AI Approach to Formal Analysis of BPMN Models: Towards a Logical Model for BPMN Diagrams

Antoni Ligęza and Tomasz Potempa

Abstract. Modeling Business Processes has become a challenging issue of today's Knowledge Management. As such it is a core activity of Knowledge Engineering. There are two principal approaches to modeling such processes, namely Business Process Modeling and Notation (BPMN) and Business Rules (BR). Both these approaches are to certain degree complementary, but BPMN seems to become a standard supported by OMG. In this paper we investigate how to build a logical model of BPMN using logic, logic programming and rules. The main focus in on logical reconstruction of BPMN semantics which is necessary to define some formal requirements on model correctness enabling formal verification of such models.

1 Introduction

Knowledge has become a valuable and critical resource of contemporary organizations. In fact, possession of valid, most complete, up-to-date and essential knowledge has become a decisive factor of success in the so competitive market.

Unfortunately — or no — human possession and processing of knowledge turns out to be fairly inefficient for a number of reasons. Some most obvious ones include difficulties with knowledge sharing, storage, and efficient execution. Hence, *Knowledge Management* (KM) must be supported with tools

Antoni Ligęza
AGH University of Science and Technology,
al. A. Mickiewicza 30, 30-059 Krakow, Poland
e-mail: `ligeza@agh.edu.pl`

Tomasz Potempa
Higher School of Tarnów
ul. Mickiewicza 8, 33-100 Tarnów, Poland
e-mail: `tpotempa@gmail.com`

and techniques coming from Software Engineering and Knowledge Engineering universe.

Design, development and analysis of progressively more and more complex business processes require advanced methods and tools. Two generic modern approaches to modeling such processes have recently gained wider popularity. These are the *Business Process Modeling and Notation* [29, 32, 1], or BPMN for short, and *Business Rules* [28, 2, 7]. Although aimed at a common target, both of the approaches are rather mutually complementary and offer somewhat distinctive features enabling process modeling [9] and executing [20].

BPMN as such constitutes a set of graphical symbols, such as links modeling workflow, various splits and joins, events and boxes representing data processing activities. It forms a transparent visual tool for modeling complex processes promoted by OMG [25]. What is worth underlying is the expressive power of current BPMN. In fact it allows for modeling conditional operations, loops, event-triggered actions, splits and joins of data flow paths and communication processes [27]. Moreover, modeling can take into account several levels of abstraction which enables hierarchical approach.

An important issue about BPMN is that it covers three important aspects of any business process; these are:

- *data processing* or *data flow* specification; this includes input, output and internal data processing,
- *inference control* or *workflow control*; this includes diagrammatic specification of the process with partial ordering, switching and merging of flow,
- *structural representation* of the process as a whole; this allows for a visual representation at several levels of hierarchy.

BPMN as such can be considered as *procedural knowledge representation*; a BPMN diagram represents in fact a set of interconnected procedures. On the other hand, the workflow diagram, however, although it provides transparent, visual picture of the process, due to lack of formal model semantics makes attempts at more rigorous analysis problematic. Further, even relatively simple inference requires a lot of space for representation; there is no easy way to specify declarative knowledge, e.g. in the form of rules.

Business Rules (BR), also promoted by OMG [23, 24], offer an approach to specification of knowledge in a *declarative* manner. The way the rules are applied is left over to the user when it comes to rule execution. Hence, rules can be considered as *declarative knowledge specification*; inference control is normally not covered by basic rules.

Note that rules can fulfill different roles in the system [23]. Some three most important ones cover:

- *declarative knowledge specification* for inference of new facts,
- *integrity constraints* for preserving consistency of the knowledge base, and

- *meta-rules* i.e. rules defining how to use other rules; this may include partial *control knowledge specification* for improving efficiency of the inference process.

Some modern classifications cover the following types of rules:

- facts – rules defining true statement (no conditional part),
- definition rules – for defining terms and notions in use,
- integrity rules – rules defining integrity constraints,
- production rules – for derivation of new facts,
- reaction rules – rules triggered by events, reactive rules or ECA rules,
- transformation rules – rules defining possible transformations, term-rewriting rules; they may include numerical recipe rules,
- data processing rules – rules defining how particular data are to be transformed; these include numerical processing rules,
- control rules – in fact meta rules used for inference process control,
- meta rules – other rules defining how to use basic rules.

Rules, especially when grouped into decision modules (such as decision tables) are easier to analyze, however, the possibility of analysis depends on the accepted *knowledge representation language*, and in fact – the logic in use [14]. Formal models of rule-based systems and analysis issues are discussed in detail in [14].

Note that the two approaches are to certain degree complementary: Business Rules provide declarative specification of domain knowledge, which can be encoded into a BPMN model. On the other hand, a careful analysis of a BPMN diagram allows to extract certain rules governing the business. However, there is no consistent study on that possibility of mutual conversion. The main problems seem to consist in lack of formal specification of KR language. A some under-identification of BPMN.

The main common problem of BPMN is the lack of a *formal declarative model* defining precisely the semantics and logic behind the diagram. Hence defining and analyzing correctness of BPMN diagrams (e.g. in terms of termination or determinism) is a hard task. There are only few papers undertaking the issues of analysis and verification of BPMN diagrams [5, 26, 27]. However, the analysis is performed mostly at the *structural* level and does not take into account the semantics of dataflow and control knowledge.

In this paper we follow the ideas initially presented in [15] and extend the work described in [16]. An attempt at defining foundations for a more formal, logical, declarative model of the most crucial elements of BPMN diagrams is undertaken. We pass from logical analysis of BPMN components to their logical models, properties and representation in PROLOG. Some prototyped procedures for checking correct data flow are also presented. The model is aimed at enabling definition and further analysis of selected formal properties of a class of restricted BPMN diagrams. The analysis should take into account properties constituting reasonable criteria of correctness. The focus is on development of a formal, declarative model of BPMN components and its overall structure. In fact, a combination of recent approaches to development and verification of rule-based systems [21, 17, 19] seems to have potential influence on BPMN analysis.

2 A Basic Structural Model for BPMN

In this section a simplified structural model of BPMN diagrams is put forward. It constitutes a restricted abstraction of crucial intrinsic workflow components. As for events, only start and termination events are taken into account. Main knowledge processing units are activities (or tasks). Workflow control is modeled by subtypes of gateways: split and join operations. Finally, workflow sequence is modeled by directed links. No time/temporal aspect are considered. The following elements will be taken into consideration:

- $\mathbb{S}$ — a non-empty set of *start events* (possibly composed of a single element),
- $\mathbb{E}$ — a non-empty set of *end events* (possibly composed of a single element),
- $\mathbb{T}$ — a set of *activities* (or *tasks*); a task $T \in \mathbb{T}$ is a finite process with single input and single output, to be executed within a finite interval of time,
- $\mathbb{G}$ — a set of *split gateways* or *splits*, where branching of the workflow takes place; three disjoint subtypes of splits are considered:
  - $\mathbb{GX}$ — a set of *exclusive splits* where one and only one alternative paths can be followed (a split of EX-OR type),
  - $\mathbb{GP}$ — a set of *parallel splits* where all the paths of the workflow are to be followed (a split of *AND* type or a *fork*), and
  - $\mathbb{GO}$ — a set of *inclusive splits* where one or more paths should be followed (a split of *OR* type).
- $\mathbb{M}$ — a set of *merge gateways* or *joins* node of the diagram, where two or more paths meet; three further disjoint subtypes of merge (join) nodes are considered:
  - $\mathbb{MX}$ — a set of *exclusive merge* nodes where one and only one input path is taken into account (a merge of EX-OR type),
  - $\mathbb{MP}$ — a set of *parallel merge* nodes where all the paths are combined together (a merge of *AND* type), and
  - $\mathbb{MO}$ — a set of *inclusive merge* nodes where one or more paths influence the subsequent item (a merge of *OR* type).
- $\mathbb{F}$ — a set of workflow links, $\mathbb{F} \subseteq \mathbb{O} \times \mathbb{O}$, where $\mathbb{O} = \mathbb{S} \cup \mathbb{E} \cup \mathbb{T} \cup \mathbb{G} \cup \mathbb{M}$ is the join set of objects. All the component sets are pairwise disjoint.

The splits and joins depend on logical conditions assigned to particular branches. It is assumed that there is defined a partial function $\texttt{Cond}\colon \mathbb{F} \to \mathbb{C}$ assigning logical formulae to links. In particular, the function is defined for links belonging to $\mathbb{G} \times \mathbb{O} \cup \mathbb{O} \times \mathbb{M}$, i.e. outgoing links of split nodes and incoming links of merge nodes. The conditions are responsible for workflow control. For intuition, a simple BPMN diagram is presented in Fig. 1.

In order to ensure *correct structure* of BPMN diagrams a set of restrictions on the overall diagram structure is typically defined; they determine the so-called *well-formed diagram* [27].

Note however, that a well-formed diagram does not assure that for any input knowledge the process can be executed leading to a (unique) solution. This further depends on the particular input data, its transformation during processing, correct work of particular objects, and correct control defined by the branching/merging conditions assigned to links.

3 An Example of BPMN Diagram with Rules

In order to provide intuitions, the theoretical considerations will be illustrated with a simple example process. The process goal is to establish the so-called *set-point* temperature for a thermostat system [22]. The selection of the particular value depends on the season, whether it is a working day or not, and the time of the day.

Consider the following set of declarative rules specifying the process. There are eighteen inference rules (production rules):

Rule 1: $aDD \in \{monday, tuesday, wednesday, thursday, friday\} \longrightarrow aTD = wd.$
Rule 2: $aDD \in \{saturday, sunday\} \longrightarrow aTD = wk.$
Rule 3: $aTD = wd \wedge aTM \in (9, 17) \longrightarrow aOP = dbh.$
Rule 4: $aTD = wd \wedge aTM \in (0, 8) \longrightarrow aOP = ndbh.$
Rule 5: $aTD = wd \wedge aTM \in (18, 24) \longrightarrow aOP = ndbh.$
Rule 6: $aTD = wk \longrightarrow aOP = ndbh.$
Rule 7: $aMO \in \{january, february, december\} \longrightarrow aSE = sum.$
Rule 8: $aMO \in \{march, april, may\} \longrightarrow aSE = aut.$
Rule 9: $aMO \in \{june, july, august\} \longrightarrow aSE = win.$
Rule 10: $aMO \in \{september, october, november\} \longrightarrow aSE = spr.$
Rule 11: $aSE = spr \wedge aOP = dbh \longrightarrow aTHS = 20.$
Rule 12: $aSE = spr \wedge aOP = ndbh \longrightarrow aTHS = 15.$
Rule 13: $aSE = sum \wedge aOP = dbh \longrightarrow aTHS = 24.$
Rule 14: $aSE = sum \wedge aOP = ndbh \longrightarrow aTHS = 17.$
Rule 15: $aSE = aut \wedge aOP = dbh \longrightarrow aTHS = 20.$
Rule 16: $aSE = aut \wedge aOP = ndbh \longrightarrow aTHS = 16.$
Rule 17: $aSE = win \wedge aOP = dbh \longrightarrow aTHS = 18.$
Rule 18: $aSE = win \wedge aOP = ndbh \longrightarrow aTHS = 14.$

Let us briefly explain these rules. The first two rules define if we have today (aTD) a workday (wd) or a weekend day (wk). Rules 3-6 define if the operation hours (aOP) are during business hours (dbh) or not during business hours ($ndbh$); they take into account the workday/weekend condition and the current time (hour). Rules 7-10 define the season (aSE) is summer (sum), autumn (aut), winter (win) or spring (spr). Finally, rules 11-18 define the

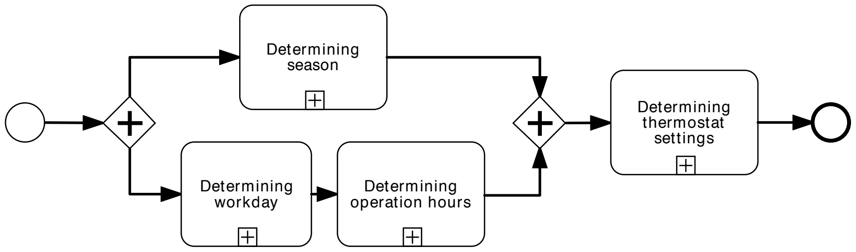

Fig. 1 An example BPMN diagram — top-level specification of the thermostat system

precise setting of the thermostat ($aTHS$). Observe that the set of rules is flat; basically no control knowledge is provided.

Now, let us attempt to visualize a business process defined with these rules. A BPMN diagram of the process is specified in Fig. 1.

After start, the process is split into two independent paths of activities. The upper path is aimed at determining the current season[1] (aSE; it can take one of the values $\{sum, aut, win, spr\}$; the detailed specification is provided with rules 7-10 below). A more visual specification of this activity with an appropriate set of rules is shown in Fig. 2.

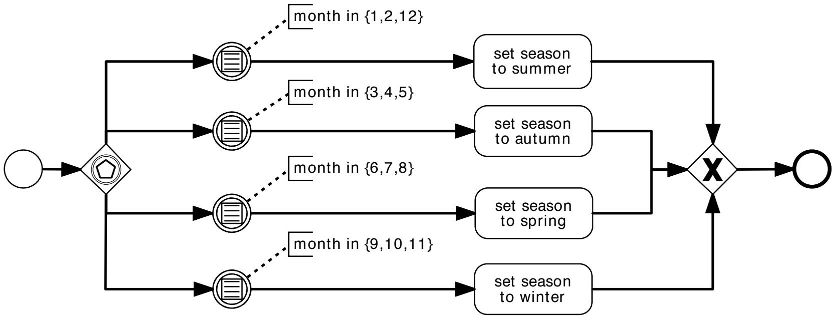

Fig. 2 An example BPMN diagram — detailed specification a BPMN task

The lower path activities determine whether the day (aDD) is a workday ($aTD = wd$) or a weekend day ($aTD = wk$), both specifying the value of today (aTD; specification provided with rules 1 and 2), and then, taking into account the current time (aTM), whether the operation (aOP) is during business hours ($aOP = dbh$) or not ($aOP = ndbh$); the specification is provided with rules 3-6. This is illustrated with Fig. 3 and Fig. 4.

[1] For technical reasons all attribute names used in this example start with lowercase 'a'.

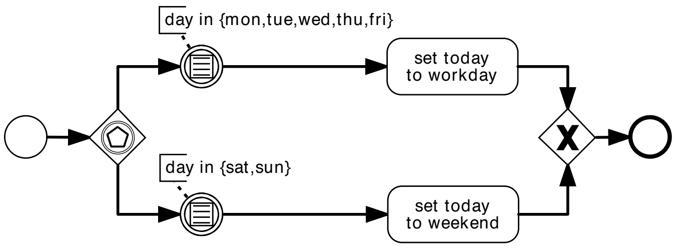

Fig. 3 An example BPMN diagram — detailed specification of determining the day task

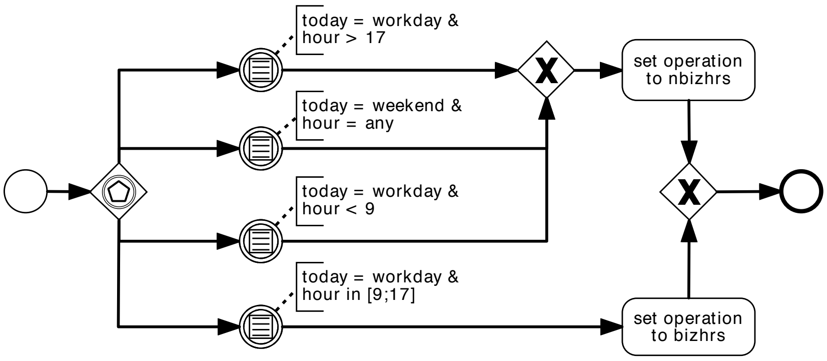
today = workday & hour > 17
set operation to nbizhrs
today = weekend & hour = any
today = workday & hour < 9
today = workday & hour in [9;17]
set operation to bizhrs

Fig. 4 An example BPMN diagram — detailed specification of working hours task

Finally, the results are merged together and the final activity consists in determining the thermostat settings ($aTHS$) for particular season (aSE) and time (aTM) (the specification is provided with rules 11-18). This is illustrated with Fig. 5.

Even in this simple example, answers to the following important questions are not obvious:

1. *data flow correctness*: is any of the four tasks/activities specified in a correct way? Will each task end with producing the desired output for any admissible input data?
2. *split consistency*: will the workflow possibly explore all the paths after a split? Will it always explore at least one?
3. *merge consistency*: will it be always possible to merge knowledge coming from different sources at the merge node?
4. *termination/completeness*: does the specification assure that the system will always terminate producing some temperature specification for *any* admissible input data?
5. *determinism*: will the output setting be determined in a unique way?

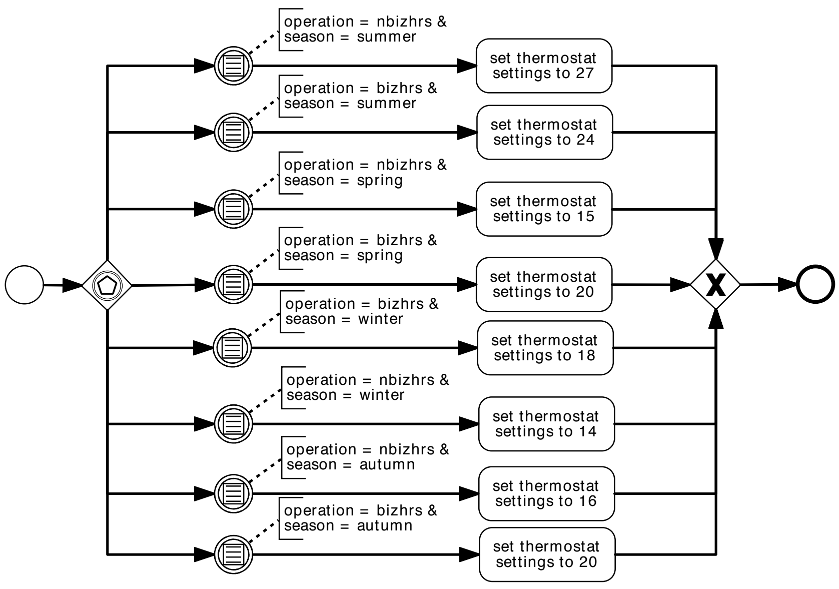

Fig. 5 An example BPMN diagram — detailed specification of the final thermostat setting task

Note that we do not ask about *correctness* of the result; in fact, the rules, embedded into a BPMN diagram, provide a kind of *executable specification*, so there is no reference point to claim that the final output is correct or not.

4 Logical Analysis of BPMN Diagrams

A BPMN diagram can model quite complex processes. Apart from *external consistency* validation (i.e. whether or not the diagram models correctly the external system in a *complete* way, does not introduce any non-existent features, and there is an *isomorphism* between those two), an important issue is the *internal consistency* requirement for *correct structure* of the diagram and *correct workflow specification.* The first one refers to the static specification of components and their connections. The second one consists in correct work of the structure for all admissible input data specification.

The structural correctness is defined by requirements for *well-formed BPMN diagram.* However, even having correct structure, the process can go wrong due to unserved data or wrong workflow control, for example. Below, an attempt is made at specification of some minimal requirements for (i) correct work of process components (tasks), (ii) assuring data flow, (iii) correct work of splits, (iv) correct work of merge nodes, and finally — (v) termination of the overall process.

A more complex issue consists in verification of internal workflow correctness. The analysis must take into account, at least the following aspects:

1. Local correctness requirements:
   a. Correct specification and work of process components performing activities,
   b. Correct specification of data flow,
   c. Correct specification and work of splits,
   d. Correct specification and work of joins.
2. Global correctness requirements:
   a. No deadlocks — the system must work for all admissible input data,
   b. Termination — the system must terminate work within a finite period of time,
   c. Determinism — the results should be repeatable for repeated input data (ambiguous rules, hazards, races).

Further requirements may refer to features such as *minimal representation*, *optimal decomposition*, *robustness*, and *optimal results* and *optimal execution.* In order to answer these questions one must (i) assure the correct work of all the components (activities/processes), (ii) assure the correctness of data flow between components, (iii) assure correct inference control (w.r.t. split and join operations), (iv) check if the static structure of the diagram is correct and, finally (v) check if all this combined together will work, i.e. the inference process is not blocked at some point (e.g. due to a deadlock).

4.1 Component Correctness

In this section we put forward some minimal requirements defining correct work of rule-based process components performing BPMN activities. Each such component is composed of a set of inference rules, designed to work within a the same context; in fact, preconditions of the rules incorporate the same attributes. In our example we have four such components: determining workday (rules 1-2), determining operation hours (rules 3-6), determining season (rules 7-10) and determining the thermostat setting (rules 11-18).

In general, the outermost logical model of a component T performing some activity/task can be defined as a triple of the form:

$$T = (\psi_T, \varphi_T, \mathcal{A}), \tag{1}$$

where ψ_T is a formula defining the restrictions on the component input, φ_T defines the restrictions for component output, and $\mathcal{A}$ is an algorithm which for a given input satisfying ψ_T produces an (desirably uniquely defined) output, satisfying φ_T. For intuition, ψ_T and φ_T define a kind of a 'logical tube' — for every input data satisfying ψ_T (located at the entry of the tube), the component will produce and output satisfying φ_T (still located within the tube at its output). The precise recipe for data processing is given by an algorithm $\mathcal{A}$.

The specification of a rule-based process component given by (1) is considered *correct*, if and only if for any input data satisfying ψ_T the algorithm $\mathcal{A}$ produces an output satisfying φ_T. It is further *deterministic* (unambiguous) if the generated output is unique for any admissible input.

For example, consider the component determining operation hours. Its input restriction formula ψ_T is the disjunction of precondition formulae $\psi_3 \vee \psi_4 \vee \psi_5 \vee \psi_6$, where ψ_i is a precondition formula for rule i. We have $\psi_T = ((aTD = wd) \wedge (aTM \in [0,8] \vee aTM \in [9,17] \vee aTM \in [18,24])) \vee (aTD = wk)$. The output restriction formula is given by $\varphi_T = (aOP = dbh) \vee (aOP = ndbh)$. The algorithm is specified directly by the rules; rules are in fact a kind of *executable specification*.

In order to be sure that the produced output is unique, the following *mutual exclusion* condition should hold:

$$\not\models \psi_i \wedge \psi_j \tag{2}$$

for any $i \neq j$, $i, j \in \{1, 2, \ldots, k\}$. A simple analysis shows that the four rules have mutually exclusive preconditions, and the joint precondition formula ψ_T covers any admissible combination of input parameters; in fact, the subset of rules is locally *complete* and *deterministic* [14].

4.2 Correct Flow of Data

In our example we consider only rule-based components. Let ϕ define the context of operation, i.e. a formula defining some restrictions over the current state of the knowledge-base that must be satisfied before the rules of a component are explored. For example, ϕ may be given by $\varphi_{T'}$ of a component T' directly preceding the current one. Further, let there be k rules in the current component, and let ψ_i denote the joint precondition formula (a conjunction of atoms) of rule i, $i = 1, 2, \ldots, k$. In order to be sure that at least one of the rules will be fired, the following condition must hold:

$$\phi \models \psi_T, \tag{3}$$

where $\psi_T = \psi_1 \vee \psi_2 \vee \ldots \vee \psi_k$ is the disjunction of all precondition formulae of the component rules. The above restriction will be called the *funnel principle*. For intuition, if the current knowledge specification satisfies restriction defined by ϕ, then at least one of the formula preconditions must be satisfied as well.

For example, consider the connection between the component determining workday and the following it component determining operation hours. After leaving the former one, we have that $aTD = wd \vee aTD = wk$. Assuming that the time can always be read as an input value, we have $\phi = (aTD = wd \vee aTD = wk) \wedge aTM \in [0, 24]$. On the other hand, the disjunction of precondition formulae $\psi_3 \vee \psi_4 \vee \psi_5 \vee \psi_6$ is given by $\psi_T = (aTD = wd) \wedge (aTM \in [0, 8] \vee aTM \in [9, 17] \vee aTM \in [18, 24])) \vee aTD = wk$. Obviously, the funnel condition given by (3) holds.

4.3 Correct Splits

An exclusive split $GX(q_1, q_2, \ldots q_k) \in \mathbb{GX}$ with k outgoing links is modeled by a fork structure assigned excluding alternative of the form:

$$q_1 \veebar q_2 \veebar \ldots \veebar q_k,$$

where $q_i \wedge q_j$ is always false for $i \neq j$. An exclusive split can be considered correct if and only if at least one of the alternative conditions is satisfied. We have the following logical requirement:

$$\models q_1 \vee q_2 \vee \ldots \vee q_k, \tag{4}$$

i.e. the disjunction is in fact a tautology. In practice, to assure (4), a predefined exclusive set of conditions is completed with a default q_0 condition defined as $q_0 = \neg q_1 \wedge \neg q_2 \wedge \ldots \wedge \neg q_k$; obviously, the formula $q_0 \vee q_1 \vee q_2 \vee \ldots \vee q_k$ is a tautology.

Note that in case when an input restriction formula ϕ is specified, the above requirement given by (4) can be relaxed to:

$$\phi \models q_1 \vee q_2 \vee \ldots \vee q_k. \tag{5}$$

An inclusive split $GO(q_1, q_2, \ldots q_k) \in \mathbb{GO}$ is modeled as disjunction of the form:

$$q_1 \vee q_2 \vee \ldots \vee q_k,$$

An inclusive split to be considered correct must also satisfy formula (4), or at least (5). As before, this can be achieved through completing it with the q_0 default formula.

A parallel split $GP(q_1, q_2, \ldots q_k) \in \mathbb{GP}$ is referring to a fork-like structure, where all the outgoing links should be followed in any case. For simplicity, a parallel split can be considered as an inclusive one, where all the conditions assigned to outgoing links are set to *true*.

Note that, if ϕ is the restriction formula valid for data at the input of the split, then any of the output restriction formula is defined as $\phi \wedge q_i$ for any of the outgoing link i, $i = 1, 2, \ldots, k$.

4.4 Correct Merge

Consider a workflow merge node, where k knowledge inputs satisfying restrictions $\phi_1, \phi_2, \ldots, \phi_k$ respectively meet together, while the selection of particular input is conditioned by formulae $p_1, p_2, \ldots, p_k$, respectively.

An exclusive merge $MX(p_1, p_2, \ldots, p_k) \in \mathbb{MX}$ of k inputs is considered correct if and only if the conditions are pairwise disjoint, i.e.

$$\not\models p_i \wedge p_j \tag{6}$$

for any $i \neq j$, $i, j \in \{1, 2, \ldots, k\}$. Moreover, to assure that the merge works, at least one of the conditions should hold:

$$\models p_1 \vee p_2 \vee \ldots \vee p_k, \tag{7}$$

i.e. the disjunction is in fact a tautology. If the input restrictions $\phi_1, \phi_2, \ldots, \phi_k$ are known, condition (7) might possibly be replaced by $\models (p_1 \wedge \phi_1) \vee (p_2 \wedge \phi_2) \vee \ldots \vee (p_k \wedge \phi_k)$.

Note that in case a join input restriction formula ϕ is specified, the above requirement can be relaxed to:

$$\phi \models p_1 \vee p_2 \vee \ldots \vee p_k, \tag{8}$$

and if the input restrictions $\phi_1, \phi_2, \ldots, \phi_k$ are known, it should be replaced by $\phi \models (p_1 \wedge \phi_1) \vee (p_2 \wedge \phi_2) \vee \ldots \vee (p_k \wedge \phi_k)$.

An inclusive merge $MO(p_1, p_2, \ldots, p_k) \in \mathbb{MO}$ of k inputs is considered correct if one is assured that the merge works — condition (7) or (8) hold.

A parallel merge $MP \in \mathbb{MP}$ of k inputs is considered correct by default. However, if the input restrictions $\phi_1, \phi_2, \ldots, \phi_k$ are known, a consistency requirement for the combined out takes the form that ϕ must be consistent (satisfiable), where:

$$\phi = \phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_k \tag{9}$$

An analogous requirement can be put forward for the active links of an inclusive merge.

$$\models p_1 \wedge p_2 \wedge \ldots \wedge p_k, \tag{10}$$

i.e. the conjunction is in fact a tautology, or at least

$$\phi \models p_1 \wedge p_2 \wedge \ldots \wedge p_k. \tag{11}$$

In general, parallel merge can be made correct in a trivial way by putting $p_1 = p_2 = \ldots = p_k = true$.

Note that even correct merge leading to a satisfiable formula assure only passing the merge node; the funnel principle must further be satisfied with respect to the following-in-line object. To illustrate that consider the input

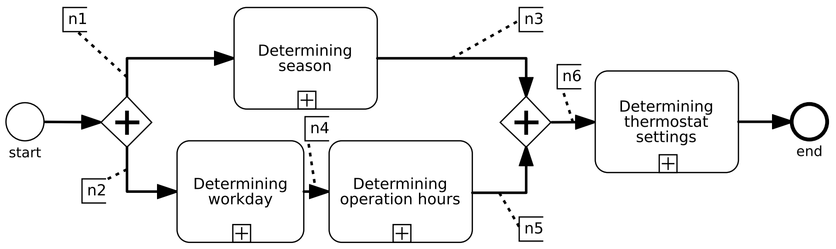

Fig. 6 An example BPMN diagram — top-level specification of thermostat system with the additional Prolog annotations

of the component determining thermostat setting. This is the case of parallel merge of two inputs. The joint formula defining the restrictions on combined output of the components for determining season and determining operation hours is of the form:

$$\begin{aligned}\phi = (aSE = sum \lor aSE = aut \lor aSE = win \lor \\ aSE = spr) \land (aOP = dbh \lor aOP = ndbh).\end{aligned}$$

A simple check of all possible combinations of season and operation hours shows that all the eight possibilities are covered by preconditions of rules 11-18; hence the funnel condition (3) holds.

5 BPMN Model in Prolog

In this section a Prolog model for the example BPMN diagram is presented. It enables logical analysis of the diagram. Below, samples of Prolog code are listed.

A BPMN diagram is represented as a complex graph with different types of nodes. Each specific component of BPMN is mapped into a Prolog fact. In order to model the structure of a BPMN diagram in a declarative way it is proposed to use a set of Prolog facts. A generic form can be as follows:

```
component_type(<id>,
   <input_node>,<input-formula>,
   <output_node>,<output-formula>).
```

In practice, such facts can be of different structure for different components.

The Thermostat example presented in Fig. 6 is defined with the Prolog code in Listing 1.

Algorithm 1. BPMN example coding in Prolog

```
%%%% BPMN Knowledge Base %%%%
%%% Start & end nodes declaration
init_node(start).
end_node(end).
nodes([n1,n2,n3,n4,n5,n6]).
%%% Tasks
% task(<id>,<input_node>,
%       <input_formula>,
%       <output_node>,
%       <output_formula>).
task(ds,n1
      [[mspr],[msum],[maut],[mwin]],
      n3,[[spr],[sum],[aut],[win]]).
task(dw,n2,[[d15],[d67]],n4,
      [[twr],[twe]]).
task(do,n4,[[twr,t18],[twr,t8],
      [twr,t917],[twe]],n5,
      [[nonbiz],[biz]]).
task(dt,n6,[[nonbiz,spr],[nonbiz,sum],
      [nonbiz,aut],[nonbiz,win],
      [biz,spr],[biz,sum],[biz,aut],
      [biz,win]],end,
      [[t14],[t15],[t16],[t18],[t20],
      [t24],[t27]]).
% split2(<id>,<split_type>,
%         <input_node>,
%         (<output_node>,
%          <logical_condition>),
%         (<output_node>,
%         <logical_condition>)).
split2(s1,and,start,(n1,true),
       (n2,true)).
% merge2(<id>,<merge_type>,
%         <input_node>,
%         <input_node>,
%         <output_node>).
merge2(m1,and,n3,n5,n6).
```

There are four tasks defined:

ds *define season* with input node $n1$, output node $n3$, input formula being a DNF - disjunction of all four precondition formulas of the four rules, and output formula being DNF - a disjunction of the conclusions of the rules, dw*define workday* with input node $n2$, output node $n4$, input formula being a DNF - disjunction of all two precondition formulas of the two rules, and output formula being DNF - a disjunction of the conclusions of the rules,

do *define operation* with input node $n4$, output node $n5$, input formula being a DNF - disjunction of all four precondition formulas of the four rules, and output formula being DNF - a disjunction of the conclusions of the rules,

dt *define temperature* with input node $n1$, output node $n3$, input formula being a DNF - disjunction of all eight precondition formulas of the eight rules, and output formula being DNF - a disjunction of the conclusions of the rules.

There are also one split node, from start to $n1$ and $n2$, and one merge node, from $n3$ and $n5$ to $n6$.

As an example of analysis let us present a code for verification of data flow (the funnel condition) between tasks *dw* and *do*; the intuition behind is that any output generated by *dw* should be accepted and further processed by *do*. The sample code is presented in Listing 2.

After calling the funnel check for the internal node $n4$ the system proves that the output formula of task *dw* implies the input formula of task *do*. The output of the program, confirming the result of the check is given below.

Other static checks for data flow have been implemented and tested as well.

6 Related Works

To the best of our knowledge, there are no related works which define BPMN model in the Prolog language. The only one approach that uses Prolog by Andročec [4] significantly differs from our work. Andročec used Prolog for specification of business processes, however, he used it to assess the cost and time of running a process and to identify potential problems with resources. Thus, he defined the model for simulation purposes.

Similar attempts to formalization of BPMN models were carried out by Lam [12], Andersson et al. [3], Wong and Gibbons [33] as well as Dijkman and Van Gorp [6]. Other attempts to formalization of process models, however not concerning BPMN, were carried out by Gruhn and Laue [8].

The BPMN model defined by Lam in [12] was used to check the diagrams against the properties specified by a user for a particular model [13]. In our case, the properties, which are correctness requirements, can be used for any BPMN model.

In the case of declarative model presented in [3], they introduced the notion of activity dependency model, which identifies, classifies, and relates activities needed for executing and coordinating value transfers. In particular, in their model, relations between activities can be specified in terms of notions like resource flow, trust, coordination, and reciprocity. The four types of dependencies which can be identified (flow, trust, trigger, and duality dependencies) are

Algorithm 2. BPMN example coding in Prolog: funnel between tasks

```
%%% Funnel condition checking for
%%% nodes among tasks
funnel(N) :-
    task(IDOUT,_,_,N,FOUT),
    task(IDIN,N,FIN,_,_),
    implies(FOUT,FIN),
    write(IDOUT),write('—>'),write(N),

    write('—>'),write(IDIN),nl.

%%% Definition of implication
%%% for two DNF
implies(true,_) :- !.
implies([],_) :- !.
implies([MIN|T],DNF) :-
    imply(MIN,DNF),
    implies(T,DNF).

%%% Definition of implication
%%% DNF |= MIN
imply([],_) :- !.
imply(MIN,DNF) :-
    member(M,DNF),  subset(M,MIN),  !.
imply(MIN,DNF) :-
    find_subsets(DNF,SDNF),
    reduce(SDNF,RSDNF),
    member(M,RSDNF),
    subset(M,MIN).

find_subsets([],[]) :- !.
find_subsets([G|T],[G|T2]):-
     find_subsets(T,T2).
find_subsets([_|T],T2):-
     find_subsets(T,T2).
```

Algorithm 3. BPMN example coding in Prolog: funnel between tasks

```
?- funnel(n4).
dw—>n4—>do
true
```

not a part of the BPMN style, thus their model is not completely BPMN compliant and has another goals then ours.

Dijkman and Van Gorp [6] formalized the BPMN model using graph rewrite rules. However, their goal was also different from ours. They focused on execution semantics. Thus, their approach is suitable for simulation, animation and execution of BPMN models.

Wong and Gibbons [33] defined the semantics of BPMN models using Communicating Sequential Processes, in which a process is a pattern of behavior. They provide only verification of such issues as: hierarchical refinement, partial refinement and hierarchical independence.

7 Concluding Remarks

Our work is a part of an approach in the area of of business processes and rules integration [10]. It constitutes an attempt at providing a logical, declarative model for well-defined BPMN diagram [15]. The model is aimed at defining formal semantics of diagram components and the workflow operation. The main focus is on the specification of correct components and correct dataflow. Global termination conditions are specified in a recursive way. Summarizing, in the paper we:

- formulated a formal model for a subset of BPMN,
- defined local and global correctness requirements,
- specified the detailed logic that stems from the diagram.

The original contribution of our work consists in:

- presenting open issues corresponding to the BPMN diagrams, such as consistency of elements,
- specifying a BPMN model in the declarative Prolog language, which helps to check the defined correctness requirements.

Note that the logical analysis can be performed *off-line*, on the base of logical requirements ϕ, ψ and φ of data. However, if such specifications are data-dependent (e.g. in case of loops or more complex non-monotonic data processing) the analysis may be possible only in *on-line* form, separately for any admissible input data.

As future work, a more complex modeling, verification and execution approach is considered. In the case of modeling issue, we plan to implement this approach by extending one of the existing BPMN tools in order to integrate it with the HeKatE Qt Editor (HQEd) for XTT2-based Business Rules [9]. The XTT2 rules [21] (and tables) can be formally analyzed using the so-called verification HalVA framework [10] or using Petri net approach [31]. Although table-level verification can be performed with HalVA [18], the global verification is a more complex issue [11]. Our preliminary works on global verification have been presented in [30].

Acknowledgment. The paper is supported by the *BIMLOQ* Project funded from 2010–2012 resources for science as a research project.

References

1. Allweyer, T.: BPMN 2.0. Introduction to the Standard for Business Process Modeling. BoD, Norderstedt (2010)
2. Ambler, S.W.: Business Rules (2003), `http://www.agilemodeling.com/artifacts/businessRule.htm`
3. Andersson, B., Bergholtz, M., Edirisuriya, A., Ilayperuma, T., Johannesson, P.: A declarative foundation of process models. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 233–247. Springer, Heidelberg (2005)
4. Andročec, D.: Simulating BPMN models with Prolog. In: Proceedings from the Central European Conference on Information and Intelligent Systems, CECIIS 2010, pp. 363–368 (2010)
5. Dijkman, R.M., Dumas, M., Ouyang, C.: Formal semantics and automated analysis of BPMN process models. preprint 7115. Tech. rep., Queensland University of Technology, Brisbane, Australia (2007)
6. Dijkman, R., Van Gorp, P.: Bpmn 2.0 execution semantics formalized as graph rewrite rules. In: Mendling, J., Weidlich, M., Weske, M. (eds.) BPMN 2010. LNBIP, vol. 67, pp. 16–30. Springer, Heidelberg (2010)
7. Giurca, A., Gašević, D., Taveter, K. (eds.): Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches. Information Science Reference, Hershey (2009)
8. Gruhn, V., Laue, R.: Checking properties of business process models with logic programming. In: Augusto, J.C., Barjis, J., Ultes-Nitsche, U. (eds.) Proceedings of the 5th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems, MSVVEIS 2007, In conjunction with ICEIS 2007, Funchal, Madeira, Portugal, pp. 84–93. Insticc Press (June 2007)
9. Kluza, K., Kaczor, K., Nalepa, G.J.: Enriching business processes with rules using the Oryx BPMN editor. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2012, Part II. LNCS, vol. 7268, pp. 573–581. Springer, Heidelberg (2012), `http://www.springerlink.com/content/u654r0m56882np77/`
10. Kluza, K., Maślanka, T., Nalepa, G.J., Ligęza, A.: Proposal of representing BPMN diagrams with XTT2-based business rules. In: Brazier, F.M.T., Nieuwenhuis, K., Pavlin, G., Warnier, M., Badica, C. (eds.) Intelligent Distributed Computing V. SCI, vol. 382, pp. 243–248. Springer, Heidelberg (2011), `http://www.springerlink.com/content/d44n334p05772263/`
11. Kluza, K., Nalepa, G.J., Szpyrka, M., Ligęza, A.: Proposal of a hierarchical approach to formal verification of BPMN models using Alvis and XTT2 methods. In: Canadas, J., Nalepa, G.J., Baumeister, J. (eds.) 7th Workshop on Knowledge Engineering and Software Engineering (KESE2011) at the Conference of the Spanish Association for Artificial Intelligence (CAEPIA 2011), La Laguna, Tenerife, Spain, pp. 15–23 (November 10, 2011), `http://ceur-ws.org/Vol-805/`
12. Lam, V.S.W.: Equivalences of BPMN processes. Service Oriented Computing and Applications 3(3), 189–204 (2009)

13. Lam, V.S.W.: Formal analysis of BPMN models: a NuSMV-based approach. International Journal of Software Engineering and Knowledge Engineering 20(7), 987–1023 (2010)
14. Ligęza, A.: Logical Foundations for Rule-Based Systems. SCI, vol. 11. Springer, Heidelberg (2006)
15. Ligęza, A.: BPMN – a logical model and property analysis. Decision Making in Manufacturing and Services 5(1-2), 57–67 (2011)
16. Ligęza, A., Kluza, K., Potempa, T.: Ai approach to formal analysis of bpmn models. towards a logical model for bpmn diagrams. In: Ganzha, M., Maciaszek, L.A., Paprzycki, M. (eds.) Proceedings of the Federated Conference on Computer Science and Information Systems, FedCSIS 2012, Wroclaw, Poland, September 9-12, pp. 931–934 (2012), `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6354394`
17. Ligęza, A., Nalepa, G.J.: A study of methodological issues in design and development of rule-based systems: proposal of a new approach. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 1(2), 117–137 (2011), doi:10.1002/widm.11
18. Nalepa, G.J., Bobek, S., Ligęza, A., Kaczor, K.: HalVA – rule analysis framework for XTT2 rules. In: Bassiliades, N., Governatori, G., Paschke, A. (eds.) RuleML 2011 - Europe. LNCS, vol. 6826, pp. 337–344. Springer, Heidelberg (2011), `http://www.springerlink.com/content/c276374nh9682jm6/`
19. Nalepa, G.J.: Semantic Knowledge Engineering. A Rule-Based Approach. Wydawnictwa AGH, Kraków (2011)
20. Nalepa, G.J., Kluza, K., Kaczor, K.: Proposal of an inference engine architecture for business rules and processes. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2013, Part II. LNCS (LNAI), vol. 7895, pp. 453–464. Springer, Heidelberg (2013), `http://www.springer.com/computer/ai/book/978-3-642-38609-1`
21. Nalepa, G.J., Ligęza, A.: HeKatE methodology, hybrid engineering of intelligent systems. International Journal of Applied Mathematics and Computer Science 20(1), 35–53 (2010)
22. Negnevitsky, M.: Artificial Intelligence. A Guide to Intelligent Systems. Addison-Wesley, Harlow (2002) ISBN 0-201-71159-1
23. OMG: Production Rule Representation RFP. Tech. rep., Object Management Group (2003)
24. OMG: Semantics of Business Vocabulary and Business Rules (SBVR). Tech. Rep. dtc/06-03-02, Object Management Group (2006)
25. OMG: Business Process Model and Notation (BPMN): Version 2.0 specification. Tech. Rep. formal/2011-01-03, Object Management Group (2011)
26. Ouyang, C., Wil, M.P., van der Aalst, M.D., ter Hofstede, A.H.: Translating BPMN to BPEL. Tech. rep., Faculty of Information Technology, Queensland University of Technology, GPO Box 2434, Brisbane QLD 4001, Australia Department of Technology Management, Eindhoven University of Technolog y, GPO Box 513, NL-5600 MB, The Netherlands (2006)
27. Ouyang, C., Dumas, M., ter Hofstede, A.H., van der Aalst, W.M.: From bpmn process models to bpel web services. In: IEEE International Conference on Web Services, ICWS 2006 (2006)
28. Ross, R.G.: The RuleSpeak Business Rule Notation. Business Rules Journal 7(4) (2006), `http://www.BRCommunity.com/a2006/b282.html`
29. Silver, B.: BPMN Method and Style. Cody-Cassidy Press (2009)

30. Szpyrka, M., Nalepa, G.J., Ligęza, A., Kluza, K.: Proposal of formal verification of selected BPMN models with Alvis modeling language. In: Brazier, F.M.T., Nieuwenhuis, K., Pavlin, G., Warnier, M., Badica, C. (eds.) Intelligent Distributed Computing V. SCI, vol. 382, pp. 249–255. Springer, Heidelberg (2011), http://www.springerlink.com/content/m181144037q67271/
31. Szpyrka, M., Szmuc, T.: Decision tables in petri net models. In: Kryszkiewicz, M., Peters, J.F., Rybiński, H., Skowron, A. (eds.) RSEISP 2007. LNCS (LNAI), vol. 4585, pp. 648–657. Springer, Heidelberg (2007)
32. White, S.A., Miers, D.: BPMN Modeling and Reference Guide: Understanding and Using BPMN. Future Strategies Inc., Lighthouse Point (2008)
33. Wong, P.Y.H., Gibbons, J.: A process semantics for bpmn. In: Liu, S., Araki, K. (eds.) ICFEM 2008. LNCS, vol. 5256, pp. 355–374. Springer, Heidelberg (2008)