative knowledge representation and reasoning methods based on Constraint Satisfaction Problems [24]. Unfortunately, BPs are not typical configuration problems oriented to execute simple tasks [25]. This lack of simplicity is due to the fact that different control flow patterns must be combined in an unknown way taking into account the possible instances that will be executed at runtime.

Several perspective can be analyzed in configuration area [26]. Configuration in Business Processes [27] deals with the problem of managing families of business processes, i.e. business processes that are similar to one another in many ways, yet differ in some other ways from one organization, project or industry to another. This problem arises for example in the context of multinational companies that need to localize their business processes to different legislation, compliance regulations, quality requirements, etc. It also manifests itself in the context of acquisition projects, where an organization needs to merge their own processes with the ones of the acquired organization [28]. In the case of the data relation, implies the possibility to create business process models only modifying the data input and output relation and keeping the rest of information. On the other hand, Vanderfeesten et al. in [29] and [30] define a product-based workflow support to create an optimized process based on a set of recommendations. They firstly analyze the data elements to combine them into a product data model. Then, they select a strategy and configure a product-based workflow system. However, the resulting product model does not make any choice about the ordering of activities.

Related to the languages used to represents declarative and imperative models. There is quite a long history of research dedicated to the development of configuration knowledge representation language [31]. In the case of Business Processes, several declarative languages have been created in the last years, as analyzed in [32].

The transformation from declarative to imperative models with data dependencies has never been studied as a configuration problem. However, some papers in literature can be found about the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 October 2018 doi:10.20944/preprints201810.0076.v1

Peer-reviewed version available at Appl. Sci. 2018, 8, 2008; doi:10.3390/app8102008

transformation between models in BPs. Kulza and Honkisz in [33] transform a Semantic of Business Vocabulary and Rules (SBVR) [34] into a BPMN model. The work is oriented to the transformation of business rules rather than to the dependencies between data. In [35], Natschläger et al. extend BPMN with Deontic Logic. The extension aims to improve the readability and the overall structural complexity, and avoid duplication. The extension proves that a transformation by applying graphs is trusted. In [36], Wiśniewski et al. develop an approach by providing a method for business process modelling. The proposal consists of collecting data coming from different participants and merging it into one declarative specification of performed tasks. Based on this semi-formal description, authors generate synthetic logs which are then used to obtain the BPMN model. Although their BPMN composition is also based on graphs, the main difference with our proposal lies in that they do not take into account data dependencies for the process modeling.

Although a certain number of these papers are focused on defining the interaction between various participants in order to achieve business goals, none of them deals with the type of problems presented in this work: the creation of an imperative model to minimize the BP execution time according to data dependencies.

4. Conf-BP Framework: A Configurable System to create Imperative BP

In order to generate an imperative model, we propose a framework, called Conf-BP (Configuration of Activities in Business Processes). Conf-BP creates automatically an imperative model from its data declarative description. Therefore, the main objective of Conf-BP is to solve the data dependencies where a business expert knows *what* is required (since the specification of the problem is given), but not the specific work-flow to know *how* to obtain it in an optimal way. The configurable model specifies the activities involved and the data relation between them. The relation between the activities is provided through the relationships and constraints of the data dependencies. Once all these features are specified, the framework is used to analyze the relationships between the activities, and to create the work-flow for an imperative model. The standard Business Process Model and Notation (BPMN) is used to represent the imperative model, since it can be enacted in any of the existing commercial BPMSs.

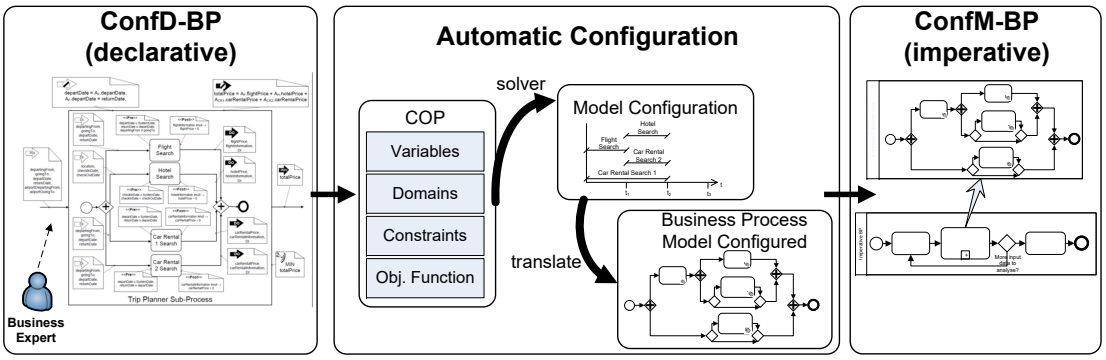

Figure 1. Conf-BP Architecture.

As shown in Figure 1, Conf-BP Declarative Specification (ConfD-BP) represents the highest level of abstraction, where the experts specify the model. The second stage consists of a configuration system in charge of the transformation from the declarative specification into a specific model for a particular imperative language, BPMN in our case. This transformation is performed in two steps: (1) the definition of a Constraint Optimization Problem which obtains the time interval relations between the activities, and (2) the application of an algorithm that builds the work-flow model. This algorithm creates the work-flow that minimizes the execution time of the process. Finally, Conf-BP Imperative

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 October 2018 doi:10.20944/preprints201810.0076.v1

Peer-reviewed version available at Appl. Sci. 2018, 8, 2008; doi:10.3390/app8102008

Modelling (ConfM-BP) completes the third stage, by deploying this configuration into a BPMS. A more detailed description of the Conf-BP stages is presented below.

5. Conf-BP Declarative Specification (ConfD-BP). A formalization of the language

The first stage of the framework is the specification of the configurable model in a declarative way [15]. In this section, the parts of the configurable declarative model are detailed. The elements are shown in Figure 2, and they are described following. As the configurable system can be combined with other elements in a more complex work-flow, we provide the way to include the configurable description into a sub-process.

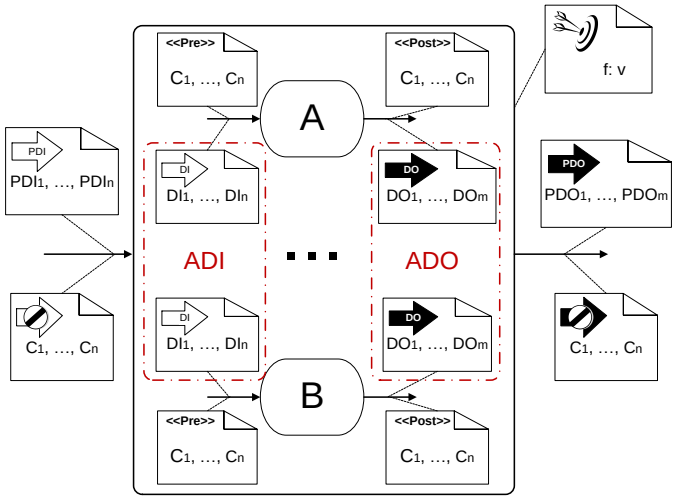

Figure 2. Parts of the Configurable Declarative Description [15].

In order to introduce the formalization of the data provided and combined (see Figure 2), the different descriptions to include are divided into: (i) Sub-process description, including the descriptions of the components associated with the activities of the imperative part, detailed in Section 5.1; and (ii) Sub-process relationships, which is the declarative description of the relationships between the components through the data-flow ($\mathbb{DF}$), addressed in Section 5.2.

5.1. Sub-process description

The *sub-process description* includes the components associated with the activities, gateways, and control-flow which are known and can be represented in an imperative way. Taking $\mathbb{A}$ as the finite set of activities $\{A_1, \ldots, A_i, \ldots, A_n\}$ contained in a determined BP, the following definitions are introduced.

Definition 1. $\mathtt{ACTIVITIES\_DATA\_INPUT(ADI)}$ and $\mathtt{ACTIVITIES\_DATA\_OUTPUT(ADO)}$ represent the sets containing, respectively, all the input data and output data involved in the execution of all the activities.

Within the set $\mathtt{ADI/ADO}$, containing all the input (output) variables, it is possible to identify the input (output) variables of each activity, defined as follows.

Definition 2. $\mathtt{DATA\_INPUT}\ (A_I)$ and $\mathtt{DATA\_OUTPUT}\ (A_I)$ describe the set of input and output data of an activity (A_I) involved in the sub-process, respectively, so that

$$\forall A_i, \mathtt{DATA\_INPUT}\ (A_i) \subseteq \mathtt{ADI}, \text{ where } \mathtt{ADI} \subset \mathbb{DF} \text{ and}$$
$$\forall A_i, \mathtt{DATA\_OUTPUT}\ (A_i) \subseteq \mathtt{ADO}, \text{ where } \mathtt{ADO} \subset \mathbb{DF}$$

Peer-reviewed version available at Appl. Sci. 2018, 8, 2008; doi:10.3390/app8102008

Since various activities can share the same data input (or output), then it is also possible that:

$$\mathbb{DATA\_INPUT}\ (A_i) \cap \mathbb{DATA\_INPUT}\ (A_j) \neq \emptyset, \text{ for i}\neq\text{j and}$$
$$\mathbb{DATA\_OUTPUT}\ (A_i) \cap \mathbb{DATA\_OUTPUT}\ (A_j) \neq \emptyset, \text{ for i}\neq\text{j}$$

The union of all the $\mathbb{DATA\_INPUT}$ sets and all the $\mathbb{DATA\_OUTPUT}$ sets of all the activities, constitute the sets $\mathbb{ADI}$ and $\mathbb{ADO}$ respectively, with non-repetitive elements.

$$\mathbb{ADI} = \{\mathbb{DATA\_INPUT}\ (A_1) \cup \ldots \cup \mathbb{DATA\_INPUT}(A_n)\}$$
$$\mathbb{ADO} = \{\mathbb{DATA\_OUTPUT}(A_1) \cup \ldots \cup \mathbb{DATA\_OUTPUT}\ (A_n)\}$$

Likewise, the specific variables that are inputs or outputs of the overall sub-process can be distinguished. They represent the information that flows from the customer to the sub-process and vice versa:

Definition 3. $\mathbb{PROCESS\_DATA\_INPUT}$ $(\mathbb{PDI})$ is the set of input variables of the sub-process, which determines the information provided and defined by the customer. $\mathbb{PDI}$ is composed of a subset of variables of $\mathbb{ADI}$.

$$\mathbb{PDI} \subset \mathbb{ADI}$$

Definition 4. $\mathbb{PROCESS\_DATA\_OUTPUT}$ $(\mathbb{PDO})$ is the set of output variables of the sub-process. $\mathbb{PDO}$ is composed of a subset of variables of $\mathbb{ADO}$.

$$\mathbb{PDO} \subseteq \mathbb{ADO}$$

5.2. Sub-process relationships

There exist different relationships between the data inputs and data outputs defined in the *sub-process description*. Those relationships are expressed as constraints, both at activity and at process level, giving rise to the *sub-process relationships*, with the following definitions:

Definition 5. $\mathbb{PRE}\ (A_i)$ is the set of constraints that limits the specific values of the $\mathbb{DATA\_INPUT}$ (A_i) that must be satisfied to execute activity A_i. Likewise, $\mathbb{POST}\ (A_i)$ is the set of constraints that limits the specific values of the $\mathbb{DATA\_OUTPUT}\ (A_i)$ that must be satisfied after the execution of activity A_i.

Definition 6. $\mathbb{INPUT\_CONSTRAINT}$ ($\mathbb{OUTPUT\_CONSTRAINT}$) relates the values of variables of $\mathbb{PDI}/\mathbb{PDO}$ with variables of $\mathbb{DATA\_INPUT}/\mathbb{DATA\_OUTPUT}$ can take according to the $\mathbb{PDI}/\mathbb{PDO}$ values in each instance, and the possible values between the input and output of the activities.

Definition 7. $\mathbb{OBJECTIVE\_FUNCTION}$ $(\mathbb{OBJ\_FUNC})$ is an optimization function defined in terms of the data output of the activities ($\mathbb{ADO}$).

The objective of this optimization function is either to maximize or to minimize some of the output data that represents the business product, which constitutes the outcome of the process.

$$\mathbb{OBJ\_FUNC} : f(v \subseteq \mathbb{ADO}) \rightarrow value, \text{ where } f \text{ is } MAX \text{ or } MIN$$

The result of this optimization problem is a set of input values that satisfies the objective function, the pre and post-conditions of the activities, and also satisfies the input and output constraints.

The set of input values that optimizes the output is found at runtime. The role of the constraints is to determine the possible values that this input and output data can take. All the constraints ($\mathbb{INPUT\_CONSTRAINTS}$, $\mathbb{OUTPUT\_CONSTRAINTS}$, $\mathbb{PRE}$, and $\mathbb{POST}$) are always defined at design time by a business expert who is familiar with the problem, although they are solved for

Peer-reviewed version available at Appl. Sci. 2018, 8, 2008; doi:10.3390/app8102008

each instance at runtime. Among every possible tuples of solutions that satisfy the constraints, the outcome of the sub-process will be the one which optimizes the variable defined in the optimization function.

5.3. Grammar

The elements used in the declarative description, describe the data relationship by means of numerical constraints. These constraints are defined by the following grammar [15], where `Variable` and `Constant`, which represent the constant value for the variables, can be defined using Integer, Natural, Float, or String domains. On the other hand, `Set` can be defined as a set of `Constant` values of a specific `Variable`. The constraints can be IF-THEN constraints, which are equivalent to the implication logic operator ($\rightarrow$). IF-THEN constraints are used to represent that the relation of values between some data is conditioned to the value of others. The grammar of constraints is the following:

```
Constraint := 'IF' General_Constraint
'THEN' General_Constraint
  | General_Constraint
General_Constraint := Atomic_Constraint BOOL_OP General_Constraint
  | Atomic_Constraint
  | '¬' Constraint
  | Variable SET_FUNCTION Set
BOOL_OP:= '∨' | '∧'
SET_FUNCTION:= '∈' | '∉'
Atomic_Constraint:= function PREDICATE function
function:= Variable FUNCTION_SYMBOL function
  | Variable
  | Constant
PREDICATE:= '=' | '≠' | '<' | '≤' | '>' | '≥'
  {For the String domain only '=' and '≠' are allowed }
FUNCTION_SYMBOL:= '+' | '−' | '*' | '/'
  {These operators are only applicable to Numerical variables}
```

5.4. Specification Applied to the Trip Planner Example

In the trip planner example, the activities involved in the model share some data, being necessary to create a work-flow aware data input and output. Figure 3 shows the trip example following the formalization detailed in previous section and the notation described in [15].

Hence, there are eight $\mathbb{PDI}$ whose values are given by the customer:

- **departingFrom:** city from where the customer departs.
- **goingTo:** destination city.
- **departDate:** the day that the customer prefers to depart.
- **returnDate:** the day that the customer prefers to return.
- **airportDepartingFrom:** the departure airport to catch the flight.
- **airportGoingTo:** the arrival airport for the flight.

Four different activities are combined in order to perform the package trip offered to customers. This package trip is composed of flights, hotel rooms and, if necessary, the renting of a car to drive to an alternative departure airport, or from the arrival airport to the destination city. Each activity calculates the price as output of the activities for each data input. For the example, the activities (A_i) and their $\mathbb{ADI}$ are:

- Flight Search Activity (A_F) returns the price of flights for a tuple of values for the data input.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 October 2018 doi:10.20944/preprints201810.0076.v1

Peer-reviewed version available at Appl. Sci. 2018, 8, 2008; doi:10.3390/app8102008

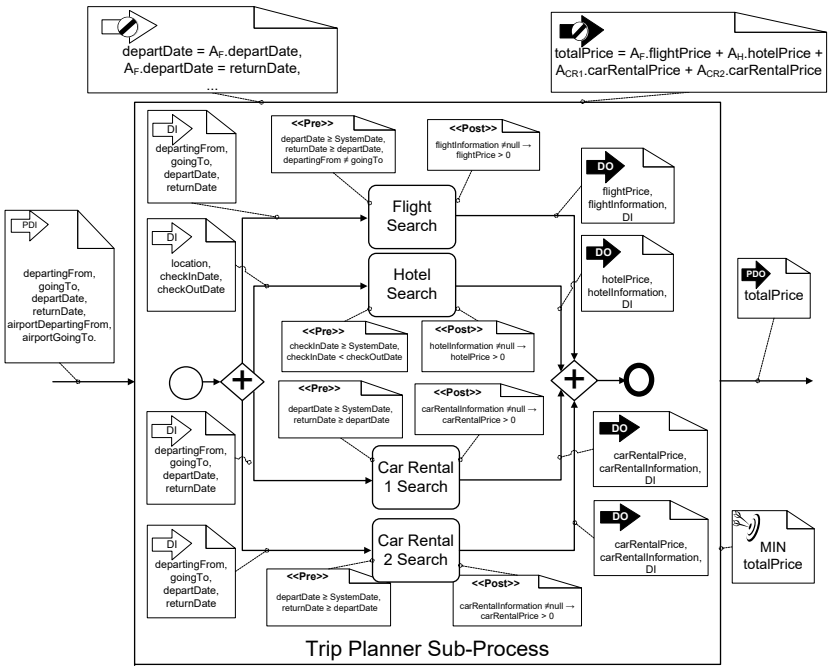

Figure 3. Example of Trip Planner Formalization [15].

$\mathbb{DATA\_INPUT}(A_F)$ = {departingFrom, goingTo, departDate, returnDate}
$\mathbb{DATA\_OUTPUT}(A_F)$ = {flightPrice, flightInformation, $\mathbb{DATA\_INPUT}(A_F)$} where flightInformation = {outwardArrivalDate, returnArrivalDate, seat, number, ...}

The pre- and post-conditions of the activity A_F are:

$\mathbb{PRE}(A_F) = departDate \geq SystemDate\ AND\ returnDate \geq departDate\ AND\ departingFrom \neq goingTo$
$\mathbb{POST}(A_F) = flightInformation \neq null \rightarrow priceFlight > 0$

- Hotel Search Activity (A_H) is employed to ascertain the cost of booking a hotel room.

  $\mathbb{DATA\_INPUT}(A_H)$ = {location, checkInDate, checkOutDate}
  $\mathbb{DATA\_OUTPUT}(A_H)$ = {hotelPrice, hotelInformation, $\mathbb{DATA\_INPUT}(A_H)$}

  The pre- and post-conditions of the activity A_H are:

  $\mathbb{PRE}(A_H) = checkInDate \geq SystemDate\ AND\ checkInDate < checkOutDate$
  $\mathbb{POST}(A_H) = hotelInformation \neq null \rightarrow hotelPrice > 0$

- Car Rental Search Activities (A_{CR1} and A_{CR2}) are employed to determine the price of renting a car. Two cars can be rented during the trip, one at the source (A_{CR1}) and another at the destination (A_{CR2}). Nevertheless, the price of renting both cars is represented by A_{CRx}, where $x = \{1 \vee 2\}$, depends on these entries:

  $\mathbb{DATA\_INPUT}(A_{CRx})$ = {departingFrom, goingTo, departDate, returnDate}
  $\mathbb{DATA\_OUTPUT}(A_{CRx}, A_{CRx})$ = {carRentalPrice, carRentalInformation, $\mathbb{DATA\_INPUT}(A_{CRx})$}

  The pre- and post-conditions of the activity A_{CRx} are:

  $\mathbb{PRE}(A_{CRx}) = departDate \geq SystemDate\ AND\ departDate < returnDate$
  $\mathbb{POST}(A_{CRx}) = carRentalInformation \neq null \rightarrow carRentalPrice > 0$

Therefore, the output of the process, $\mathbb{PDO}$, are the outputs of the activities, which contain the information about the various components of the trip, as well as the total price of the trip ($totalPrice$).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 October 2018 doi:10.20944/preprints201810.0076.v1

Peer-reviewed version available at Appl. Sci. 2018, 8, 2008; doi:10.3390/app8102008

The customer only provides data to the sub-process: the eight $\mathbb{PDI}$. Then, each search activity only uses the necessary data, as detailed in its specification, for the searching. On the one hand, the sub-process can have input data that is not used by some search activity: for example, the Flight Search Activity takes the possible dates and cities given by the customer to the sub-process, meanwhile, the Hotel Search Activity takes the possible dates, the destination city and the preferences. The constraints between the input data of the sub-process and the activities are specified as $\mathbb{INPUT\_CONSTRAINTS}$ at design time. In the same way, the sub-process returns different data with respect to those returned by the activities, and it is necessary to define, at design time, how these data are calculated and related with the output data of the activities. Therefore, the $\mathbb{OUTPUT\_CONSTRAINTS}$ define these relationships between these output data of the activities and the data returned by the sub-process to the customer. Although in the experimental evaluation all the constraints have been included, only the most representative constraints have been formulated in this section. Regarding the relationships between the data input and output that belong to the sub-process and the activities, the most representatives $\mathbb{INPUT\_CONSTRAINTS}$ and $\mathbb{OUTPUT\_CONSTRAINTS}$ are detailed below:

There are several numerical constraints that relate data input and output belonging to the process and the activities. Some of the constraints are defined below:

- $\mathbb{INPUT\_CONSTRAINTS}$:
  - The constraints that establish the values of departure date ($C1$) and return date ($C2$) of the flights have to coincide with the input data proposed by the customer.

    $(C1)\ departDate = A_F.departDate$
    $(C2)\ returnDate = A_F.returnDate$

  - The constraints that describe the values of the departure airport ($C3$) and arrival airport ($C4$) of the flight have to coincide with the input data proposed by the customer.

    $(C3)\ A_F.departingFrom = airportDepartingFrom$
    $(C4)\ A_F.goingTo = airportGoingTo$

  - The date of check-in into the hotel should coincide with the arrival date of the outward flight ($C5$).

    $(C5)\ A_H.checkInDate = A_F.outwardArrivalDate$

  - If the flight does not depart from the departure location ($C6$), then the rental of a car (CR_1) is necessary .

    $(C6)\ departingFrom \neq airportDepartingFrom \rightarrow A_{CR1}.departingFrom = departingFrom \wedge$
    $A_{CR1}.goingTo = A_F.departingFrom \wedge A_{CR1}.departDate = A_F.departDate \wedge$
    $A_{CR1}.returnDate = A_F.returnArrivalDate$

  - If the flight arrives at the destination city $(C7)$, then it is not necessary to rent a car at the destination city.

    $(C7)\ goingTo = A_F.goingTo \rightarrow A_F.goingTo = A_H.location$

  - If the flight does not depart from the departure location ($C8$), then the rental of a car (CR_2) is necessary.

    $(C8)\ goingTo \neq A_F.goingTo \rightarrow A_{CR2}.departingFrom = A_F.goingTo \wedge A_{CR2}.goingTo = A_H.location$
    $\wedge A_{CR2}.departDate = A_F.outwardArrivalDate \wedge A_{CR2}.returnDate = A_F.returnDate$

- $\mathbb{OUTPUT\_CONSTRAINTS}$:
  - The total price is the sum of all the prices returned by the activities, as presented in constraint ($C9$).

$$(C9)\ totalPrice = A_F.fightPrice + A_H.hotelPrice + A_{CR1}.carRentalPrice + A_{CR2}.carRentalPrice$$

In this example, the optimization involves the minimization of the total price of the trip, which is composed of the cost of buying flight tickets, staying in a hotel room, and renting cars for the departure and arrival cities.

$$\mathbb{OBJ\_FUNC} : \text{MIN}\ (totalPrice)$$

6. Conf-BP Imperative Modelling (ConfM-BP)

Once the declarative model is described, it can be possible to obtain an imperative model that satisfy the data dependencies. Between the different options, we propose the most optimal, according to the execution time of the instances of the process. In order to model the BP work-flow in an imperative way, the standard BPMN [37] is chosen, since it is supported by several commercial BPMS. The activities will be combined in a sub-process that starts and ends with the corresponding events. The various order combination of the activities will be represented with a sequence or by means of the control flows: parallel, exclusive or inclusive execution.

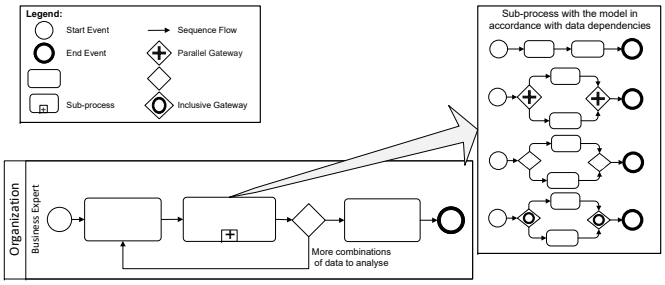

Figure 4. Imperative Representation of the Declarative Model.

Figure 4 shows how the imperative configuration obtained from the declarative description is included in a business process model. The activity *"Supply Input Data Values"* provides the input data values used during the process execution. As was commented before, sometimes the most appropriate input data to optimize the business product are unknown at design time, since it depends on each instance. The used example, the trip planner, has these characteristics, since the dates and city airport must be found, as detailed in [15]. The sub-process *"Execute sub-process"* represents the imperative model that should be created according to the data dependencies described and detailed in the following sections. This sub-process is formed of the set of activities involved in the declarative model by means of the BPMN connections and gateways as detailed in Section 7. Several possible imperative models exist that satisfy the data dependencies, but our objective is to find the optimal model with respect to execution time of any instantiation of the BP.

A simplification of the possibilities of the problem are shown in Figure 5, where only the activities *"Flight Search"* and *"Car Rental 1 Search"* are considered for the modelling configuration. Analyzing the five possibilities we can find: (a) and (b) describe a sequence relation between two activities that can mean that one of the activities has priority order over the another activity, for the example it will be used if 'Car Rental 1' cannot be executed until 'Flight Search' ends or vice versa, and; (c), (d) and (e) represent the execution of one (XOR gateway), more than one (or gateway), or every activities at the same time. For the example it will be used if there not exists an order dependency between the activities. Therefore, which is the best model option to satisfy our trip problem?

This modelling combination and analysis increase considerably as soon as the number of activities grows. Hitherto, this configuration has been made by the a human experts, we propose obtain it automatically in the Conf-BP Framework. Therefore, in order to transform this declarative model into an imperative model that supports any value of input variables of the process by considering the data dependencies, we propose the use of the Constraint Programming paradigm, as explained in Section 7.

7. Automatic Transformation from Declarative to Imperative Model

The transformation from the declarative description into an imperative model is a difficult and hard task, since it implies the analysis of every possible configuration to obtain at design time the most optimal model

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 October 2018 doi:10.20944/preprints201810.0076.v1

Peer-reviewed version available at Appl. Sci. 2018, 8, 2008; doi:10.3390/app8102008

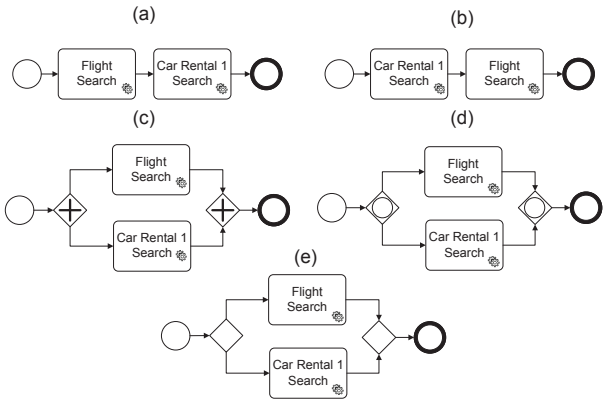

Figure 5. *"Flight Search"* and *"Car Rental 1 Search"* Model Possibilities

at runtime. The difficulty lies in establishing the order of the activities and the control flows components that combine the activities, taking into account the data dependencies. In order to perform the optimal transformation, we propose the three steps shown in Figure 6.

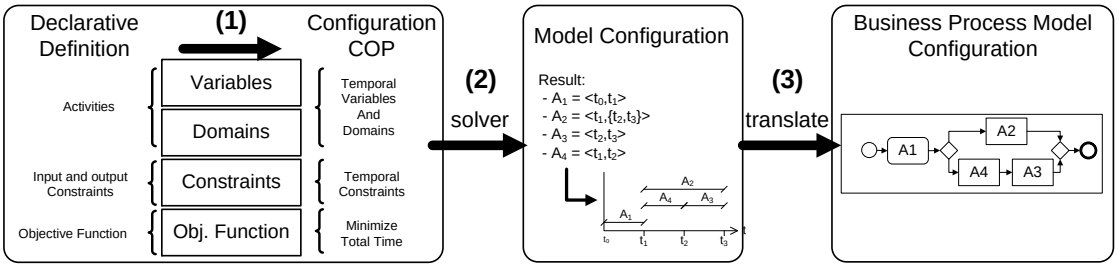

Figure 6. Configuration problem Transformation

1. **Create a Constraint Optimization Problem using the declarative Model**: As explained in detail in Section 7.1, we used Constraint Satisfaction Problems (CSPs) [38] to solve the configuration problem.

   CSPs represent a reasoning methodology consisting of the representation of a problem by means of a set of variables, domains and constraints. CSPs have also a declarative description, then very similar to the Declarative Specification of ConfD-BP. CSPs are a widely used model-based knowledge representation formalism. In a formal way, it is defined as a tuple $\langle X, D, C \rangle$, where $X = \{x_1, x_2, \ldots, x_n\}$ is a finite set of variables, $D = \{d(x_1), d(x_2), \ldots, d(x_n)\}$ is a set of domains of the values of the variables, and $C = \{C_1, C_2, \ldots, C_m\}$ is a set of constraints. A constraint $C_i = (V_i, R_i)$ specifies the possible values of the variables in V that simultaneously satisfy R.Let $V_k = \{x_{k_1}, x_{k_2}, \ldots, x_{k_l}\}$ be a subset of X, and an l-tuple $(x_{k_1}, x_{k_2}, \ldots, x_{k_l})$ from $d(x_{k_1}), d(x_{k_2}), \ldots, d(x_{k_l})$ can therefore be called an *instantiation* of the variables in V_k. An instantiation is a solution if and only if it satisfies the constraints C. If an objective function is included in the CSP, it is called a Constraint Optimization Problem (COP). It is necessary to highlight that CSP and COP specifications are very similar to the process models proposed here, since both are declarative models which define the problem but do not solve it. A solver of CSPs or COPs tries to find a possible tuple of values for the variables that satisfies the constraints defined in the domain.
2. **Solve the COP**: In order to solve the COP created in the second step, it is necessary to analyze the possible values of the variables to find the satisfied tuples. In order to avoid the analysis of every possibilities of values of variables, the solvers use a combination of search and consistency techniques [39] reducing drastically the complexity and time consuming. The consistency techniques remove inconsistent values from the domains of the variables during or before the search. Several local consistency and optimization

Peer-reviewed version available at Appl. Sci. 2018, 8, 2008; doi:10.3390/app8102008

techniques have been proposed as ways of improving the efficiency of search algorithms. There are several commercial constraint problem solvers. The main different between them is the programming language that they used, and the types of constraints that can be included. For the proof of concept to validate the configuration problem, we have used Jsolver^{TM} [40], although any of the existing CSP solvers could be used, since the constraints that we need to include (explained in Subsection 7.1) are very common and supported by every commercial solvers. The resolution of the COP will obtain a set of time intervals each of them associated to an activity, representing the moment when each activity can start and end.

3. **Translate the COP solution into a BPMN Model**: As explained in detail in Section 7.2, by using the results obtained from the COP, the imperative model can be created. This result has to be interpreted as gateways that relates the activities. For example, if the COP allows that two activities A and B could start at the same instant of time, perhaps there is a constraint which states that only one activity can be executed in each instance. In that case, there can be an exclusive or inclusive gateway relating the two activities. The decision between the possible gateways between the various activities is based on: (a) the study of the domains of the data related in the declarative problem description, and; (b) the data obtained from the resolution of the COP as it is explained in Subsection 7.2.

7.1. Configuration of a BP model: Creating a COP from the declarative model

The relation between the input and output of the activities determines their relational order. For the same declarative model, it is possible to find several configurations of activities that satisfy the requirements and description. To find the best configuration, we propose the use of an intelligent system to find the optimal BP model that maximizes the parallelism of the activities, with the aim of minimizing the execution time. To determine this configuration, we suggest the transformation of the declarative model into a COP, to analyze the possible instant when the activities can start and end their executions, according to data dependency.

Firstly, and before to explain how to create the COP, it is necessary a previous step to ascertain the data relation, specially when they are included in the IF-THEN constraints of the description of the problem. The definition of Tuple of Condition-Relation in order to introduce this relation is introduced.

Definition 8. A Tuple of Condition-Relation is a tuple formed of $\langle Condition, Activity\ 1, Activity\ 2\rangle$, where *Condition* represents under which condition *Activity* 2 is executed after *Activity* 1. Therefore, the value *true* in the *Condition* implies that *Activity* 2 is always executed after *Activity* 1, or, in the case when the *Condition* contains an expression, *Activity* 2 is executed after *Activity* 1 if and only if the expression is met.

These tuples are built analyzing the data relations found in the **INPUT_CONSTRAINTS** and **OUTPUT_CONSTRAINTS** that describe the declarative model. For each constraint where the output of an activity A_i is related to the input of an activity A_j, a tuple is created. If the input-output relation appears in an IF-THEN constraint with a c condition, the tuple will be $\langle c, A_i, A_j\rangle$, and $\langle true, A_i, A_j\rangle$ otherwise.

With the list of intervals and the list of tuples of data relations, the following COP is created:

- **Variables:** The possible instants when each activity can be executed, are defined as the tuple of variables $\langle t_{ini}^{A_i}, t_{end}^{A_i}\rangle$, where $t_{ini}^{A_i}$ is the instant of time in which the activity A_i begins, and $t_{end}^{A_i}$ is the instant of time in which the activity A_i finishes. For each activity A_i, the COP will include a couple of variables $\langle t_{ini}^{A_i}, t_{end}^{A_i}\rangle$. To represent the execution time of the whole process, the variable T is included, that represents the t_{end} of the last activity executed. If every activities will be executed in parallel, T will be the maximum value of t^{A_i}, but if every activities are executed sequentially, the value of T will be the summation of every t^{A_i}.
- **Domain:** The domain of the $t_{ini}^{A_i}$ and $t_{end}^{A_i}$ variables is related to the execution time of each activity. If we suppose that each activity spends t^{A^i} units of time, then the domain of each variable is:
  - $t_{ini}^{A_i}$: $0..(\sum_{1..j..N} t^{A^j}) - t^{A^i}$
  - $t_{end}^{A_i}$: $t^{A^i}..(\sum_{1..j..N} t^{A^j})$
  - T: $\max(t^{A^i})..\sum_{1..j..N} t^{A^j}$
- **Constraints:** The necessary constraint to model the COP are:
  - It is mandatory that for every activity A_i, $t_{ini}^{A_i} + t^{A^i} \leq t_{end}^{A_i}$
  - T has to be the greatest value of t_{end}, then $T \geq t_{end}^{A_i}$ for every A_i.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 October 2018 doi:10.20944/preprints201810.0076.v1

Peer-reviewed version available at Appl. Sci. 2018, 8, 2008; doi:10.3390/app8102008

– For each element of the list of tuples building by using the data relations, a numerical constraint is included in the COP. Each relation between data input and output in the way $\langle c, A_j, A_i \rangle$ is translated into the constraint $t_{ini}^{A_i} \geq t_{end}^{A_j}$. For example, in the trip planner, the constraint $\{A_H.checkInDate = A_F.outwardArrivalDate\}$ is translated into $t_{ini}^{A_H} \geq t_{end}^{A_F}$. Therefore, all the activities could start at the same instant unless there exists a relationship requiring one activity to start after another.

- **Objective:** The optimization function of the COP is the minimization of the total time, it implies to minimize the value of T (minimization(T)).

Table 1 gives a summary of the equivalence of the elements of the formalization and the COP built.

Table 1. Declarative model and COP elements relationship

Declarative	COP
Activity	$\langle t_{ini}, t_{end} \rangle$
PDI, ADI, PDO, ADO	Variables
Numerical Constraints (Input and Output Constraints)	Temporal Constraints
Pre and Post Conditions	Activity Definition
Objective Function	Minimize Total Time

The solving of the COP will obtain a pair of start and end instants of time for each activity that minimizes any instantiation time (T) for the model. As mentioned earlier, the minimum execution time of an activity is modelled as a t^{A^i} units of time. But it is possible to obtain the same value of (T), where the execution time is minimized, with different $t_{ini}^{A^i}$ and $t_{end}^{A^i}$ for the same problem. Suppose activities (A_1, A_2, A_3 and A_4) that last a unit of time, where the input data of A_4 depends on the output of A_1, and the input data of A_3 depends on the output of A_4. Figure 7 depicts two possible configurations (a) and (b) that minimize the theoretical execution time (t_3). However, there exist more variations, since A_2 can be executed in a parallel way with any of the other activities.

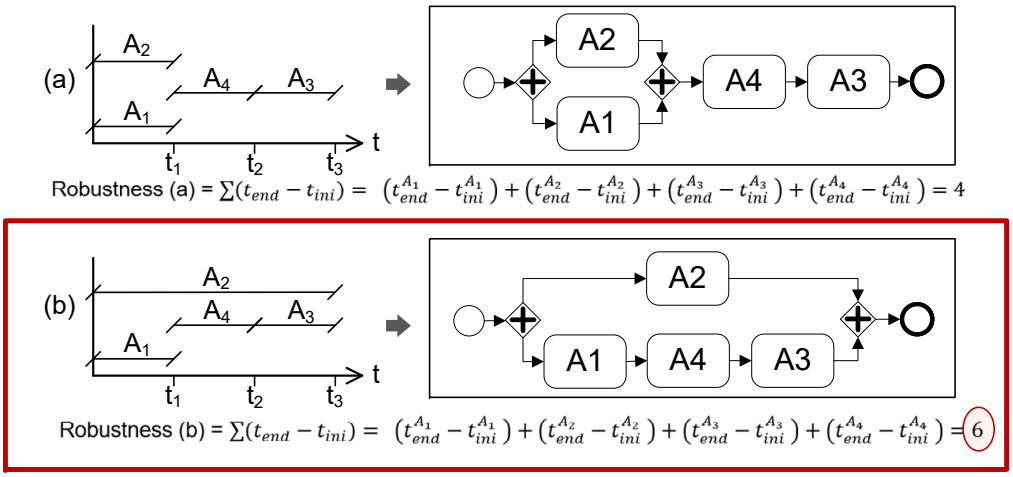

Figure 7. Flexibility of the BP in terms of the execution time of the Activities

Although both options (a) or (b) are optimal according time consuming, the model of Figure 7(b) is better than the model of Figure 7(a), since it supports an unexpected delay of A_2 executing, not increasing the total execution time (t_3).

To make the imperative model more robust to unexpected delays, we propose to modify the proposed COP to find the most paralleled process. To this end, the COP must find the greatest value of $t_{end}^{A^i}$ for each A_i. It is obtained including the $t_{ini}^{A^i}$ variables as a goal in the COP, and leaving the domain of $t_{end}^{A^i}$ open during the search. Then, the $t_{ini}^{A^i}$ variables will be instantiated during the propagation phase, while all the possible values of the

domains of $t_{end}^{A^i}$ variables will be obtained as solutions of the COP for each value of $t_{ini}^{A^i}$. The greatest value of the $t_{end}^{A^i}$ domain is the most appropriate value to build an imperative robust business process model.

Algorithm 1 describes the creation of the COP from a declarative specification as explained before.

Algorithm 1 Creation of a COP from a Declarative Specification

1: Create variable: $int\ TStart = 0$
2: Create variable: $int\ TEnd = 0..\sum_{i=1}^{n} t^{A^i}$
3: **for** each $Activity(A_i)$ **do**
4: Create a variable: $int\ t_{ini}^{A^i} = 0..\sum_{i=1}^{n} t^{A^i}$
5: Create variable: $int\ t_{end}^{A^i} = 0..\sum_{i=1}^{n} t^{A^i}$
6: **end for**
7: **for** each $Activity(i)$ **do**
8: Create constraint: $(t_{ini}^{A^i} + t^{A^i}) \leq t_{end}^{A^i}$
9: Create constraint: $TEnd \geq (t_{end}^{A^i} + t^{A^i})$
10: **end for**
11: **for** each $ADO_i = ADI_j$, where an output of *Activity i* is related to the input of *Activity j* **do**
12: Create constraint: $(t_{end}^{A^i} + t^{A^i}) = t_{ini}^{A^j}$
13: **end for**
14: Define goal variables: $\{ (t_{end}^{A^1} + t^{A^1}), ..., (t_{end}^{A^n} + t^{A^n}), TEnd \}$
15: Define objective function: $Minimize\ (TEnd - TStart)$

7.2. Transformation of the COP results into a BP Imperative Model

The list of time intervals, obtained from the COP, with the most appropriate values for $t_{ini}^{A^i}$ and $t_{end}^{A^i}$ for each activity, has to be translated into an imperative BP model. For the example of Figure 7(b), the obtained list is: $\{A_1[0, t_1]$, $A_2[0, t_3]$, $A_3[t_2, t_3]$, $A_4[t_1, t_2]\}$. There exist various imperative models in business processes, but the most widely used standard language is BPMN [37]. Although the model in the standard is represented by means of an XML file, to facilitate the understanding of the algorithm that obtains the imperative model, we propose the use of a BPMN-Graph. As explained in [1], BPMN is a graph-oriented language in which control and action nodes can be connected. Therefore, we propose to model the BPMN using a directed graph as defined in [41].

The problem is solved in two steps: firstly, the BPMN-Graph is built not specifying the type of gateways, only including which are splits and joins, and; secondly, the uncertain about the gateways is solved analyzing the list of tuples created in function of the IF-THEN constraints of the model, that related the activity execution on function of the value of other variables.

7.2.1. Step 1: Building BPMN-Graph

The proposed BPMN-Graph is a direct graph composed of: (i) *vertexes*, which represent the activities and the gateways; and (ii) *edges*, which represent the sequence flows that join the various vertexes. Firstly, the idea of the algorithm is explained by means of a trace, after which the algorithm is detailed. Analyzing the $t_{ini}^{A^i}$ and $t_{end}^{A^i}$ of each activity, it is possible to discover the activities that have parallel or sequential relation between them. The main idea of the algorithm is to detect these relations building sub-graphs, that will be combined in a sequential or parallel way. An example of the trace of the algorithm is shown in Figure 8. The initial *Problem* is formed by a set of activities $A_1, ..., A_8$, associated to a time interval $\langle t_{ini}^{A^i}, t_{end}^{A^i} \rangle$, and a list of tuples that describe the condition about the activities order dependencies. The algorithm divides the problem in function of the *sequential points* and the *parallel subsets*. Both definitions are introduced:

Definition 9. Sequential point (List of Time Intervals): For a List of Time Intervals, a sequential point is an instant, or set of instants of time at which no activity can be executed. It implies that in a sequential point, only the execution of an activity can start or end, and therefore the problem can be broken into two smaller *problems* that can be executed sequentially.

Definition 10. Parallel subsets (List of Time Intervals, List of Tuples of Relations): For a list of Time Intervals associated with the activities $A_1 \ldots A_n$, the parallel subsets are a set of subset of activities $ps_1, \ldots, ps_m$, where $\nexists$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 October 2018 doi:10.20944/preprints201810.0076.v1

Peer-reviewed version available at Appl. Sci. 2018, 8, 2008; doi:10.3390/app8102008

any activity $a \in ps_i$ and an activity $b \in ps_j$, where $ps_i \neq ps_j$, such that $\exists$ a tuple in the List of Tuples of Relations that $\langle$condition, a, b$\rangle$. A way to obtain the parallel subsets, is creating a parallel subset for each activity (ps_1, ..., ps_n), if there exist an activity $a \in ps_i$ and another activity $b \in ps_j$, both subsets are merged. This process is repeated until no more sets can be merged.

With the aim of facilitating the understanding of the algorithm and our proposal in general, we use the example of Figure 8, with the list of tuples:

- (R1) $\langle A_1.output \leq 50, A_1, A_3 \rangle$
- (R2) $\langle A_1.output < 100, A_1, A_4 \rangle$
- (R3) $\langle true, A_3, A_5 \rangle$
- (R4) $\langle true, A_4, A_5 \rangle$
- (R5) $\langle true, A_5, A_8 \rangle$
- (R6) $\langle true, A_2, A_8 \rangle$
- (R7) $\langle true, A_6, A_7 \rangle$

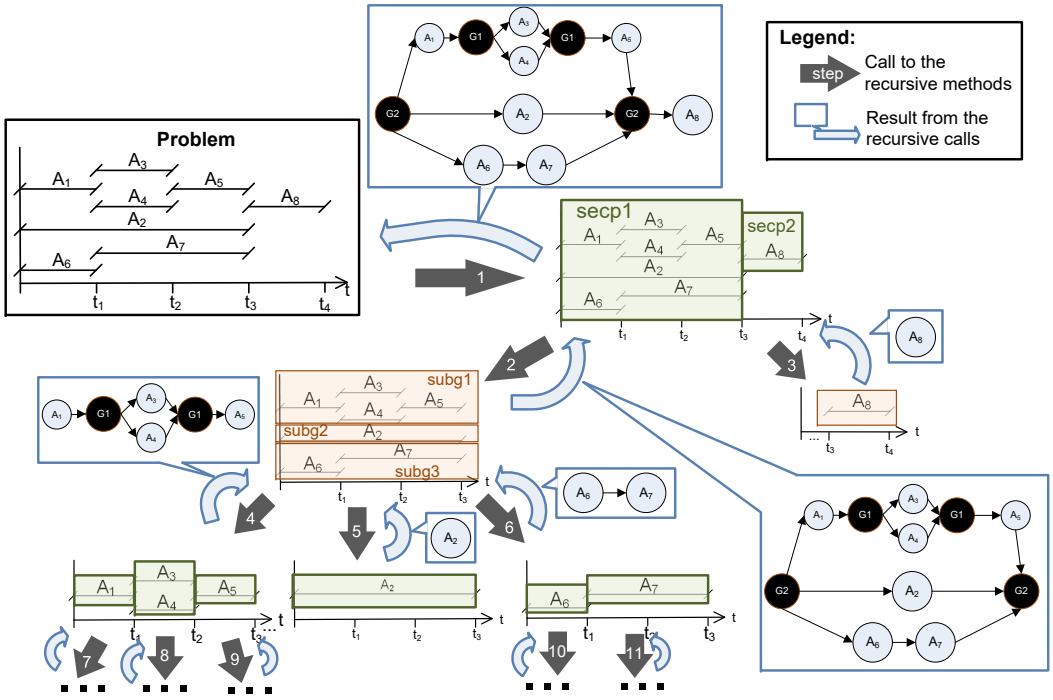

Figure 8. Trace example of the Algorithm to create a BPMN-Graph using the COP results

The trace follows the next steps:

1. **Look for sequential points:** Firstly, the set of activities should be separated by the sequential points (arrow 1 in Figure 8). The separation is carried out by detecting those instants of time where there is a sequential point. The intervals that are derived from these sequential points delimit the subset of activities. In the example, the set of activities is separated into two subsets of activities, since there is only one sequential point in t_3. The first subset $secp1$ goes from t_0 to t_3 and includes all activities except activity A_8. On the other hand, the second subset $secp2$ goes from t_3 to t_4 and only includes activity A_8.
2. **Solving sequential sub-problems:** Each subset of activities is solved as a sub-problem (arrows 2 and 3) and combined sequentially both BPMN-sub-graphs are solved.
3. **Look for parallel subsets:** The $secp1$ is separated in three parallel subsets: $subg1$, $subg2$ and $subg3$. To obtain these subsets, the idea described in Definition 10 is applied:

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 October 2018 doi:10.20944/preprints201810.0076.v1

Peer-reviewed version available at Appl. Sci. 2018, 8, 2008; doi:10.3390/app8102008

(a) Create a subset of each activity: {ps_1: {A_1}, ps_2: {A_2}, ps_3: {A_3}, ps_4: {A_4}, ps_5: {A_5}, ps_6: {A_6}, ps_7: {A_7}}.

(b) Since A_1 is related with A_3 in $R1$, and with A_4 in $R2$, both subsets are merged. Also, A_6 and A_7 are related in $R7$. The obtained parallel subsets are: {ps_1: {A_1, A_3, A_4}, ps_2: {A_2}, ps_5: {A_5}, ps_6: {A_6, A_7}}.

(c) Since A_5 is related with A_3 in $R3$, and with A_4 in $R4$, both subsets are merged. Therefore, the parallel subsets: {ps_1: {A_1, A_3, A_4, A_5}, ps_2: {A_2}, ps_6: {A_6, A_7}} are also obtained.

(d) Tuple relations $(R5)$ and $(R6)$ are not used in the creation of subsets since A_8 is not involved in the parallel analysis.

4. **Solving parallel sub-problems:** The next step implies to solve each sub-problem (arrows 4, 5 and 6), followed by the search of **sequential points**.
5. **Combining parallel sub-problems:** To solve *secp*1 it is necessary to combine the sub-graphs obtained by solving *subg*1, *subg*2 and *subg*3 (return of arrow 4, 5 and 6). These BPMN-sub-graphs are combined as various branches joined by a split and join node ($G2$ in the example).
6. **Combining sequential sub-problems:** The obtained results after a sequential analysis of sub-graphs (returns of arrow 2 and 3) implies joining the results of *secp*1 and *secp*2. To this end, the result (BPMN-Graph) of *secp*1 and *secp*2 are joined by an edge between the final vertex of *secp*1 and the initial vertex of *secp*2, which are $G2$ and A_8 respectively.

Algorithm 2 Create a BP model from COP result

```
1:  procedure SEQUENTIALTREATMENT(Problem p): BPMN-Graph
2:      sol: BPMN-Graph to return
3:      lprob: list of Problems
4:      lsol: list of BPMN-Graphs
5:      if (p.listActivities.size() == 1) then
6:          sol ← p
7:      else
8:          lprob ← separate p by detecting sequential points
9:          for each p ∈ lprob do
10:             lsol.add(parallelTreatment(p))
11:         end for
12:     end if
13:     sol ← link sequentially lsol
14:     return sol
15: end procedure
16: procedure PARALLELTREATMENT(Problem p): BPMN-Graph
17:     sol: BPMN-Graph to return
18:     lprob: list of Problems
19:     lsol: list of BPMN-Graphs
20:     if (p.listActivities.size() == 1) then
21:         sol ← p
22:     else
23:         lprob ← separate p by detecting parallel groups
24:         for each p ∈ lprob do
25:             lsol.add(sequentialTreatment(p))
26:         end for
27:     end if
28:     sol ← link lsol by gateways
29:     return sol
30: end procedure
```

Algorithm 2 details the procedures for the sequential and parallel treatments explained above. Algorithm 2 takes a *Problem* and transforms it into a BPMN-Graph. Initially, the *Problem* is treated sequentially with the algorithm *sequentialTreatment*, which finds the sequential points following Definition 9. Each of the *subproblems* found, has to be analyzed to identify the parallel subsets of activities with the algorithm *parallelTreatment*

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 October 2018 doi:10.20944/preprints201810.0076.v1

Peer-reviewed version available at Appl. Sci. 2018, 8, 2008; doi:10.3390/app8102008

(following Definition 10). The parallel treatment of a *Problem* consists of gathering the activities that are related into subsets and creating new problems from these subsets. The relationships among the activities are given by the tuples of Condition-Relation of the *Problem*. Each *subproblem* is then solved recursively. To combine the parallel subsets, two gateway vertexes are inserted as split and join gateway respectively. Each BPMN-sub-graph will be a branch related by means of the gateway. To combine two sequential BPMN-sub-graphs, only and edge needs to be included between the last node of the first solution and the first node of the second solution. The algorithm draws to a halt when reaching a base case, or when the problem contains only one activity. In both treatments, the solution of a base case is a graph with a unique vertex that represents this activity.

7.2.2. Step 2: Defining the gateways

Once the graph is created, the correct gateways that diverge the sequence flows have to be selected. In the graph, the gateways are identified by means of studying the tuples of Condition-Relation obtained from the constraints, defined in the declarative specification.

The different types of relations described by means of the tuples of Condition-Relation, and how they affect to the type of gateway are explained below:

- **Parallel**: two activities are executed in a parallel way if there are no conditions that relate the t_{ini} of both activities (see Figure 9).

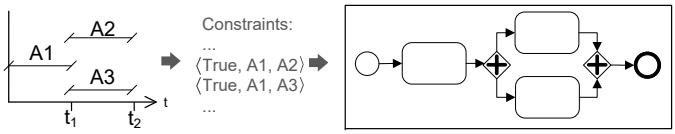

Figure 9. Parallel Relationship

- **Exclusive**: two activities are executed in an exclusive way if the constraints that relate the t_{ini} of the two activities and the domain of these constraints are complete and do not overlap. For example, as shown in Figure 10, if the execution of A_2 and A_3 depends on a value of the output data $A_1.output$, but there are no overlaps (A_2 is executed when $A_1.output$ is less than 50 and A_3 is executed when $A_1.output$ is greater than or equal to 50), then there is an exclusive relationship between them. There is another possibility when only one condition is described, for example only exists the condition (A_2 is executed when $A_1.output$ is less than 50), in that case it means that there are two branches for the XOR gateway, but one of them with no activities.

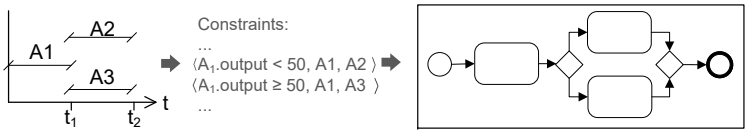

Figure 10. Exclusive Relationship

- **Inclusive**: two activities are executed in an inclusive way if there are conditions that relate the t_{ini} of both activities, and the domain of these conditions are complete and overlapped. For example, as shown in Figure 11, if the execution of A_2 and A_3 depends on a value of the output data $A_1.output$, but there are overlaps between the domains that satisfy the constraints (A_2 is executed when $A_1.output$ is less than 75 and A_3 is executed when $A_1.output$ is greater or equal to 25, and hence both coincide when $A_1.output$ is greater than 25 and less than 75), then there is an inclusive relationship.

Table 2 gives a summary of the resulting gateways depending on the conditions established by the constraints that relate the data of the activities.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 October 2018 doi:10.20944/preprints201810.0076.v1

Peer-reviewed version available at Appl. Sci. 2018, 8, 2008; doi:10.3390/app8102008

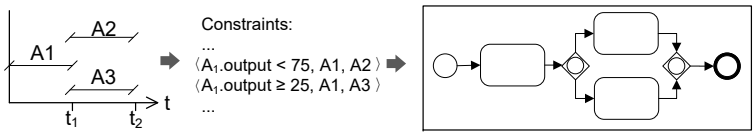

Figure 11. Inclusive Relationship

Table 2. Type of gateway decision

Gateway	Constraints Conditions
Parallel	No dependencies in domains
Exclusive	Domains without overlaps
Inclusive	Domains with overlaps

For the example explained in Figure 8, it is necessary to analyze $G2$ and $G1$ (see Figure 12). Since there are no relations between activities A_1, A_2, and A_6, then $G2$ is a parallel gateway. On the other hand, the execution of activities A_3 and A_4 depends on the outputs of A_1. The conditions specified in relations $R1$ and $R2$ indicate that the domain of the constraints is complete and there are no overlaps, therefore, $G1$ is an exclusive gateway.

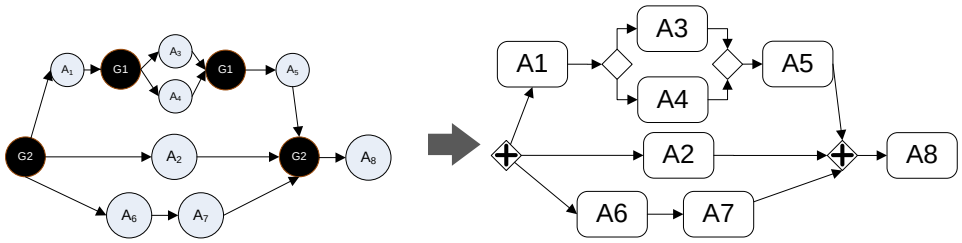

Figure 12. From graph to BPMN Model Example

8. Results: Transformation applied to the Trip Planner Example

In order to illustrate the use of the algorithms, the application of the algorithms to the Trip Planner example is presented in this section.

The first step is to obtain the list of tuples of Condition-Relation according Definition 8. Analyzing the constraints of the example, we can find: (i) Constraints C1, C2, C3 and C4 do not relate input and output of the activities, then they are not involved in the list of tuples of Condition-Relation; (ii) Constraint C5 is used to create the relation: $\langle$ *true*, A_F, A_H $\rangle$; (iii) Constraint C6 is used to create the tuple: $\langle$Condition of $C6$, Start_Event, A_{CR1} $\rangle$, and; (iv) Constraint C8 is used to create the tuple: $\langle$Condition of $C8$, A_F, $A_{CR2}\rangle$}.

After that, the second step is to build the COP as detailed in Subsection 7.1, that for the example will have the form.

- **Variables:** $t_{ini}^{A^F}$, $t_{end}^{A^F}$, $t_{ini}^{A^H}$, $t_{end}^{A^H}$, $t_{ini}^{A^{CR1}}$, $t_{end}^{A^{CR1}}$, $t_{ini}^{A^{CR2}}$, $t_{end}^{A^{CR2}}$, T.
- **Domain:** Supposing that the theoretical execution time of each activity is a unit of time, the domains are:
  - $t_{ini}^{A^F}$, $t_{ini}^{A^H}$, $t_{ini}^{A^{CR1}}$, $t_{ini}^{A^{CR2}}$: 0..4 //Since there are 4 different activities.
  - $t_{end}^{A^F}$, $t_{end}^{A^H}$, $t_{end}^{A^{CR1}}$, $t_{end}^{A^{CR2}}$: 1..4
  - T: 1..4
- **Constraints:**
  - $t_{ini}^{A^F} + 1 \leq t_{end}^{A^F}$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 October 2018 doi:10.20944/preprints201810.0076.v1

Peer-reviewed version available at Appl. Sci. 2018, 8, 2008; doi:10.3390/app8102008

- $t_{ini}^{A^H} + 1 \leq t_{end}^{A^H}$
- $t_{ini}^{A^{CR1}} + 1 \leq t_{end}^{A^{CR1}}$
- $t_{ini}^{A^{CR2}} + 1 \leq t_{end}^{A^{CR2}}$
- $T \geq t_{end}^{A^F} \wedge T \geq t_{end}^{A^H} \wedge T \geq t_{end}^{A^{CR1}} \wedge T \geq t_{end}^{A^{CR2}}$
- For the list of tuples of Condition-Relation mentioned above, the constraints included in the COP are:
  - $t_{ini}^{A^H} \geq t_{end}^{A^F}$
  - $t_{ini}^{A^{CR2}} \geq t_{end}^{A^F}$

- **Goal:** $t_{ini}^{A^F}, t_{ini}^{A^H}, t_{ini}^{A^{CR1}}, t_{ini}^{A^{CR2}}$
- **Objective:** minimization(T).

The obtained values of the variables after the resolution of the COP are:

- $TEnd$: 2
- $t_{ini}^{A^F}$: 0 and $t_{end}^{A^F}$: 1
- $t_{ini}^{A^H}$: 1 and $t_{end}^{A^H}$: 2
- $t_{ini}^{A^{CR1}}$: 0 and $t_{end}^{A^{CR1}}$: [1..2], selecting the greatest value (2).
- $t_{ini}^{A^{CR2}}$: 1 and $t_{end}^{A^{CR2}}$: 2

The resulting BP Model is shown in Figure 13. According to the results, "*Flight Search*" and "*Car Rental 1 Search*" activities start at instant 0. However, the "*Car Rental 2 Search*" activity depends on the value of the output data of the "*Flight Search*" activity, so there is an exclusive gateway before this activity. Since there is not a complete domain with the constraint that determines the execution of the "*Car Rental 2 Search*" activity, then there is a branch by default which indicates that nothing happens when the condition is not met. The same situation occurs with "*Hotel Search*" and "*Car Rental 2 Search*" activities.

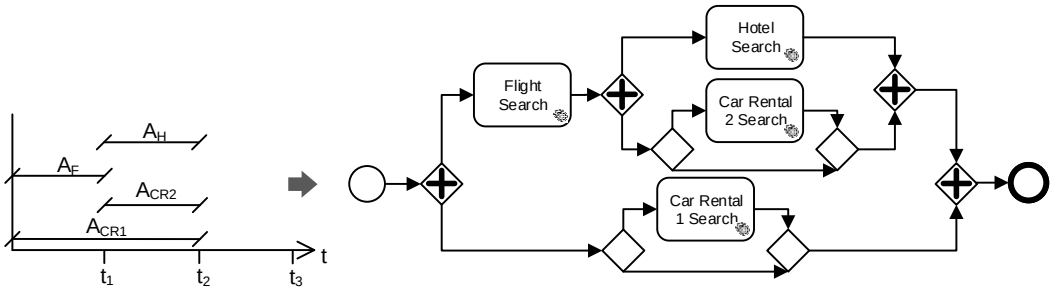

Figure 13. Trip Planner Model Result

The solution parallelizes the activities according to the data input and output relations, being possible execute it in two units of times.

8.1. Empirical Evaluation

Since the generated imperative only depends on the declarative model, it is created only once at design time. The transformation performed involves a COP and two own-developed algorithms. Therefore, the critical points of our proposal lies in the resolution of the COP and these algorithms.

Regarding the resolution of a COP, we use one of the existing commercial tools to evaluate the COP, since the COP resolution has remained a known problem studied by researchers in the area over several decades, and is supported by an important set of tools used in real problems. In general, the time to find the solution of a COP depends on: (1) the type of constraints (lineal, polynomial); (2) the number of constraints; and (3) the number and type of the variables. It is possible to find an analysis in [42] on the complexity of the NP-complete constraint problem resolution according to these characteristics. The complexity of resolution of CSPs depends on the

number of possible solutions of the problem, and on whether it is neither under constraint nor unsolvable. The most complex of these problems are those that are neither under constraint nor unsolvable. For these reasons, no affirmation can be given concerning the efficiency in a generic way of our proposal, since our declarative specification enables any type and any number of constraints; therefore the evaluation time depends on the specific problem. Depending on the number of constraints associated to a COP, and the number of variables, the COP evaluation time remains variable.

On the one hand, in relation to Algorithms 1 and Algorithm 2, the computational order is lineal since it only depends on the number of activities. On the one hand, Algorithms 1 And on the other hand, Algorithm 2 defines two recursive functions which are related between each other. The main idea of this algorithm is to apply a Divide-and-Conquer Method by means of reducing the problem in sub-problems, being the smallest problem, in our case, a single activity. Therefore, in both the best and the worst cases, the computational order is lineal since it depends on the number of activities either they have a parallel or sequential relationships.

With the aim of performing the evaluation, both algorithms are executed over a set of generated test cases. Each test case establishes a different number of $\mathbb{PDI}$ and constraints that relate them. Figure 14 shows the computing time needed to solve both algorithms. The test cases are measured using a PC with an Intel Core i7-2675QM CPU with a 2.2 GHz processor and a 8GB of RAM.

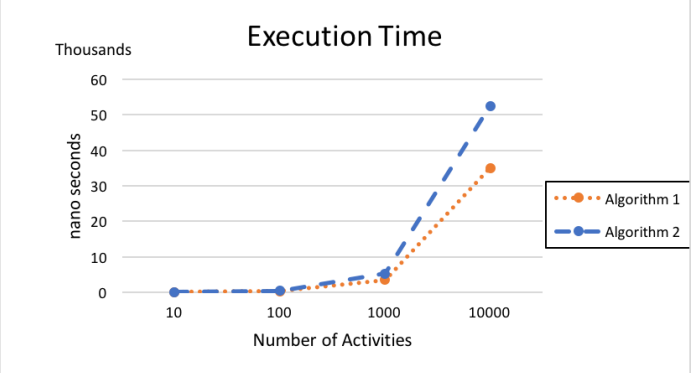

Figure 14. Execution Time of Algorithm 1 and Algorithm 2.

9. Conclusions and Future Work

Declarative languages have been focused on the order of activities, and not on how the data values and dependencies can affect the optimal execution of a process. In this paper, we apply configuration problems ideas to the construction of an imperative model according to the declarative description of the data relation. Our proposal shields the business experts from unnecessary details, and to provide assistance when experts know what they want (the BP requirements by means of data dependencies) but do not know how to attain what they want (establish the order of the activities), Conf-BP Framework is proposed. Conf-BP establishes a methodology to transform a declarative model into an imperative representation using BPMN. Business experts only have to specify the BP requirements, leaving the configuration of the activities in the imperative model to the intelligent system, which uses the Constraint Programming paradigm and a set of proposed definitions and algorithms. This automatic methodology obtains an imperative model described by BPMN where the optimal model related to execution time is obtained. In addition, the framework is applied to an example: the process of organising a trip. Future work into this area involves increasing the capabilities of Conf-BP by including loops within the model, where there exists a cyclic relation between the data.

Acknowledgments: This work has been partially funded by the Ministry of Science and Technology of Spain (TIN2015-63502-C3-2-R) and the European Regional Development Fund (ERDF/FEDER).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ADI: Activities Data Input
ADO: Activities Data Output
BP: Business Process
BPEL: Business Process Executation Language
BPM: Business Process Management
BPMN: Business Process Model and Notation
BPMS: Business Process Management System
Conf-BP: Configuration of Activities in Business Processes
ConfD-BP: Conf-BP Declarative Specification
ConfM-BP: Conf-BP Imperative Modelling Specification
COP: Constraint Optimization Problem
CSP: Constraint Satisfaction Problem
OBJ_FUNC: Objective Function
PDI: Process Data Input
PDO: Process Data Output

Bibliography

1. Weske, M. *Business Process Management: Concepts, Languages, Architectures*; Springer, 2007.
2. Aguilar-Saven, R.S. Business process modelling: Review and framework. *International Journal of Production Economics* **2004**, *90*, 129–149.
3. Tsai, A.; Wang, J.; Tepfenhart, W.; Rosea, D. EPC Workflow Model to WIFA Model Conversion. Proceedings of IEEE International Conference on Systems, Man and Cybernetics. IEEE International Conference on Systems, Man and Cybernetics, 2006. SMC '06., 2006, Vol. 4766/2007, pp. 2758 – 2763.
4. Sinogas, P.; Vasconcelos, A.; Caetano, A.; Neves, J.; Mendes, R.; Tribolet, J.M. Business Processes Extensions to UML Profile for Business Modeling. ICEIS (2), 2001, pp. 673–678.
5. List, B.; Korherr, B. A UML 2 Profile for Business Process Modelling. ER (Workshops), 2005, pp. 85–96.
6. Bosilj-Vuksic, V.; Hlupic, V. Petri Nets and IDEF diagrams: Applicability and efficacy for business process modelling. *An International Journal of Computing and Informatics* **2001**, *25*, 123–133.
7. Pichler, P.; Weber, B.; Zugal, S.; Pinggera, J.; Mendling, J.; Reijers, H.A. Imperative versus Declarative Process Modeling Languages: An Empirical Investigation. Business Process Management Workshops (1). Springer, 2011, Vol. 99, *Lecture Notes in Business Information Processing*, pp. 383–394.
8. Zugal, S.; Soffer, P.; Haisjackl, C.; Pinggera, J.; Reichert, M.; Weber, B. Investigating expressiveness and understandability of hierarchy in declarative business process models. *Software & Systems Modeling* **2015**, *14*, 1081–1103.
9. Fahland, D.; Lubke, D.; Mendling, J.; Reijers, H.; Weber, B.; Weidlich, M.; Zugal, S. Declarative versus Imperative Process Modeling Languages: The Issue of Understandability. In *Enterprise, Business-Process and Information Systems Modeling*; Springer Berlin Heidelberg, 2009; Vol. 29, *Lecture Notes in Business Information Processing*, pp. 353–366.
10. Fahland, D.; Mendling, J.; Reijers, H.; Weber, B.; Weidlich, M.; Zugal, S. Declarative versus Imperative Process Modeling Languages: The Issue of Maintainability. In *Business Process Management Workshops*; Springer Berlin Heidelberg, 2010; Vol. 43, *Lecture Notes in Business Information Processing*, pp. 477–488.
11. Sadiq, S.W.; Orlowska, M.E.; Sadiq, W. Specification and validation of process constraints for flexible workflows. *Information Systems* **2005**, *30*, 349 – 378.
12. Rychkova, I.; Regev, G.; Wegmann, A. High-level design and analysis of business processes: the advantages of declarative specifications. RCIS; Pastor, O.; Flory, A.; Cavarero, J.L., Eds. IEEE, 2008, pp. 99–110.

13. Pesic, M.; van der Aalst, W.M.P. A Declarative Approach for Flexible Business Processes Management. Business Process Management Workshops; Eder, J.; Dustdar, S., Eds. Springer, 2006, Vol. 4103, *Lecture Notes in Computer Science*, pp. 169–180.

14. Rychkova, I.; Regev, G.; Wegmann, A. Using Declarative Specifications in Business Process Design. *IJCSA* **2008**, *5*, 45–68.

15. Parody, L.; Gómez-López, M.T.; M. Gasca, R. Hybrid business process modeling for the optimization of outcome data. *Information & Software Technology* **2016**, *70*, 140–154.

16. Teppan, E.C.; Friedrich, G. The Partner Units Configuration Problem. *CoRR* **2013**, *abs/1308.6206*.

17. GröNer, G.; BošKović, M.; Silva Parreiras, F.; GašEvić, D. Modeling and Validation of Business Process Families. *Inf. Syst.* **2013**, *38*, 709–726.

18. Petrie, C.J. *Automated Configuration Problem Solving*; Springer Publishing Company, Incorporated, 2012.

19. Gillmann, M.; Mindermann, R.; Weikum, G. Benchmarking and Configuration of Workflow Management Systems. In *Cooperative Information Systems*; Springer Berlin Heidelberg, 2000; Vol. 1901, pp. 186–197.

20. van der Aalst, W.M.P.; van Hee, K. *Workflow Management: Models, Methods, and Systems*; MIT Press: Cambridge, MA, USA, 2004.

21. Albert, P.; Henocque, L.; Kleiner, M. An End-to-End Configuration-Based Framework for Automatic SWS Composition. 20th IEEE International Conference on Tools with Artificial Intelligence, 2008, Vol. 1, pp. 351–358.

22. Bertoli, P.; Pistore, M.; Traverso, P. Automated composition of Web services via planning in asynchronous domains. *Artificial Intelligence* **2010**, *174*, 316 – 361.

23. Mesmoudi, A.; Mrissa, M.; Hacid, M.S. Combining configuration and query rewriting for Web service composition. IEEE International Conference on Web Services (ICWS), 2011, pp. 113–120.

24. Drescher, C. The Partner Units Problem a Constraint Programming Case Study. IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI 2012, Athens, Greece, November 7-9, 2012. IEEE Computer Society, 2012, pp. 170–177.

25. Mittal, S.; Frayman, F. Towards a Generic Model of Configuraton Tasks. Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 2; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 1989; IJCAI'89, pp. 1395–1401.

26. Rosa, M.L.; Dumas, M.; ter Hofstede, A.H.M.; Mendling, J. Configurable multi-perspective business process models. *Inf. Syst.* **2011**, *36*, 313–340.

27. van der Aalst, W.M.P.; Dumas, M.; Gottschalk, F.; ter Hofstede, A.H.M.; Rosa, M.L.; Mendling, J. Correctness-Preserving Configuration of Business Process Models. Fundamental Approaches to Software Engineering, 11th International Conference, FASE 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings; Fiadeiro, J.L.; Inverardi, P., Eds. Springer, 2008, Vol. 4961, *Lecture Notes in Computer Science*, pp. 46–61.

28. Rosemann, M.; van der Aalst, W.M.P. A configurable reference modelling language. *Inf. Syst.* **2007**, *32*, 1–23.

29. Vanderfeesten, I.; Reijers, H.A.; van der Aalst, W.M. Product-based workflow support. *Information Systems* **2011**, *36*, 517 – 535. Special Issue: Semantic Integration of Data, Multimedia, and Services.

30. Vanderfeesten, I.T.P.; Reijers, H.A.; van der Aalst, W.M.P. Product Based Workflow Support: Dynamic Workflow Execution. Advanced Information Systems Engineering, 20th International Conference, CAiSE 2008, Montpellier, France, June 16-20, 2008, Proceedings; Bellahsene, Z.; Léonard, M., Eds. Springer, 2008, Vol. 5074, *Lecture Notes in Computer Science*, pp. 571–574.

31. Gottschalk, F.; van der Aalst, W.M.P.; Jansen-Vullers, M.H.; Rosa, M.L. Configurable Workflow Models. *Int. J. Cooperative Inf. Syst.* **2008**, *17*, 177–221.

32. Parody, L.; Gómez-López, M.T.; Martínez Gasca, R. Data-Oriented Declarative Language for Optimizing Business Processes. 22nd International Conference on Information Systems Development (ISD2013), 2013, p. To appear.
33. Kluza, K.; Honkisz, K. From SBVR to BPMN and DMN Models. Proposal of Translation from Rules to Process and Decision Models. Artificial Intelligence and Soft Computing; Rutkowski, L.; Korytkowski, M.; Scherer, R.; Tadeusiewicz, R.; Zadeh, L.A.; Zurada, J.M., Eds. Springer International Publishing, 2016, pp. 453–462.
34. (OMG), O.M.G. Semantics of Business Vocabulary and Business Rules (SBVR). Version 1.4: Formal Specification., 2017.
35. Natschläger, C.; Kossak, F.; Schewe, K.D. Deontic BPMN: a powerful extension of BPMN with a trusted model transformation. *Software & Systems Modeling* **2015**, *14*, 765–793.
36. Wiśniewski, P.; Kluza, K.; Ligęza, A. An Approach to Participatory Business Process Modeling: BPMN Model Generation Using Constraint Programming and Graph Composition. *Applied Sciences* **2018**, *8*.
37. OMG. *Business Process Model and Notation (BPMN) Version 2.0*; Object Management Group Standard, 2011.
38. Rossi, F.; van Beek, P.; Walsh, T. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*; Elsevier Science Inc.: New York, NY, USA, 2006.
39. Dechter, R. *Constraint processing*; Elsevier Morgan Kaufmann, 2003; pp. I–XX, 1–481.
40. Manual, R. JSolver 2.1. Accedded on the 24th of February of 2014.
41. Gómez López, M.T.; M. Gasca, R.; Pérez-Álvarez, J.M. Decision-Making Support for the Correctness of Input Data at Runtime in Business Processes. *Int. J. Cooperative Inf. Syst.* **2014**, *23*.
42. Cheeseman, P.; Kanefsky, B.; Taylor, W.M. Where the Really Hard Problems Are. IJCAI; Mylopoulos, J.; Reiter, R., Eds. Morgan Kaufmann, 1991, pp. 331–340.