HAL open science

Artificial Intelligence for Knowledge Management with BPMN and Rules

Antoni Ligęza, Tomasz Potempa

To cite this version:

Antoni Ligęza, Tomasz Potempa. Artificial Intelligence for Knowledge Management with BPMN and Rules. 1st IFIP International Workshop on Artificial Intelligence for Knowledge Management (AI4KM), Aug 2012, Montpellier, France. pp.19-37, 10.1007/978-3-642-54897-0_2 . hal-01256584

HAL Id: hal-01256584

https://hal.science/hal-01256584v1

Submitted on 15 Jan 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Artificial Intelligence for Knowledge Management with BPMN and Rules

Antoni Ligęza and Tomasz Potempa

[1] AGH University of Science and Technology, Krakow, Poland,
email: ligeza@agh.edu.pl
[2] Higher School of Tarnów, Tarnow, Poland,
email: t_potempa@pwsztar.edu.pl

Abstract This paper presents a framework combining BPMN and BR as a tool for Knowledge Management (KM). An attempt at providing a common model supported with Artificial Intelligence (AI) techniques and tools is put forward. Through an extended example it is shown how to combine BPMN and BR and how to pass to semantic level enabling building executable specifications and knowledge analysis. Some of the problems concerning these two approaches can be to certain degree overcome thanks to their complementary nature. We only deal with a restricted view of Knowledge Management, where knowledge can be modeled explicitly in a formal representation, and it does not take into account the knowledge residing in people's heads.

1 INTRODUCTION

Design, development and analysis of progressively more and more complex business processes require advanced methods and tools. Apart from variety of classical Artificial Intelligence (AI) stuff, two generic modern approaches to modeling such processes have recently gained wider popularity; these are: *Business Process Model and Notation* (BPMN) [1] and *Business Rules* (BR) [2,3]. Although aimed at a common target, both of these approaches are rather mutually complementary and offer distinctive features enabling process modeling.

BPMN constitutes a set of graphical symbols, such as links modeling workflow, various splits and joins, events and boxes representing data processing activities. It is a transparent visual tool for modeling complex processes promoted by OMG [1]. 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 among processes. Moreover, modeling can take into account several levels of abstraction enabling a hierarchical approach.

BPMN can be considered as *procedural knowledge representation*; a BPMN diagram represents in fact a set of interconnected procedures. Although BPMN provides transparent, visual representation of the process, due to lack of formal model semantics it 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, also promoted by OMG [4,5], 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, however, is not covered by basic rules. Hence, the same set of rules can be used in numerous ways, and it may become problematic to *find* a solution even having at hand all the necessary rules.

Note that these two approaches are to certain degree complementary: BR provide declarative specification of domain knowledge, which can be encoded into a BPMN model. On the other hand, BPMN diagram can be used as procedural specification of the workflow, including inference control [6]. However, BPMN lacks of a *formal declarative model* defining the semantics and logic behind the diagram. Hence, defining and analyzing correctness of BPMN diagrams is a hard task. There are papers undertaking the issues of analysis and verification of BPMN diagrams [7,8,9,10]. 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 position paper, we follow the ideas initially presented in [11]. An attempt at defining foundations for a more formal, logical, declarative model of the most crucial elements of BPMN diagrams combined with BR is undertaken. We pass from logical analysis of BPMN component to their logical models, properties and representation in PROLOG [12]. 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 the recent approaches to development and verification of rule-based systems [13,14,15] seems to have potential influence on the BPMN analysis.

2 MOTIVATION

Knowledge has become a valuable resource and a decisive factor for successful operation of organizations, companies and societies. As vast amounts of knowledge are in use, tools supporting Knowledge Management (KM) are inevitable support for Decision Makers. Such tools can be roughly classified into the following categories:

- *Conceptual Models* — various symbolic and visual ways of Knowledge Representation (KR), analysis, and supporting design of knowledge-intensive systems and applications; as an example one can mention various schemes, graphs, networks and diagrams, with UML [16] and BPMN [17] being some perfect examples,
- *Logical Models* — more formal KR and knowledge processing (reasoning, inference) tools, supporting both *representation* and *application* of knowledge [18,19]. It is important that such models typically support also *semantic* issues; as an example one can mention various types of logics and logic-derived formalisms including rules and Business Rules (BR) as some perfect examples.
- *Functional and Procedural Models* — these include all algorithmic-type recipes for performing operations; some typical examples may vary from linguistically

represented procedures, e.g. ISO, to programs encoded with any programming languages.

When speaking about Conceptual Models one usually assumes more or less informal, abstract, illustrative presentation of concepts, relations, activities, etc. In case of Logical Models, clear syntax and semantic rules are in background; this assures possibility of identification and verification of properties, such as (i) *consistency*, (ii) *completeness*, (also: coverage), (iii) *unique interpretation* (lack of ambiguity), (iv) *efficiency* (minimal representation, lack of redundancy, efficient operation), (v) *processability*. Some further requirements may refer to: readability and transparency, easy modifications and extensionability, support for knowledge acquisition and encoding, etc.

The above-mentioned models are used to represent, analyze, process and optimize knowledge. Note that there are at least the following types of knowledge aimed at separate goals and requiring different way of processing:

- *typological* or *taxonometric* knowledge (e.g. a taxonomy in typed logics and languages or TBox in Description Logics),
- *factographic* knowledge representing facts and relations about object (e.g. a set of the FOPC atomic formulae or ABox in DL),
- *inferential* or *transformation* knowledge — specification of legal knowledge rewriting rules or production rules,
- *integrity and constraints* knowledge on what is impossible, not allowed, etc.
- *meta-knowledge* — all about how to use the basic knowledge (e.g. inference control rules).

Now, most typical KM activities require solving such issues as:

1. Knowledge Representation,
2. Inference — knowledge processing rules,
3. Inference Control — principles on how to apply inference rules in a correct and efficient manner,
4. Knowledge Acquisition and Updating,
5. Knowledge Analysis and Verification,
6. Friendly User Interface,
7. Generalization and Learning.

A tool, or a set of tools, for efficient Knowledge Management should support as many KM activities in a smooth way and deserve handling as many types of knowledge within a single framework.

2.1 BPMN as a tool for Knowledge Management

BPMN [1] appears to be an effective choice for Knowledge Management tasks. It offers a wide spectrum of graphical elements for visualizing of events, activities, workflow splits and merges, etc. It can be classified as Conceptual Modeling tool of high expressive power, practically useful and still readable to public.

Let us briefly analyze the strengths and weaknesses of BPMN as a KM tool. It is mostly a way of *procedural knowledge specification*, so it supports p. 3 above, but

neither p. 1 nor 2. Certainly, refering to p. 6 its user interface is nice. An important issue about BPMN is that it covers three important aspects of knowledge processing; these are:

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

Some more serious weakness issues concerning characteristics and activities presented in Section 2 are as follows:

- BPMN — being a Conceptual Modeling tool — does not provide formal semantics,
- it is inadequate for knowledge analysis and verification (p. 5),
- it neither support declarative representation of taxonometric, factographic, nor integrity knowledge.

However, some of these weaknesses can be overcome by combination of BPMN with Business Rules.

2.2 Business Rules as a tool for Knowledge Management

Business Rules (BR) can be classified as Logical Model for Knowledge Representation (KR). They constitute a declarative specification of knowledge. There can be different types of BR serving different purposes; in fact all the types of knowledge (taxonometric, factographic, transformation, integrity, and meta) can be encoded with BR.

A closer look at foundations of Rule-Based Systems [20] shows that rules can:

- have high expressive power depending on the logic in use,
- provide elements of procedural control,
- undergo formal analysis.

Some modern classification cover the following types of rules:

- facts – rules defining true statement (with 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) [21], are easier to analyze. However, the possibility of analysis depends on the accepted *knowledge representation language*, and in fact – the logic in use. Formal models of rule-based systems and analysis issues are discussed in detail in [20].

The main weakness of BR consists in lack of procedural (inference control) specification and transparent knowledge visualization. However, these issues can be solved at the BPMN level.

2.3 BPMN and BR: Towards an Integration Framework

In order to integrate BPMN and BR, a framework combining and representing intrinsic mechanisms of these two approaches is under development. It should be composed of the following elements:

- Workflow Structure/Sequence Graph (WSG) — an AND-OR graph representing a workflow structure at abstract level,
- Logical Specification of Control (LSC) — logical labels for WSG,
- Dataflow Sequence Graph (DSG) — a DFD-type graph showing the flow of data,
- Logical Specification of Data (LSD) — constraints imposed on data being input, output or processed at some nodes.
- Temporal Constraint Specification (TCS) — not discussed in this paper.

3 WORKFLOW STRUCTURAL GRAPH for BPMN

Workflow Structure/Sequence Graph (WSG) is in fact a simplified structural model of BPMN diagrams. 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 two subtypes of gateways: split and join operations. Finally, workflow sequence is modeled by directed links. No time or temporal aspect is considered.

The following elements will be taken into consideration [11]:

- $\mathbb{S}$ — 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 path can be followed (a split of the $EX - OR$ type),
  - $\mathbb{GP}$ — a set of *parallel splits* where all the paths of the workflow are to be followed (a split of the AND type or a *fork*), and
  - $\mathbb{GO}$ — a set of *inclusive splits* where one or more paths should be followed (a split of the OR type).
- $\mathbb{M}$ — a set of *merge gateways* or *joins*, where two or more paths meet; three 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 the $EX - OR$ type),
- $\mathbb{MP}$ — a set of *parallel merge* nodes where all the paths are combined together (a merge of the AND type), and
- $\mathbb{MO}$ — a set of *inclusive merge* nodes where one or more paths influence the subsequent item (a merge of the 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, an exemplary simple BPMN diagram is presented in Fig. 1.

In order to assure *structural correctness* of BPMN diagrams a set of restrictions on the overall diagram structure is typically defined; they determine the so-called *well-formed diagram* [9]. Classical AI graph search methods can be applied for analysis. 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.

The further issues, i.e. Logical Specification of Control (LSC), Dataflow Sequence Graph (DSG), and Logical Specification of Data (LSD) will be analyzed on the base of an example presented below.

4 BPMN AND BR EXAMPLE: THE THERMOSTAT CASE

In order to provide intuitions, the theoretical considerations will be illustrated with a simple exemplary 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: $aDD \in \{\mathit{monday}, \mathit{tuesday}, \mathit{wednesday}, \mathit{thursday}, \mathit{friday}\} \longrightarrow aTD = wd.$
Rule 2: $aDD \in \{\mathit{saturday}, \mathit{sunday}\} \longrightarrow aTD = wk.$
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: $aMO \in \{\mathit{january}, \mathit{february}, \mathit{december}\} \longrightarrow aSE = sum.$
Rule 8: $aMO \in \{\mathit{march}, \mathit{april}, \mathit{may}\} \longrightarrow aSE = aut.$
Rule 9: $aMO \in \{\mathit{june}, \mathit{july}, \mathit{august}\} \longrightarrow aSE = win.$
Rule 10: $aMO \in \{\mathit{september}, \mathit{october}, \mathit{november}\} \longrightarrow aSE = spr.$

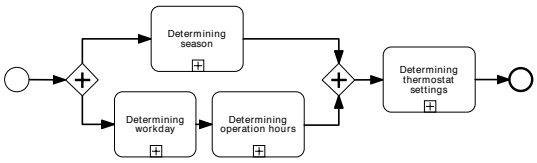

Figure 1. An example BPMN diagram — top-level specification of the thermostat system

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 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 presented 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 (aSE; it can take one of the values $\{sum, aut, win, spr\}$; the detailed specification is provided with rules 7-10). A visual specification of this activity with an appropriate set of rules is shown in Fig. 2.

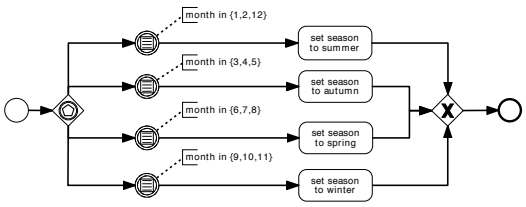

Figure 2. An example BPMN diagram — detailed specification a BPMN task

The lower path determines 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.

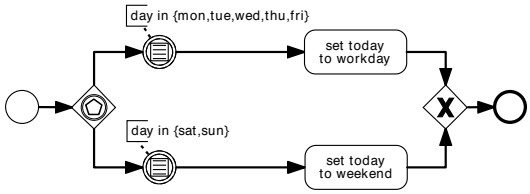

Figure 3. An example BPMN diagram — detailed specification of determining the day task

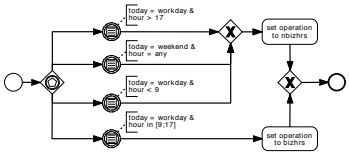

Figure 4. An example BPMN diagram — detailed specification of working hours task

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 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?

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 final output is correct or not.

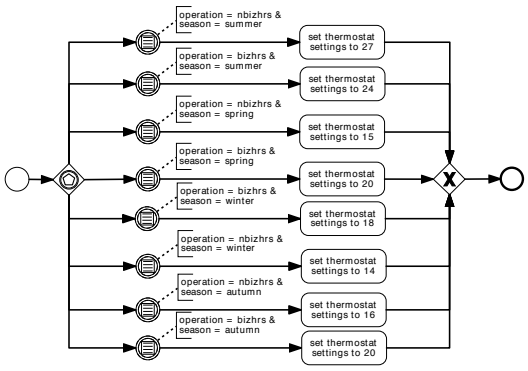

Figure 5. An example BPMN diagram — detailed specification of the final thermostat setting task

5 LOGICAL SPECIFICATION OF CONTROL

This section is devoted to analysis of logical specification of control. In fact, two types of control elements are analyzed: split nodes and merge nodes.

5.1 Analysis of Split Conditions

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:

$$\models q_1 \vee q_2 \vee \ldots \vee q_k, \tag{1}$$

i.e. the disjunction is in fact a tautology. In practice, to assure (1), 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 (1) can be relaxed to:

$$\phi \models q_1 \vee q_2 \vee \ldots \vee q_k. \tag{2}$$

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 (1), or at least (2). 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, it 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, $i = 1, 2, \ldots, k$.

5.2 Analysis of Merge Conditions

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.

$$\not\models p_i \wedge p_j \tag{3}$$

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:

$$\models p_1 \vee p_2 \vee \ldots \vee p_k, \tag{4}$$

i.e. the disjunction is in fact a tautology. If the input restrictions $\phi_1, \phi_2, \ldots, \phi_k$ are known, condition (4) 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:

$$\phi \models p_1 \vee p_2 \vee \ldots \vee p_k, \tag{5}$$

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 (4) or (5) 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:

$$\phi = \phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_k \tag{6}$$

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

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

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

$$\phi \models p_1 \wedge p_2 \wedge \ldots \wedge p_k. \tag{8}$$

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 of the component determining thermostat setting (see Fig. 1). 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:

$$\phi = (aSE = sum \vee aSE = aut \vee aSE = win \vee \\ aSE = spr) \wedge (aOP = dbh \vee aOP = ndbh).$$

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 (11) holds.

6 DATAFLOW SEQUENCE GRAPH

A Dataflow Sequence Graph (DSG) can be any DFD-type graph showing the flow of data that specifies the data transfers among data processing components. It shows that data produced by certain components should be sent to some next-in-chain ones. In the case of our thermostat example, it happens that the DSG can be represented with the graph shown in Fig. 1 — the workflow and the dataflow structure are the same.

7 LOGICAL SPECIFICATION OF DATA

Logical Specification of Data (LSD) are constraints on data being input, output or processed at some nodes.

7.1 Logical Constraints on Component Behavior

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 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:

$$T = (\psi_T, \varphi_T, \mathcal{A}), \tag{9}$$

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 algorithm $\mathcal{A}$.

The specification of a rule-based process component given by (9) 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:

$$\not\models \psi_i \wedge \psi_j \tag{10}$$

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* [20].

7.2 Logical Specification of Data Flow

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:

$$\phi \models \psi_T, \tag{11}$$

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 (11) holds.

8 ANOMALIES

Answering questions about essential properties of the model, such as correctness, consistency, termination/completeness requires at the very beginning verification whether the model contains any anomalies. These anomalies may appear in the model in all five layers of the framework proposed in section 2.3. Proper detection of an anomaly can in simple case use information delivered and available within single layer of the framework, but in more complex cases it must utilize information which are derived from two or even more layers. An example of the former is deterministic deadlock (XOR split gateway followed by an AND join gateway) where only graph structure determines occurrence of an anomaly in control flow, whereas an example of the latter is potential livelock (model with infinite cycle, AND split gateway followed by OR join gateway) where besides specific graph structure also some control flow preconditions must be satisfied.

Essential anomalies which can be detected by framework are as follows:

1. Within Workflow Structure/Sequence Graph and Logical Specification of Control:
   - deadlocks;
   - livelocks;
   - races;
   - lacks of synchronization;
   - dead tasks/nodes;
   - starvation;
2. Within Dataflow Sequence Graph and Logical Specification of Data:
   - too restrictive preconditions;
   - implicit routing;
   - implicit constraints on the execution order;

Business process is *deadlocked* if activities, which are on path to the end event, are waiting infinitely for a token that can only be passed by another activity in the process. In such situation, all activities of business process are put into idle state, therefore none operations of business process are carried out. Common example of an anti-pattern where deadlock can occur is construction consisting of an XOR/OR split together with an AND join. In such anti-pattern XOR/OR split creates alternative paths that are later joined by an AND join causing that eventually process cannot make any forward progress. Examples of deterministic and potential *deadlock* are illustrated with Fig. 6 and Fig. 7.

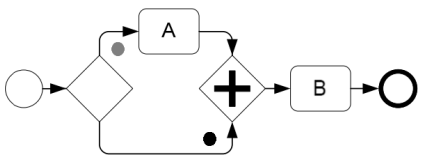

Figure 6. *Deterministic deadlock*

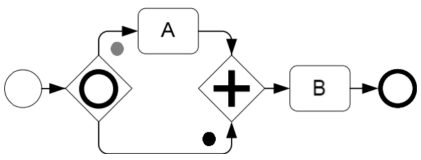

Figure 7. *Potential deadlock*

In the literature numerous different descriptions of *livelock* have been used, however taking into consideration business processes, only one of them can be applied, i.e. infinite execution. Infinite execution would occur in business process diagram when the individual activities of a business process model are running successfully but the model as a whole stuck in a loop. Such erroneous situation may occurs, in trivial case, when activity *A* always passes token to activity *B* that similarly, continuously passes a token back to process *A*. A *livelock* is very similar to a *deadlock*, with an exception that the states of the processes participating in the livelock continually change with regard to one another without any progress [23]. Common, significant property which joins definitions of deadlock and livelock is impossibility to maintain liveness of a business process model. Examples of deterministic and potential *livelock* are illustrated with Fig. 8 and Fig. 9.

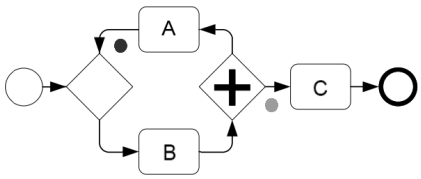

Figure 8. *Deterministic livelock*

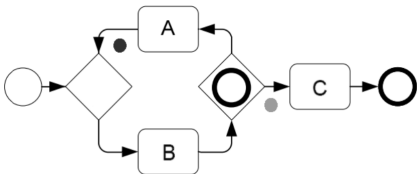

Figure 9. *Potential livelock*

Race in business process means situation when result of the execution of business process depends on order in which activities are performed. In model depicted in Fig. 10 when the former token is before split gateway *s2* and the latter one before merge gateway *j2*, gateway *j2* is allowed to be activated since there exists direct path from gateway *s2* to gateway *j2*. Alternatively, when tokens are respectively before merge gateway *j2* and before split gateway *s3*, gateway *j2* cannot be activated unless token will leave gateway *j3*. Such two scenarios lead to totally different execution of the business process.

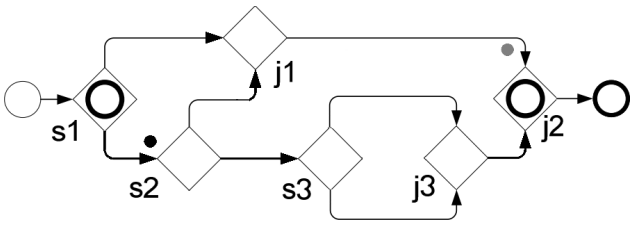

Figure 10. Races

Lack of synchronization occurs in case of multiple, unintentional activation of the activity. Such situation may be triggered as result of appearance more than one token in the same flow branch. *Lack of synchronization* may occurs after XOR join gateway

which merges two paths coming from OR/AND split gateway. Since BPMN uses implicit termination semantics, which means that a process instance completes only when each token existing in the model reaches an end event, *lack of synchronization* does not trigger *multiple termination* anomaly.

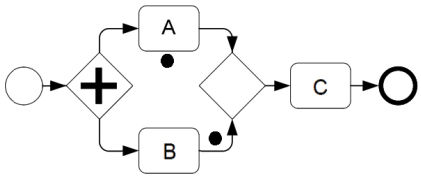

Figure 11. Deterministic lack of synchronization

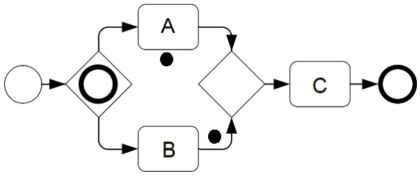

Figure 12. Potential lack of synchronization

Dead task within BPMN model is an activity that can never be reached and executed. Occurrence of *dead task* is always connected with existence of *dead transition*. This anomaly could occur because of some specific input preconditions, that would always route control flow omitting one of all possible flows Fig. 14 or in simple case there is no path from start to task/node or no path from task/node to the end Fig. 13.

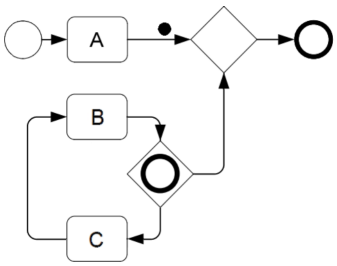

Figure 13. Dead tasks

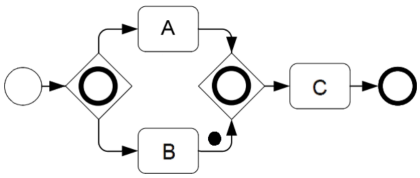

Figure 14. Potential dead task

Starvation anomaly can be observed in message flow when specific task is waiting infinitely for a message from another task. This anomaly may arise because of existence of dead tasks in process or simply because of specific control flow which omits task responsible for sending messages between pools.

Fig. 15 depicts case when task *B1* of *Pool 1* potentially cannot make any forward progress due to specific input precondition which route control flow omitting task *A2* of *Pool 2*. Although control flow in *Pool 1* seems to be correct the whole process cannot be terminated. In contrast to deadlock and livelock which can be considered only within single pool, starvation is a consequence of improper interactions between pools.

Too restrictive preconditions [24] is an error that occurs when task which is ready for for an execution from a flow control perspective is put into idle state because of

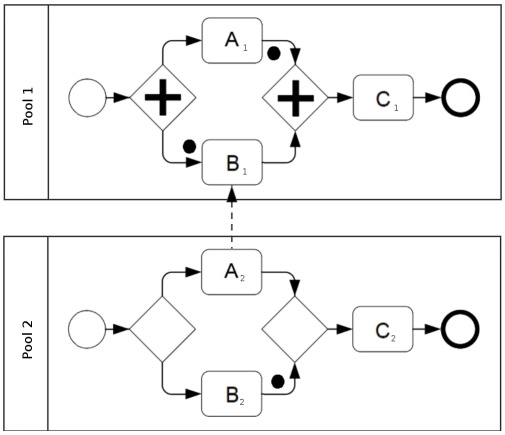

Figure 15. Starvation

waiting for a particular data object state. This data object state is a necessary condition for an activation of task. In Fig. 16 task *D* is expecting data object to be in state *State 4* while there is possibility that data object will be in *State 3*.

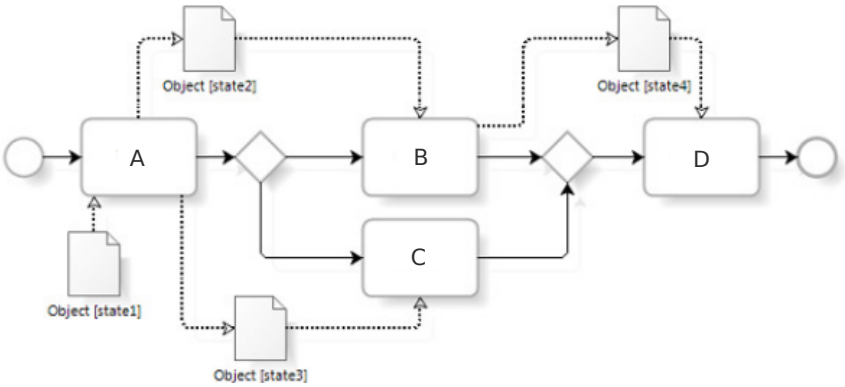

Figure 16. Data flow anomalies

Implicit routing is an anomaly which causes that some necessary data preconditions are omitted during execution of the process. Such situation can happen in model with XOR/OR split gateway where improper path is selected, eventually resulting in omitting some data object. Although process still can be continued, final result may be wrong. Example of this problem is depicted in Fig. 16 where task *A* can change data object's state to *State 2* whereas the control flow could be still routed to task *C* by XOR split gateway. Such anomaly could be considered as a specific variant of *too restrictive precondition* error.

Implicit constraints on the execution order anomaly takes into account situations when two concurrent activities share preconditions. The problem may occur when one of these activities updates the state of the data object.

9 CONCLUSIONS AND FUTURE WORK

In this paper, BPMN and BR were explored as tools for Knowledge Management. It is argued that integration of these approaches can overcome some disadvantages of these approaches when considered in separate. Four areas of knowledge specification were put forward: Workflow Specification Graph, Logical Specification of Control (missing in current BPMN), Dataflow Sequence Graph and Logical specification of Data. The ideas of knowledge representation and analysis were illustrated with an example.

The original contribution of our work consists in presenting a framework combining BPMN and BR as a tool for Knowledge Management. The papers only deals with a restricted view of Knowledge Management, where knowledge can be modeled explicitly in a formal representation, and it does not take into account the knowledge residing in people's heads.

As future work, a more complex modeling and verification 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 [25]. XTT2 [15] constitutes a formalized attributive language for representing rules in decision tables. Thus, the XTT2 rules (and tables) can be formally analyzed using the so-called verification HalVA framework [17] Although table-level verification can be performed with HalVA [26], the global verification is a more complex issue [27]. Our preliminary works on global verification have been presented in [28].

References

1. OMG: Business Process Model and Notation (BPMN): Version 2.0 specification. Technical Report formal/2011-01-03, Object Management Group (2011)
2. Ambler, S.W.: Business Rules. http://www.agilemodeling.com/artifacts/ businessRule.htm (2003)
3. 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, New York (2009)
4. OMG: Production Rule Representation RFP. Technical report, Object Management Group (2003)
5. OMG: Semantics of Business Vocabulary and Business Rules (SBVR). Technical Report dtc/06-03-02, Object Management Group (2006)
6. Kluza, K., Nalepa, G.J., Łysik, Ł.: Visual inference specification methods for modularized rulebases. Overview and integration proposal. In Nalepa, G.J., Baumeister, J., eds.: Proceedings of the 6th Workshop on Knowledge Engineering and Software Engineering (KESE6) at the 33rd German Conference on Artificial Intelligence September 21, 2010, Karlsruhe, Germany, Karlsruhe, Germany (2010) 6–17

7. Dijkman, R.M., Dumas, M., Ouyang, C.: Formal semantics and automated analysis of BPMN process models. preprint 7115. Technical report, Queensland University of Technology, Brisbane, Australia (2007)
8. Lam, V.S.W.: Formal analysis of BPMN models: a NuSMV-based approach. International Journal of Software Engineering and Knowledge Engineering **20** (2010) 987–1023
9. Ouyang, C., Wil M.P. van der Aalst, M.D., ter Hofstede, A.H.: Translating BPMN to BPEL. Technical report, 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)
10. Wynn, M., Verbeek, H., Aalst, W.v.d., Hofstede, A.t., Edmond, D.: Business process verification – finally a reality! Business Process Management Journal **1** (2009) 74–92
11. Ligęza, A.: BPMN – a logical model and property analysis. Decision Making in Manufacturing and Services **5** (2011) 57–67
12. Ligęza, A., Nalepa, G.J.: Knowledge representation with granular attributive logic for XTT-based expert systems. In Wilson, D.C., Sutcliffe, G.C.J., FLAIRS, eds.: FLAIRS-20: Proceedings of the 20th International Florida Artificial Intelligence Research Society Conference: Key West, Florida, May 7-9, 2007, Menlo Park, California, Florida Artificial Intelligence Research Society, AAAI Press (2007) 530–535
13. 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** (2011) 117–137
14. Nalepa, G.J.: Semantic Knowledge Engineering. A Rule-Based Approach. Wydawnictwa AGH, Kraków (2011)
15. Nalepa, G.J., Ligęza, A.: HeKatE methodology, hybrid engineering of intelligent systems. International Journal of Applied Mathematics and Computer Science **20** (2010) 35–53
16. Nalepa, G.J., Kluza, K.: UML representation for rule-based application models with XTT2-based business rules. International Journal of Software Engineering and Knowledge Engineering (IJSEKE) **22** (2012) 485–524
17. Kluza, K., Maślanka, T., Nalepa, G.J., Ligęza, A.: Proposal of representing BPMN diagrams with XTT2-based business rules. In Brazier, F.M., Nieuwenhuis, K., Pavlin, G., Warnier, M., Badica, C., eds.: Intelligent Distributed Computing V. Proceedings of the 5th International Symposium on Intelligent Distributed Computing – IDC 2011, Delft, the Netherlands – October 2011. Volume 382 of Studies in Computational Intelligence. Springer-Verlag (2011) 243–248
18. Bobek, S., Kaczor, K., Nalepa, G.J.: Overview of rule inference algorithms for structured rule bases. Gdansk University of Technology Faculty of ETI Annals **18** (2010) 57–62
19. Nalepa, G., Bobek, S., Ligęza, A., Kaczor, K.: Algorithms for rule inference in modularized rule bases. In Bassiliades, N., Governatori, G., Paschke, A., eds.: Rule-Based Reasoning, Programming, and Applications. Volume 6826 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2011) 305–312
20. Ligęza, A.: Logical Foundations for Rule-Based Systems. Springer-Verlag, Berlin, Heidelberg (2006)
21. Ligęza, A., Szpyrka, M.: Reduction of tabular systems. In Rutkowski, L., Siekmann, J., Tadeusiewicz, R., Zadeh, L., eds.: Artificial Intelligence and Soft Computing - ICAISC 2004. Volume 3070 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2004) 903–908
22. Negnevitsky, M.: Artificial Intelligence. A Guide to Intelligent Systems. Addison-Wesley, Harlow, England; London; New York (2002) ISBN 0-201-71159-1.
23. Mogul, J.C., Ramakrishnan, K.K.: Eliminating receive livelock in an interrupt-driven kernel. ACM Trans. Comput. Syst. **15** (1997) 217–252

24. Awad, A., Decker, G., Lohmann, N.: Diagnosing and repairing data anomalies in process models. In Rinderle-Ma, S., Sadiq, S., Leymann, F., eds.: Business Process Management Workshops. Volume 43 of Lecture Notes in Business Information Processing. Springer Berlin Heidelberg (2010) 5–16
25. Kluza, K., Kaczor, K., Nalepa, G.J.: Enriching business processes with rules using the Oryx BPMN editor. In Rutkowski, L., [et al.], eds.: Artificial Intelligence and Soft Computing: 11th International Conference, ICAISC 2012: Zakopane, Poland, April 29–May 3, 2012. Volume 7268 of Lecture Notes in Artificial Intelligence., Springer (2012) 573–581
26. Nalepa, G., Bobek, S., Ligęza, A., Kaczor, K.: HalVA - rule analysis framework for XTT2 rules. In Bassiliades, N., Governatori, G., Paschke, A., eds.: Rule-Based Reasoning, Programming, and Applications. Volume 6826 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2011) 337–344
27. 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): November 10, 2011, La Laguna (Tenerife), Spain. (2011) 15–23
28. 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., Nieuwenhuis, K., Pavlin, G., Warnier, M., Badica, C., eds.: Intelligent Distributed Computing V. Proceedings of the 5th International Symposium on Intelligent Distributed Computing – IDC 2011, Delft, the Netherlands – October 2011. Volume 382 of Studies in Computational Intelligence. Springer-Verlag (2011) 249–255