Journal on Data Semantics (2021) 10:3–18
https://doi.org/10.1007/s13740-021-00127-w

ORIGINAL ARTICLE

Execution of Knowledge-Intensive Processes by Utilizing Ontology-Based Reasoning

ODD-BP: An Ontology- and Data-Driven Business Process Model

Eric Rietzke[1,4] · Carsten Maletzki[2] · Ralph Bergmann[2,3] · Norbert Kuhn[1]

Received: 2 February 2020 / Revised: 16 March 2021 / Accepted: 23 April 2021 / Published online: 17 May 2021

Abstract
Modeling and executing knowledge-intensive processes (KiPs) are challenging with state-of-the-art approaches, and the specific demands of KiPs are the subject of ongoing research. In this context, little attention has been paid to the ontology-driven combination of data-centric and semantic business process modeling, which finds additional motivation by enabling the division of labor between humans and artificial intelligence. Such approaches have characteristics that could allow support for KiPs based on the inferencing capabilities of reasoners. We confirm this as we show that reasoners can infer the executability of tasks based on a currently researched ontology- and data-driven business process model (ODD-BP model). Further support for KiPs by the proposed inference mechanism results from its ability to infer the relevance of tasks, depending on the extent to which their execution would contribute to process progress. Besides these contributions along with the execution perspective (start-to-end direction), we will also show how our approach can help to reach specific process goals by inferring the relevance of process elements regarding their support to achieve such goals (end-to-start direction). The elements with the most valuable process progress can be identified in the intersection of both, the execution and goal perspective. This paper will introduce this new approach and verifies its practicability with an evaluation of a KiP in the field of emergency call centers.

Keywords Ontology · Data-oriented business process · Knowledge-intensive process · Inferencing

1 Introduction

This paper introduces and evaluates an ontology- and data-driven business process approach (ODD-BP), which aims to deal with the special requirements of knowledge-intensive processes (KiPs). The motivation arises from the research about KiPs with its data-oriented character [7,10], which is why we placed the data "into the driver's seat" of our new approach [18,22–24]. Several BP-approaches have addressed the advantages of declarative and data-oriented workflow principles [5,7,16,27]. The data-oriented character of ODD-BP reflects those works by collecting the essence of these approaches to set up a reduced and simplified metamodel for further examinations. In parallel, several papers have addressed the advantages of semantic process modeling (SPM) [11,26] to reduce the ambiguity of process definitions and allow process contribution by utilizing reasoners to infer process relevant knowledge. Because most of the approaches related to SPM have focused on control-flow oriented workflow principles, the data-oriented approaches have not been combined with a semantic process definition. This work focuses on combining both methodologies to provide an ontology and data-driven approach that fits the needs of KiPs and provides significant process contributions through a semantic process definition.

Today's established business process, approaches usually follow a control-flow-oriented principle. Operational knowledge of a particular domain is often integrated as alternative routes (splits and joins) through a process model, offering just a limited amount of flexibility (Flexibility by Design). In this scenario, a rich amount of operational knowledge would lead

✉ Eric Rietzke
eric.rietzke@livereader.com

1 Trier University of Applied Sciences, Trier, Germany

2 University of Trier, Trier, Germany

3 German Research Center for Artificial Intelligence (DFKI) Branch University of Trier, Trier, Germany

4 LiveReader GmbH, Oberthal, Germany

to a very complicated process model, which extends the complexity that a process designer can handle. Since KiPs rely on a wide range of knowledge, these solutions with a limited set of predefined paths do not cope with the specific demands of KiPs. To overcome these limitations, the ODD-BP approach follows a declarative and data-oriented principle where the process steps are not predefined, but the result of a permanent process planning procedure. This allows the integration of a wide-ranging set of rules, reflecting extensive operational, relevant expert knowledge.

This work presents a full integration of all process affecting knowledge into one joint knowledge base, starting with a metamodel that provides the essence of data-oriented workflow principles. Additionally, domain-specific knowledge is integrated into the knowledge base defining the overall information model and considering further expert knowledge. Besides this terminological knowledge, the process definition and the explicit situational facts of a specific process instance are represented as assertional knowledge. We expect that such a fully integrated knowledge base can be utilized to deduce the process states of process elements and can be used to drive a process instance without the need for a separate workflow engine. Furthermore, we expect that the integrated domain knowledge can affect a process execution without being explicitly part of a process definition and reduce its overall complexity.

Our work is part of the research project SEMANAS[1] [22] and focusses a KiP in the field of an emergency call center. In this paper, we introduce a metamodel, OWL2, and SWRL-rules as part of the knowledge base and we show how these rules allow inferring the executability of tasks and the overall relevance of process elements. We show how such a permanent process planning procedure, combined with a data-driven approach, enables a flexible process execution that supports the requirements of KiPs. The variability of such flexible process executions is measured and compared with the state-of-the-art static process as a reference. Additionally, we show that AI inferencing can utilize expert knowledge within the knowledge base to support users in their decision-making processes. For evaluation, we measure the quality of the process outcome and compare the results between state of the art and an enhanced process definition, which takes advantage of the ODD-BP approach.

Section 2 presents related topics like data-centric and semantic business process modeling. The application scenario in emergency call centers is described in Sect. 3. Section 4 briefly introduces the ODD-BP Model with its main concepts. The rules to deduce executability and relevance of process elements are defined and explained in Sect. 5. With the help of use cases, the general feasibility of the new approach is shown in Sects. 6 and 7. Section 8 presents the results of the evaluation ahead of conclusion in Sect. 9.

2 Related Work

Semantic process modeling (SPM) uses ontologies to enhance common semi formal representations of processes. During SPM, elements of processes (such as tasks) are represented as instances of ontology classes describing the constructs of a process language [26]. Research on SPM is often motivated by easing semantic ambiguity in process models and focusses mainly on control-flow oriented approaches [3,11,26]. According to Heinrich et al. [14], an essential advantage of SPM is accessibility for automated processing. This has been studied in terms of automated creation or adjustment of process models, and it was observed that reasoners can be used to derive facts that are implicitly stated in process models [3,14,26].

In data-centric business process modeling, processes are not described by links between tasks (as in traditional control-flow oriented approaches like BPMN—Business Process Model and Notation), but instead, tasks are linked to data elements that are required for their execution (input) or the result of it (output). This implies a process logic that focuses on data availability rather than on predefined sequences of tasks [23]. Several approaches of data-centric business process modeling consider data in the context of artifacts that are created, evolved, and typically archived during their lifetime in businesses [7,27]. Di Ciccio et al. [10] offer an analysis of different data-centric approaches and evaluate their coverage regarding the characteristics and requirements of KiPs. CMMN [20] as an example of a well-established and by the OMG-defined standard steps up to eliminate a significant drawback of BPMN, the rigid control-flow oriented characteristic, and offers a flexible process execution. In fact, CMMN captures work methods that are based on the handling of cases requiring various activities that may be performed in an unpredictable order; especially, the concept of a *Sentry* represents a core characteristic of a data-centric approach by defining activities as *enabled* or *active* according to related data. Nevertheless, CMMN does not provide an assessment regarding the relevance of activities, nor does it provide a goal-oriented strategy to support users with meaningful guidance. These are essential requirements, but for a more detailed analysis of different approaches and how they meet the specific requirements of KiPs, we refer to [10].

Despite these two usually separated research fields, especially the combination of SPM and data-oriented principles promises advantageous capabilities. Due to the dynamics of knowledge-intensive processes and their close relationship to information and knowledge, it seems promising if a supporting system is based on semantic technologies [9]. Similar

[1] SEMANAS is funded by the Federal Ministry of Education and Research (BMBF), grant no. 13FH013IX6, duration: 2017–2021.

assessments can be found in [13,17]. In workflow management systems, workflow engines may be used to identify what could be done and who could do it [28]. To the best of our knowledge, it has not been investigated yet whether reasoners can take over the job of workflow engines to identify what could be done next in a process. Additionally, we could not identify a combined approach that has verified the expected benefits while executing KiPs.

3 Application Scenario

In general, KiPs are highly dependent on knowledge workers (like managers, researchers, and engineers) who perform interconnected knowledge-intensive decision-making tasks [10,27]. Since knowledge workers are the most significant determinant of the worth of their organizations [8], supporting them in KiPs is accordingly essential. One of such a KiP can be found in an emergency call center (ECC), where operators handle incoming calls. The successful handling of an emergency call (EC) relies on the operator's knowledge and his capability to interpret the data given by the caller, and thus, they are highly knowledge-intensive processes.

Today, in these calls, the operators follow a predefined set of questions according to an emergency call guideline (ECG), realized within an ECC-software. While the ECG ensures that the most important questions are asked (who, where, how many), the guideline lacks in supporting the operator to achieve a detailed insight about "what happened." Current ECGs follow decision trees' principles and thus have the same characteristics as control-flow-oriented business process models. As a result, the operator has to follow the questions in a fixed order. Flexibility is only integrated to a small amount and one specific case leads to exactly one path through the decision tree. The manifold possible situations about "what happened" cannot be covered by such a static system as it would end in a decision tree with immense size, impossible to design and maintain. This ends up in the situation that the ECGs are just supportive of covering an initial part of an emergency call, offering no flexibility, and the operators must fill the gap with their knowledge. Since the operators have a different amount of experience, different backgrounds (paramedic, firefighters), and even each person's individual performance varies over the work-time, the outcome of an emergency call procedure (ECP) is not consistent.

With today's inflexible ECGs in mind, operators cannot transfer significant facts, often given by the caller's opening statement, to the system. In fact, the operator has to come back to such initial given data afterward. With a view to the rising cases where emergency calls are initiated or executed by IoT devices, the need for a data-oriented and flexible approach becomes even more important. With the eCall, modern cars can perform an emergency call in case of an accident and transfer relevant data to the ECC. Modern smartwatches can detect a fall, call the ECC, and initially deliver answers to seven out of the operator's nine most important questions. We demonstrate that a system following the ODD-BP approach can accept and integrate such datasets into the ECP at any time. In the following, it can adapt the process according to the already received and additionally required data and can offer a significant advantage compared to state of the art. Additionally, we will show how the new approach integrates expert knowledge, supporting the operator while dealing with the "what happened" aspects. With the help of an inferring artificial intelligence, we demonstrate how the expert knowledge is utilized to consider the current state of facts and deduces which further questions are worth asking to gain more valuable data. Finally, we show how the operator can be supported to choose appropriate rescue resources. The evaluation (Sect. 8) will demonstrate that the ODD-BP approach offers a flexible and adaptive process execution and supports a consistent and improved process outcome of ECPs.

4 ODD-BP—Ontology- and Data-Driven Business Process Model

The first and foremost difference between the established process executing systems and ODD-BP is the semantic integration of all process-relevant knowledge into a unified knowledge base. Usually, knowledge from different sources affects a process execution, and these sources are located in separated segments. Typically, a workflow engine contains the knowledge on how to execute a process and thus defines a metamodel, which developers implicitly manifest in program code. The data model and customizations reflect domain knowledge and are often defined by the ER-model of a database or individualized scripts. A process definition specifies a template of how a certain kind of processes can be executed and is often described with a specialized language. Additionally, each process instance carries situational knowledge, reflecting as a digital twin some relevant facts about the real world. As shown on the left side in Fig. 1, these knowledge sources are usually embedded in separated segments, and each has a significant impact on the execution of a process.

Opposed to this, the ODD-BP approach follows the idea of semantic process modeling [11] and integrates these knowledge parts semantically into one unified knowledge base as presented in Fig. 1 at the right side. The foundation is built by a base ontology that defines the core elements of a process and the possible relations and rules. The domain ontology defines the data model for a specific domain and generalized domain knowledge with essential or useful expert knowledge. Both

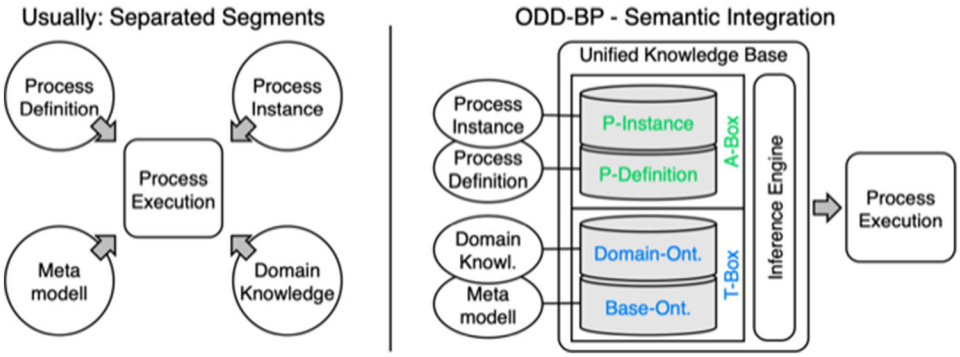

Fig. 1 Process relevant knowledge and it's influence to process execution

define a set of conceptualizations in the T-Box, a finite set of terminological axioms. The process definitions and the situational knowledge of process instances are reflected by sets of linked individuals in the A-Box, a finite set of assertional axioms. This unified knowledge base allows taking advantage of a reasoner to deduce process-relevant facts to support a process execution (Sect. 5).

4.1 Metamodel

All established workflow approaches follow a specific and predefined metamodel [25]. According to a wide range of publications [12,19], a metamodel defines "the frames, rules, constraints, models, and theories applicable and useful for modeling a predefined class of problems." Knowledge-intensive processes (KiPs) and their specific demands [10] can be considered as such a predefined class of problems. Within the ODD-BP approach, we define a data-oriented workflow metamodel aligned to the requirements of KiPs.

The centerpiece of ODD-BP is the base ontology, which defines all concepts and relations to build the metamodel to define and execute processes. The most fundamental artifacts of a process are *Tasks*, *Dataobjects*, and *Documents* and are usually represented in one or another way in all workflow approaches. The metamodel manifests its specific character through the kind of relations between these artifacts. Accordingly, the data-oriented character of the ODD-BP approach is created by the relations between *Tasks* at the one side and *Documents*, *Dataobjects*, and *Attributes* on the other side. An excerpt from the base ontology, which is implemented using OWL2,[2] is depicted in Fig. 2, where circles with dashed lines represent classes and directed edges represent object properties between classes.

According to this metamodel, a *Document* can be *demanded_by* a *Task* as input or a *Task* can *produce* a *Document* as the outcome of its execution. Analogous to this, a *Dataobject* or an *Attribute* can be *required_by* a *Task* as input or a *Task* can deliver such elements as output. The deeper meaning of *Dataobject* and *Attribute* will be explained more in detail in the following, but the general importance of data for a process execution is already apparent.

All in all, the metamodel defines how a process can be designed and executed, and these general rules are the same for process definitions *PD* and process instances *PI*, which are defined as a specialization of the concept *Process*. A process is modeled and described by individuals of the introduced concepts and by links between these individuals according to the base ontology relations. Since process instances represent the ongoings in a business process and are meant to hold data about the process state [28], they are relevant for inferring facts to drive the process execution.

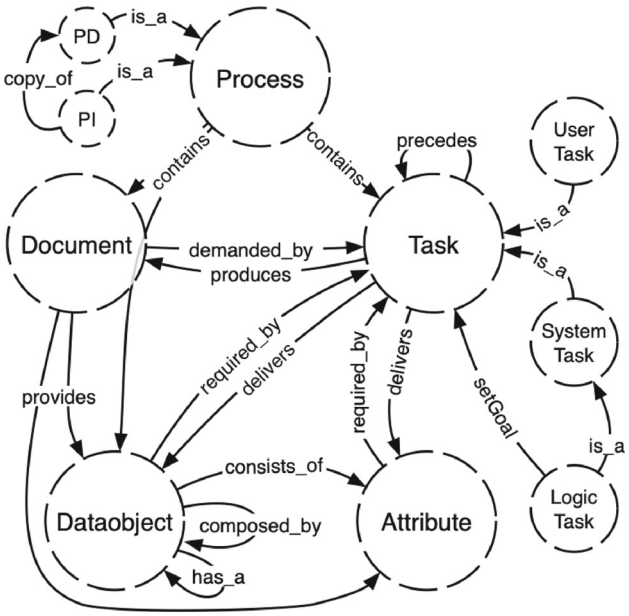

Fig. 2 Base ontology of the ODD-BP model

[2] OWL 2 spec: https://www.w3.org/TR/owl2-syntax.

4.2 Data Objects and Attributes

The general data-orientation of the metamodel can be seen by the manyfold relations between *Tasks* and the data-carrying elements *Dataobjects*, *Attributes*, and *Documents*, in certain ways similar to artifact centric approaches [4,5,7]. However, the ODD-BP approach is not only placing data into a more central position; the data are integrated with the intention to drive a process.

Usually, an information system organizes data about the real world with entities and relations. With a view to databases, entities are managed as entries into a table, while a knowledge store manages entities as individuals of a specific class. Such an individual represents an object of the real world, while its object characteristics are stored as data properties of the individual. This realization lacks in expressiveness, as the knowledge store cannot express dependencies between tasks and data-properties, and thus, the data properties could not be used to drive the process.

This leads us to a conceptualization in which object characteristics are represented through a separate concept, Attributes. They can be understood as key–value pairs, while an individual of this type represents a single characteristic of an entity. The existence of such Attributes can now be used to infer further facts about the process state and thus can be used to drive the process.

4.3 Process Example Utilizing the Metamodel

Figure 3 depicts an excerpt of a process instance of an emergency call procedure, which is modeled using the presented base ontology. The process instance contains a dataobject representing a *Person* with three attributes *self-affected*, *conscious*, and *breathing*. The Attribute *self-affected* is used to perform a *Conscious Check* to determine if the caller is also the affected person. If this is the case, the person is obviously conscious, which makes the question-task *Is Person responsive* irrelevant and can be skipped.

The shape of the elements represents different concepts of the metamodel, as shown by the legend. Process states in the ODD-BP model are represented by marking unavailable data elements and unexecuted tasks as so-called *Placeholders*, an additional concept that will be introduced in detail in the next section. Initially, a process instance only contains elements with *Placeholders* markings that are removed as the process proceeds, which leaves only elements behind that describe existing (meaningful) data elements and executed tasks. All white elements of the excerpt are marked as placeholder elements so that only the dataobject *Person*, the attribute *self-affected*, and the user task *Who is affected* are known. This simple excerpt may not represent all characteristics of KiPs, but it illustrates how data-driven aspects can be modeled using the ODD-BP approach.

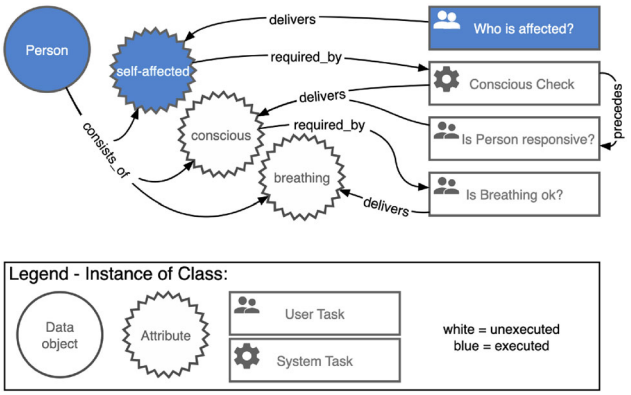

Fig. 3 Process instance modeled using the base ontology

The metamodel basically defines a declarative workflow approach where processes are designed by describing the general dependencies between data and task elements. This leads to a flexible and adaptive process execution compared to classic imperative control flow-oriented approaches like BPMN. Nonetheless, sometimes dependencies between tasks can be described more easily by control flow-like expressions like the *precedes* and *setGoal* relation (see Fig. 2), which allow direct connections between tasks. The excerpt in Fig. 3 is using a *precedes* constraint to ensure that the system task *Conscious Check* is executed before the user task *Is Person responsive* can be executed. Especially Logic Tasks, which offer the possibility to prove against predefined conditions, lead to an imperative segment within the overall declarative process definition. We do not discuss this more in detail since this paper's focus lies in the possibilities to utilize inference mechanisms for process execution, rather than the pros and cons of concurrent workflow principles, which are addressed within [7,23,24].

5 Inferring Process-Relevant Facts

This section presents an inference mechanism that aims at supporting knowledge workers by providing execution relevant information about all process elements, illustrated in Fig. 4. For doing so, the mechanism is inferencing in and against the execution direction process-relevant facts. It is worth noting that even without a control-flow-oriented principle, a flexible data-oriented approach like ODD-BP still has an execution direction in the graph along with the relations between the process elements. At first, the mechanism is inferencing in execution direction the states of data dependencies of unexecuted tasks to determine the executability ① of tasks. In a second step, the mechanism is inferencing in execution direction the execution relevance ② by considering if the possible outcome of a task provides any process contribution like new data elements. Additionally,

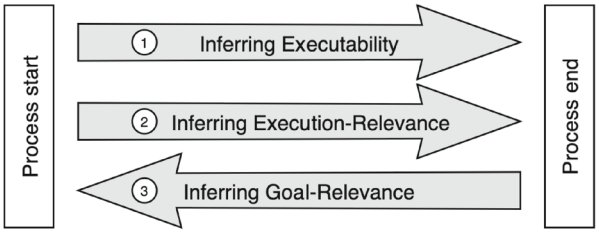

Fig. 4 Inferencing concept

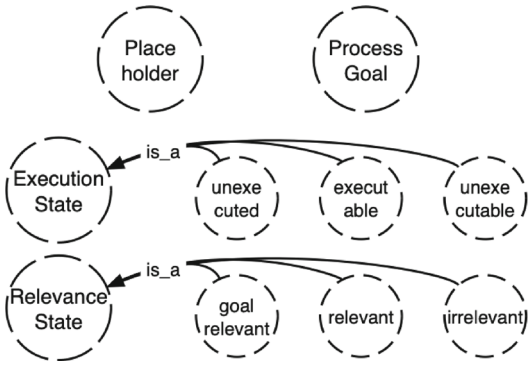

Fig. 5 Extension of the base ontology

the mechanism considers process goals and deduces against the execution direction the goal relevance ③ for each process element. These three parts of the inference mechanism will be introduced in the following three individual subsections.

As a first step, the base ontology requires additional concepts that can be used to express process relevant facts. Figure 5 presents an extension of the base ontology, and one concept (*Placeholder*) was already mentioned briefly. Any individual representing one of the introduced process elements can also be assigned to the class *Placeholder* to define the specific element as not-meaningful in regard to the process state of a process instance. In creating a new PI as a copy of a PD, any process element will be initialized this way. Once an element becomes meaningful or a task is executed, the *Placeholder* assignment of the individual will be removed.

While the *Placeholder* concept is used to express the execution state of the process elements, the concept *Process Goal* is designed to express which elements of a PI must be achieved. This allows an adaptive and flexible behavior that fits very well to the demands of KiPs [10], and we support different ways how a process element also becomes a *Process Goal* assignment as we explain in Sect. 5.5.

To infer process relevant facts, we additionally introduce the concept *Execution State* and *Relevance State*. The first one is exclusively for *Task* elements and offers three subclasses (*unexecuted*, *executable*, *unexecutable*). The second one offers also three subclasses (*goal-relevant*, *relevant*, *irrelevant*) and can partially be used for any process element. The deeper meaning of the State-concepts will be explained in detail through the formalization in the following sections.

5.1 Inferring Executability

Among the many reasoning options of OWL 2, most interesting for our purpose is inferring that an individual belongs to a class—in our case, inferring that a task belongs to the class of executable tasks. A class in OWL 2 is a set of individuals described by restrictions on the individual's characteristics [6]. In data-centric business process modeling, executable tasks share the characteristic that all required input data elements are available [23]. In order to infer executability, we need to describe this in OWL 2 by a class. Since OWL 2 relies on the open-world assumption (OWA), this is challenging if the result shall be suitable for practice. The OWA makes it impossible to tell if something is inexistent until its inexistence has been stated [15]. If an inference mechanism for executability relies on a class that is restricted by a statement about "everything," considerable effort has to be taken into account in the context of KiPs. For example: If a class describes the set of tasks whose input elements are all available, during reasoning, a statement has to tell that a task has no more than the known input data elements. Such is possible by adding statements that restrict the cardinality of relations between individuals [2] and herewith define a closure for a specific class. The challenge in this context is that a task can have a different number of incoming documents, dataobjects and attributes, and for each possible combination of these incoming data elements, an individual specialization of task is required with an according closure definition. This would lead to a wide range of specialized task concepts with individual rules to infer executability and yet it still remains uncompleted. Additionally, since KiPs require substantial flexibility at design- and run-time [27], data dependencies often have to be adjusted. As a result, this approach would be impractical since data dependencies and specialized task concepts with a corresponding closure rule would need to be adjusted.

Alternatively, reasoning can be realized with the help of existential quantifiers, which is easier in the OWA context. During reasoning, a statement using the existential quantifier becomes true if the existence of something is declared. It cannot be derived as true if the existence is not declared. Transferred to the discussed use case, we can decide the opposite—the unexecutability of a task—as soon as a single incoming required data element is stated as unavailable. This can be realized with a single universal *Task* concept and a generic rule, which does not require a closure definition and is much more suitable for practical use. Such an alternative way of describing executable tasks by examining unexecutability based on data availability will be introduced in the following and utilizes three different technologies, OWL2-rules, SWRL-rules, and Queries. While OWL2-rules and SWRL-rules are defined by formulas expressed in description logic, the Queries are expressed by SPARQL.

In the first step, we define unexecuted tasks as tasks that are marked as placeholders. Consequently, the class of unexecuted tasks corresponds to the intersection between the classes of placeholders and tasks (1).

$$Task_{unexecuted} \equiv Task \sqcap Placeholder \tag{1}$$

Due to placeholder markings, the class of unavailable data elements corresponds to the intersection of the classes of data elements and placeholders (3) and (4) (data element is abbreviated in formulas with "Data"). Since *unexecutable* tasks have at least one unavailable input data element or at least one *unexecuted preceding* task element, they can be described by the existential quantifier (5). Note that "input" is the super property of the *required_by* and *demanded_by* properties of the base ontology (2).

$$input \sqsupseteq required\_by \& \quad input \sqsupseteq demanded\_by \tag{2}$$

$$Data \equiv (Document \sqcup Dataobject \sqcup Attribute) \tag{3}$$

$$Data_{unavailable} \equiv Data \sqcap Placeholder \tag{4}$$

$$Task_{unexecutable} \equiv Task_{unexecuted} \sqcap (\exists input^{-}.Data_{unavailable} \sqcup \exists precedes^{-}.Task_{unexecuted}) \tag{5}$$

A task can be considered *executable* if it belongs to the class of *unexecuted* task and not to the class of *unexecutable* tasks. Since we do not introduce closure axioms, the executable tasks are derived with a SPARQL-query (6). In fact, the query's execution and interpretation can be seen as a subset of the knowledge-graph, which follows the rules of the closed-world assumption (CWA).

$$Task_{executable}(abbreviatedasT\_exec)$$
```
SELECT ?T_exec WHERE {
  ?T_exec rdf:type :unexecuted
  MINUS { ?T_exec rdf:type :unexecutable }}
```
(6)

It is important to note that the query is a simplification, and besides the definition of prefixes, it must be assured that the underlying system delivers the inferencing results as part of SPARQL-queries.

Although this alternative description of executable tasks appears rather cumbersome, it suits very well for practical use since intersections of classes can easily be queried from knowledge bases.

While the executability relies on the related input data element, the following section focuses on the existence of output data elements to assess a task execution's relevance.

5.2 Inferring Execution Relevance

In data-centric business process modeling, multiple tasks can lead to the same output data element. For example, one could either make a call or write an email to acquire the same data. When one makes a call and gets the data, writing an email becomes irrelevant. Alternatively, if the call did not produce the desired data, one could choose to write an email additionally. Hence, *unexecuted* tasks whose execution would lead to at least one unknown data element or precede at least one unexecuted task are *relevant* for execution. The class of such *relevant* tasks is expressed with formula (8). Note that "output" is the super property of the "delivers" and "produces" properties of the base ontology (7). Based on our previous findings, we can also describe combinations like executable relevant tasks with the help of another SPARQL-query (9).

$$output \sqsupseteq delivers \quad output \sqsupseteq produces \tag{7}$$

$$Task_{relevant} \equiv Task_{unexecuted} \sqcap (\exists output.Data_{unavailable} \sqcup \exists precedes.Task_{unexecuted}) \tag{8}$$

$$Task_{executable\_relevant}(abbreviatedasT\_ex\_re)$$
```
SELECT ?T_ex_re WHERE {
  ?T_ex_re rdf:type :unexecuted .
  ?T_ex_re rdf:type :relevant .
MINUS { ?T_ex_re rdf:type :unexecutable }}
```
(9)

An inference mechanism based on the classes above considers tasks as relevant for execution if their output is unavailable. However, it does not consider that execution can get obsolete when the data it would generate are not required to achieve a process goal or milestone since it is only considering the next step. An extension of the example above elucidates this: Let us assume writing an email or making a call is done to acquire data that are needed by another task to produce a document. The document thereby marks a process goal. If the document already exists, the tasks of writing an email or making a call are obsolete because the goal for which they would be executed has already been achieved. Since KiPs are goal oriented as they evolve through the achievement of intermediate goals or milestones [10], they would benefit if the inference mechanism considers goal relevance.

5.3 Inferring Goal Relevance

In the following, we regard goals as individuals of tasks and data elements assigned to the class *Process Goal*, denoting a particular purpose for executing or generating them. The exceptionality in this case is that inferring goal relevance results in iterating backward through the chain of process elements, while all process elements (abbreviated as PE) (10)

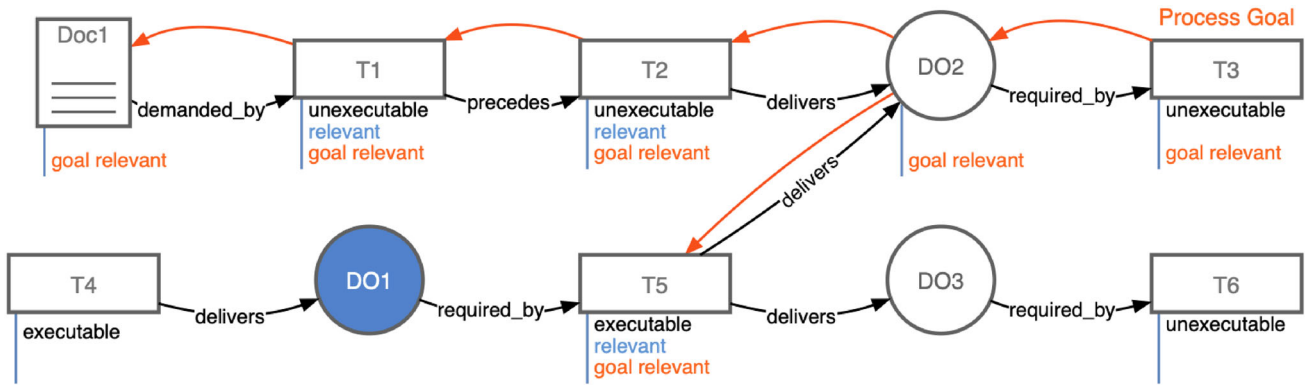

Fig. 6 Determining execution states and relevance states

marked as placeholders (not only Tasks) become goal relevant until the first element without a placeholder marking occurs. Conversely, all elements not included in the loop are not goal relevant since their overall contribution does not support achieving an unfulfilled *Process Goal*.

In order for the inference mechanism to behave as sketched above, we need a starting point, which obviously is an individual marked as Process Goal. This becomes our first goal relevant (abbreviated in formulas with goalRel) process element as defined in formula (11). The interdependency and self-reference forces a reasoner to recursively infer goal relevance (12), which behaves similar to a loop to determine goal-relevance for all contributing process elements. Since loops need a defined termination, any executed task or available data element (not marked as Placeholder) leads to an end of the recursive reasoning procedure.

$$PE \equiv Data \sqcup Task \tag{10}$$

$$PE_{goalRel} \equiv Placeholder \sqcap ProcessGoal \tag{11}$$

$$\begin{aligned} PE_{goalRel} \equiv\ & Placeholder \sqcap (\\ & \exists required\_by.PE_{goalRel} \\ & \sqcup\ \exists delivers.PE_{goalRel} \\ & \sqcup\ \exists demanded\_by.PE_{goalRel} \\ & \sqcup\ \exists produces.PE_{goalRel} \\ & \sqcup\ \exists provides.PE_{goalRel} \\ & \sqcup\ \exists precedes.PE_{goalRel}) \end{aligned} \tag{12}$$

For a better understanding, Fig. 6 shows an abstract example that presents all deduced facts (the Execution States and the Relevance States) as annotations below each process element. According to formula 11, task T3 is defined as *Process Goal*. As a result of the introduced inferencing capabilities, the tasks T1–T6 are categorized into the states *unexecutable*, *executable* and *relevant*. The dataobject DO1 is the only meaningful element, while all other process elements are still *Placeholders*. T1, T2, and T5 are the only tasks that could deliver something beneficial, so they are seen as *relevant* while especially T4 is *irrelevant* as DO1 is already acquired. T4 and T5 are the only *executable* tasks in this example, as required input data is available. Since T3 is defined as *Process Goal*, it becomes the first *goal-relevant* element. Following the orange arrows backward through the graph, all process elements also become *goal relevant* (DO2, T2, T5, T1, Doc1) until we reach the meaningful element DO1. As a result, DO1 and T4 are not stated as *goal-relevant* as well as DO3 and T6, which do not contribute to the only process goal T3. Considering the outcome, the best choice for a process progress is executing T5, which is *executable*, *relevant*, and *goal-relevant* or acquiring Doc1 as a *goal-relevant* and still missing document.

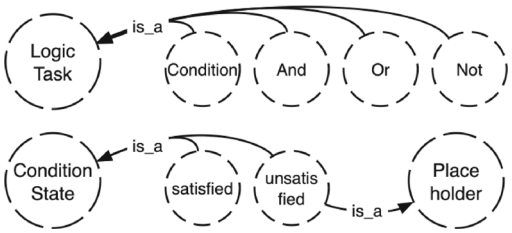

Fig. 7 Logic tasks extension of the base ontology

5.4 Logic Task and Condition States

The introduced mechanisms are sufficient to control the executability of tasks based on the availability of process elements. However, so far we cannot control the executability of tasks by considering a specific process element value. As an example, we can express that we need a meaningful (not Placeholder) attribute *self-affected* of the dataobject *Person* to proceed, but we cannot express whether the value of the Attribute *self-affected* has to be *true* or *false*. For this purpose, we expand our base ontology by *Logic Tasks* (*Condition*, *And*, *Or*, *Not*) and *Condition States* (*satisfied*, *unsatisfied*) as shown in Fig. 7.

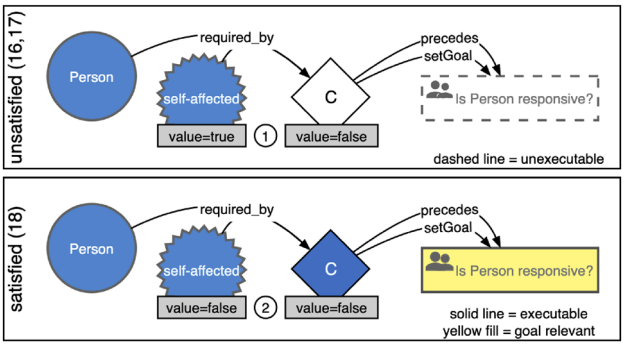

Fig. 8 Example with a condition element

Most important is the new *Condition* type, which has a data property *value*, exactly the same data property, which is also used for *Attributes* to store its item value. As a precondition, a condition element must have precisely one attribute element as input. If the attribute element is a Placeholder or both elements' value is not equal, the condition is considered unsatisfied. If the value of both elements is equal, the condition is satisfied. Since this exceeds the expressiveness of OWL2, it requires the use of the Semantic Web Rule Language (SWRL), which leads us to the following definitions.

$$Condition(?con) \wedge required\_by(?att, ?con) \wedge Placeholder(?att) \rightarrow unsatisfied(?con) \quad (13)$$

$$Condition(?con) \wedge value(?con, ?v1) \wedge required\_by(?att, ?con) \wedge value(?att, ?v2) \wedge unequal(?v1, ?v2) \rightarrow unsatisfied(?con) \quad (14)$$

$$Condition(?con) \wedge value(?con, ?v1) \wedge required\_by(?att, ?con) \wedge value(?att, ?v2) \wedge equal(?v1, ?v2) \rightarrow satisfied(?con) \quad (15)$$

Unlike other process element types, a condition element is never explicitly stated as *Placeholder*. As soon as a condition element is inferred as *unsatisfied*, it automatically becomes a *Placeholder* assignment since the concept *unsatisfied* is also a subclass of the *Placeholder* concept (Fig. 7). Thanks to this, a condition element fulfills all requirements to support the previously introduced mechanisms. The other Logic Tasks (And, Or, Not) serve the purpose to combine several input elements to expand the general expressiveness of process definitions and are discussed in more detail in Sect. 5.6.

Figure 8 illustrates the use of a condition element with the self-affected person example. The diamond-shaped element C is a condition element, which requires an input attribute with the same value. If the values are different ①, the condition is unsatisfied and the condition element also becomes a Placeholder state. If the values are equal ②, the condition is satisfied and the condition element is considered as executed. Through the already known *precedes* relation, the subsequent task can become executable (solid line) or unexecutable (dashed line). The *setGoal* relation will be explained in the following section.

Note: For simplification, we explain the mechanism on an untyped data-property and do not distinguish between different value types (like string, int, date), which is reflected by an abstract comparison expression (unequal, equal) in formula (14, 15).

5.5 Inferring Process Goals

Until now, we have seen process goals as an explicit statement by assigning a process element to the according concept. This could be the result of a predefined process goal, stated in the process definition PD and copied to a process instance PI during instantiation. Alternatively, any process element could be manually defined by a user as a process goal during the process execution. Besides this, we can imagine that a process element is deduced as a process goal through the inference engine. Actually, we want to be able to design a process situation in which a specific process element also becomes a process goal. This expands our Metamodel's expressiveness and serves the specific demands of KiPs for a flexible and goal-oriented process execution [10].

Therefore, we can make use of the condition element, we introduced in chapter 5.4, together with the *setGoal* relation (Fig. 2) to realize this new functionality, which is defined by formula 16.

$$Condition(?con) \wedge satisfied(?con) \wedge setGoal(?con, ?target) \rightarrow ProcessGoal(?target) \quad (16)$$

Figure 8 presents an example of inferring a *Process Goal*. As soon as condition C is satisfied, the task "Is person responsive" becomes executable and a *Process Goal*, expressed with a yellow background color. To better understand, the mechanism to infer process goals is explained in Sect. 6 by an additional example.

5.6 Limitations and Challenges by the OWA

The implementation of an inference mechanism to realize the presented functionality has some specific requirements. At first, all formulas have to be decidable under the open-world assumption (OWA). This is the case since the formulas (1–5; 7–8; 10–12) are valid expressions within the OWL 2 DL language. With the help of the introduced SPARQL-queries (6, 9), the OWA's limitations can be overcome through the interpretation of the query under the CWA perspective. The formulas (13–16) are valid SWRL DL-safe rules, which means that all presented rules are part of the decidable fragment of the first-order predicate logic.

From an academic perspective, this is all we need to know, but we want to utilize our findings from a practical perspective. Fortunately, we can process all the rules with the Pellet reasoner, which in our tests seems to be the only open-source reasoner that is capable of dealing with the inferencing loop caused by the goal relevance (12). It is worth noting that the performance of the Pellet reasoner is not suited for use in a real application at this stage, since the reasoner always infers the complete knowledge store and cannot be restricted to the required subset, which is used to define a specific process. A possible solution could be an incremental reasoning procedure as evaluated by Reyes-Alvarez et al. [21].

However, there is still a challenge with the Logic Tasks (And, Or, Not) as shown in Fig. 7 we have to deal with. For a better understanding, we explain the problem with the *Or* Logic-Task. The desired behavior of an *Or* process element is that if at least one of maybe many input elements is meaningful (not Placeholder), the *Or* element should be meaningful as well. This means we need to assign the *Or* element to the class Placeholder if *ALL* input elements are Placeholder as well. Due to the OWA, this condition cannot be modeled without introducing an appropriate closure. The only way to do this is to define several OR-subclasses, all with a fixed number of input elements, to close OWA's assumption that there could be further (yet unstated) input elements. In combination with specialized SWRL rules for each of these OR subclasses, the Placeholder assignment could technically be inferred. As a result, the process designer would have to choose the right OR-process element, depending on the number of input elements, which again seems to be an impractical way. Basically, this is the same challenge as inferencing the executability of tasks with one difference. While the SPARQL queries defined by formula (6) and (9) only have to deliver an output, the classification of a *Logic Task* as *Placeholder* is required for further inferencing steps. As a result, the assessment of Logic Tasks must be manifested in the knowledge store. This could be realized with the help of SPARQL-queries or with the help of dedicated agents, the option we choose. Such agents analyze the input elements and set or remove the according placeholder-assignment explicitly to realize the Logic Tasks' intended behavior.

5.7 Process Contribution Through Expert Knowledge

The discussed examples are following a process definition based on the concepts, relations, OWL2-, and SWRL-rules given by the base ontology. As an extension to this methodology, generalized expert knowledge, embedded by further rules in the domain ontology, can be used to influence a process execution. Such a rule could define a certain process element as a process goal as soon as some specific preconditions are fulfilled.

$$\begin{aligned} &Process(?proc) \wedge CUC(?cuc) \wedge conscious(?con) \wedge \\ &contains(?proc, ?cuc) \wedge contains(?proc, ?con) \wedge \\ &valueStr(?con, ?val) \wedge swrlb:matches(?val, \text{“}no''\text{)} \\ &\rightarrow ProcessGoal(?cuc) \end{aligned} \tag{17}$$

Rule (17) serves as a simplified example of how expert knowledge can be expressed in a domain ontology. This rule is focused on the process element "Cause: Unconscious" (CUC) and defines any instance of this type as a process goal, as soon as the same process contains an attribute "conscious" with a value equals to "no." Once the CUC element is deduced as a *Process Goal*, the already established mechanisms are used to transfer the *GoalRelevance* (rule 11, 12) to any related process element. In other words, as soon as the ECP comes to conclusion that we have an unconscious person, the process adapts to a new and additional process goal to find out more about the cause. This example will be picked up in the following section and will be explained more in detail.

6 Demonstration on Use Cases

The proposed inference mechanism allows using ontologies to classify unexecuted tasks in data-driven processes in the dimensions executability and relevance while pursuing goals and identifying skippable tasks. Since relevance is determined by the state of output data elements and process goals, users can choose tasks for execution based on the extent to which their execution would contribute to process progress.

The capabilities of the inference mechanism are shown in the following using Protégé 5.2.0[3] and the ontology reasoner Pellet[4] The demonstration uses an extended excerpt from the emergency call procedure, as shown in Fig. 9. For a better reading, the attributes are depicted as rectangles connected to the belonging dataobject, including the value of their data-property *strValue*. Moreover, only the most important relations are shown to provide a reduced view of this example's most relevant aspects.

The upper part of Fig. 9 represents a process instance that is partially executed (blue-filled elements). The lower part of the figure shows the corresponding individuals within the knowledge store with its assigned classes. Additionally, the deduced classes are pointed out, including a reference to the utilized rules.

[3] Protégé is a free-to-use Stanford University ontology editor, available online at: https://protege.stanford.edu.

[4] Pellet is released under Dual Licensing and available online at: https://github.com/stardog-union/pellet.

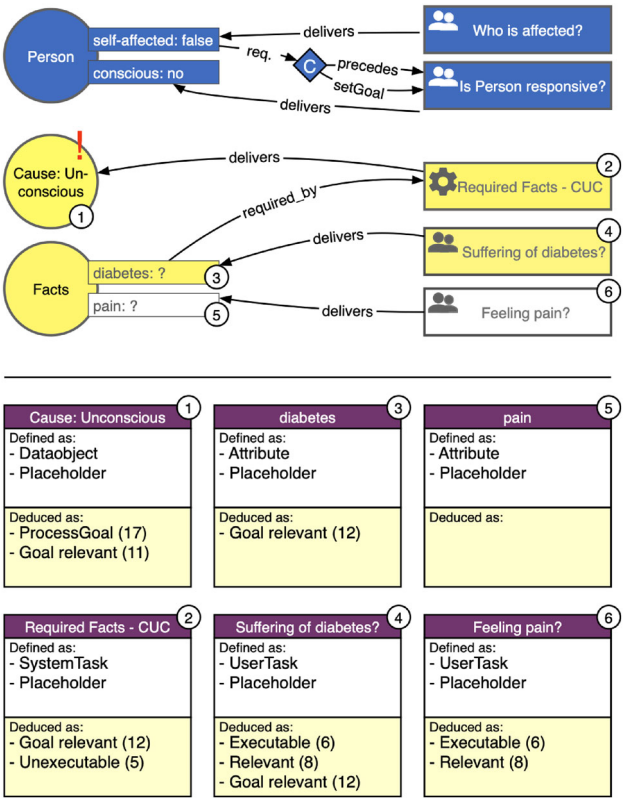

Fig. 9 Example with view to the inference mechanism

The executed part of the PI delivers the facts that the caller is not *self-affected* and the affected person is not *conscious*. Without any direct relation to element ① and with the help of rule (17), the expert knowledge embedded in the domain ontology allows inferring this element as a *ProcessGoal* which makes it *goal relevant* as well. With the help of the defined OWL2- and SWRL-rules, the states of further process elements can be deduced. The system task ② so far is *unexecutable* but becomes *goal-relevant* since its contribution would support another goal-relevant element ①. The same counts for the attribute *diabetes* ③ which is also identified as *goal-relevant*. All tasks which can deliver attribute ③ become goal relevant as well and with a view to task ④, even more states (executable and relevant) can be deduced. Opposed to this, the attribute *pain* ⑤ is not input for② and thus is not seen as goal-relevant. Accordingly, the task ⑥ does not become goal-relevant either and is just identified as executable, since it can just deliver something unknown ⑤ which is not goal relevant. This small example presents a case where, according to the given data, a process adapts to a new process goal, which results in the recalculation of relevance for all process elements.

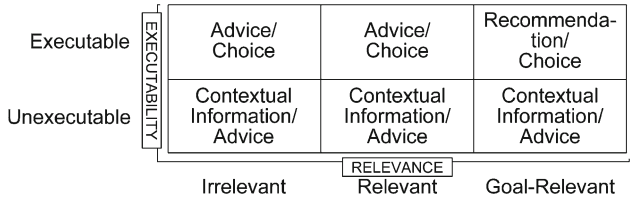

Fig. 10 Possible statements about executability and their use

7 Possible Process Support

The proposed inference mechanism allows inferring whether a task is executable and classifies its execution in terms of relevance. Figure 10 shows a coordinate system that summarizes which statements along the dimensions of executability and relevance are possible. Each field in the coordinate system represents a possible statement and the heading of each field describes what we believe this statement could be used to support KiPs.

Decision-makings in KiPs should be supported by, e. g., recommendations, choices, pieces of advice or contextual information, and knowledge workers should be free to either follow proposed actions or not [1,10,27]. It is obvious that executable and goal-relevant tasks are ideal for knowledge workers to reach their goals as soon as possible while skipping obsolete tasks. Statements about executable tasks on all relevance levels can be seen as choices for knowledge workers to either follow the inferred ideal execution path to achieve goals or deviate from it. If they choose to deviate, the inference mechanism provides them with pieces of advice about which executions could be beneficial if they want to generate data. The advantage of using a reasoner to determine the executability of tasks becomes particularly clear if tasks are unexecutable. Reasoners can then explain why they inferred that tasks are unexecutable, generating lists of unavailable data elements. Since those explanations rely on the triple syntax, they could easily be transformed into human language and displayed as contextual information. Knowledge workers can figure out how they will generate these data elements to turn unexecutable tasks into executable ones. This could lead them to ad hoc planned tasks that exploit existing data elements. For example, a known email address of some person could be used to obtain a missing date of birth attribute of the same person by writing an email and asking for it. Such activities are implicitly contained in process instances and can only be performed when cross-relations between data elements are detected. Such cross-relationships could probably be discovered autonomously by agents, taking explanations of reasoners as an input. Furthermore, they could find appropriate subworkflows to generate unavailable data elements and advise knowledge workers to start them. This would lead to the ability to make implicit activi-

ties explicit and allow proactive support to turn unexecutable tasks into executables as soon as possible.

8 Evaluation

In alignment with the presented examples, the ODD-BP approach was also experimentally evaluated in the domain of emergency call centers. Section 3 introduces the application scenario, which also defines the environment to evaluate our new ODD-BP approach's contributions.

8.1 Goal and Hypothesis

As stated before, the introduced approach aims to support KiPs by enabling a flexible process execution. With its permanent process planning procedure, the execution of a process instance is not limited to a predefined path. The reasoner can deduce process relevant states of each process element at each process step, which leads to our first hypothesis.

– **H1:** The ODD-BP approach supports a flexible process execution that results in higher variability of executed tasks than a standard workflow engine.

Additionally, we expand the semantic process definition with expert knowledge. This expert knowledge is used to adapt a process instance to meet important process goals by inferring the relevance of process elements according to the process instance's currently available data. As a result, AI inferencing can play an active role during the process execution and can support users in decision-making processes, which leads to our second hypothesis.

– **H2:** The ODD-BP approach leads to an improved quality of process outcome compared with a standard workflow engine.

8.2 Preparation of the Experiment

To evaluate the new approach, the presented methodology was realized in a prototype with a client-server architecture. The server is realized as a stateless REST-API web-server written in Java and handles the knowledge store's access using the Apache Jena framework. Furthermore, the server takes advantage of the Pellet reasoner to deduce the process state and relevance for any process element according to the introduced rules. The described gap, caused by the OWA, is closed by implementing a built-in agent, which manually checks and classifies all logic elements regarding their executability. The client side is realized as a webpage, which uses the D3-framework to visualize the process and data graph in any up-to-date browser.

The setup of the system has been done in cooperation with the ECC in the German city Ludwigshafen (Rhine)—called "Integrierte Leitstelle Ludwigshafen" (ILS-LU). In the first step, the ILS-LU's existing ECG was used as a template to design an initial process definition for the evaluation. In a second step, the domain ontology was enriched with medical expert knowledge from the ILS-LU medical head. Next, the process definition was enriched by smart process elements (SPE), regularly connected to other process elements. The purpose of the SPEs is to enhance the ECG and to suggest essential questions, to acquire further and valuable data to achieve a better process result. SPEs can be influenced through the underlying expert knowledge, reflected by rules in the domain ontology, which define these elements as process goals if the available data reaches a predefined state. Finally, the SPEs can also reach a specific state based on further rules and thus reflect a medical assessment, according to the given data.

To compare the process execution and the process results between the established base ECG (bECG) and the new enhanced ECG (eECG), both variants can be performed by the prototypical system. In fact, both variants were realized within a single process definition, using a data element to activate the enhanced capabilities of the eECG in specific cases, described in 8.4. While the bECG offered the same process execution as the original model, the eECG offered an extended and through the SPEs enriched process execution. Each process instance's execution was recorded with all details along with each process interaction (each process step), including the IDs of the process elements, delivered data values, and a precise time stamp.

8.3 System Demonstration

A process instance to guide an emergency call looks in its initial stage as shown at the left box in Fig. 11. All elements with low relevance are hidden and only the most essential process elements are visible. In this Emergency Call Process (ECP), the question tasks (rectangles with speech bubbles) to run the anamnesis are located on the left side. The data (circles) acquired by these questions are collected in the middle, while the disposition options (rectangles with flash) are shown on the right side. They can be executed at any time, independently from the overall state or progress.

The right box of Fig. 11 shows the ECP in a stage where the bECG usually would end and the adaption through an SPE kicks in. ① are the question-tasks following the bECG. ② and ③ are the adaptations proposed by the expert system to acquire additional details. The yellow element ② is the SPE, which represents an aspect, that should be examined more in detail. This element is deduced as a process

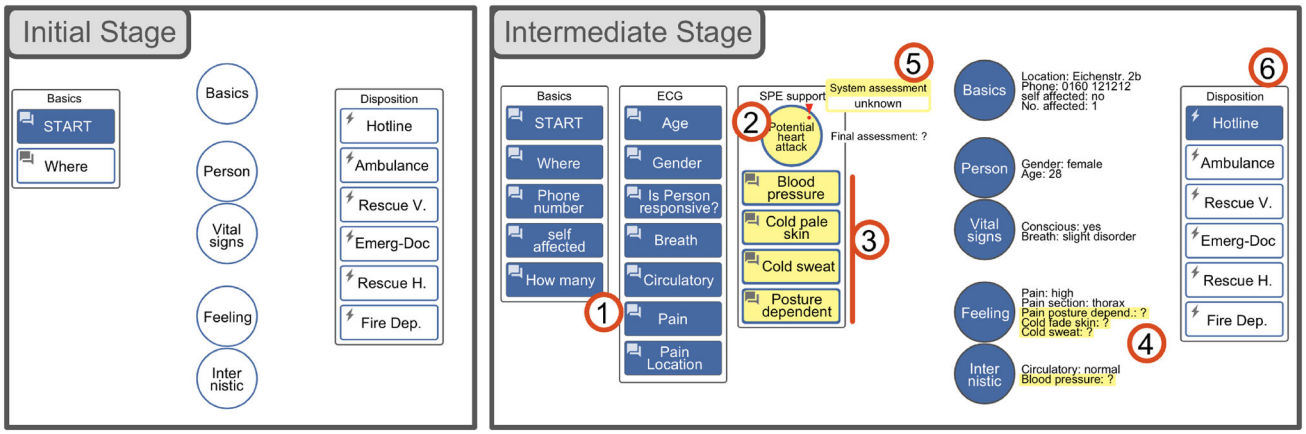

Fig. 11 Emergency Call Process (ECP)—initial and final stage

goal through the underlying domain ontology, considering the current situation's known facts.

Following the same principles as described in Sect. 6, a system task, which remains hidden in the process visualization, has a predefined set of required (yellow highlighted) attributes ④. They all become goal-relevant, which in the following is transferred to the question-tasks ③ that can deliver these attributes. Based on these inference steps, the elements ② and ③ are now more relevant for the process execution and this results in the presentation and recommendation of the formally hidden elements. If a required attribute is already known, the related question is less relevant and will remain hidden to reduce the requested process steps and simplify the overall process visualization. Once all required attributes ④ are available, the expert system will be utilized to deliver a system assessment ⑤ to support and guide the operator who has the freedom to follow or deviate with his own final assessment. Based on the final assessment, specific emergency services ⑥ will be recommended for disposition.

8.4 Experimental Setup

In order to set up the experiment with the operators from ILS-LU, we prepared three real-world EC-cases. For this purpose, the medical expert chose common but complex cases (A, B, C), each with a different underlying story and with different possible disposition results. Each of the three cases with their own background stories can be handled with a different combination of 8 available emergency services. The medical head of the evaluation partner defined the optimal combination of the eight services for each case. Depending on the level of collected information about each case, the operator was expected to decide on a different set of emergency services (a hotline, an ambulance, a rescue vehicle, an emergency doctor, a rescue helicopter, firefighters, 2 specific pieces of advice). All 3 cases have in common that the disposition of the emergency resources may differ with the help of the eECG compared to state-of-the-art bECG. They have also in common that the additional questions result from the situational knowledge, and the domain ontology with its integrated medical expert knowledge is used to adapt the eECG accordingly.

- Case A deals with a situation where someone finds an unconscious person. With the eECG, the operator should be supported in figuring out the reason for the unconsciousness. Within the story, the reason is CO poisoning, which leads to sending out the firefighters aside to the rescue workers.
- Case B raises the assumption of a potential heart attack. With the eECG, the operator should collect indications that the symptoms are caused by a back problem, which leads to redirection to a medical hotline.
- In Case C, a man was hurt in a car accident. While the story starts with a harmless description of the situation, the eECG should support the operator in asking questions to unveil some serious conditions, which leads to sending out an emergency doctor.

The introduced prototypical system and the three modeled cases (A, B, C) build our experimental setup. The three cases were executed as regular emergency calls with professional operators from the evaluation partner ILS-LU, following a predefined fixed story. The operators were only using the prototypical system and each case was performed as a single process instance.

To compare the process execution and the process results between the bECG and eECG, both variants had to be performed for all 3 cases. As a result, each operator executed one case following the established bECG and the subsequent two cases following the new eECG. To obtain results for both variants for all three cases, the order of the cases A,

B, C was mixed between different participating operators. This way, we collected the data that defines the baseline for the established bECG and the data to measure the expected improvements with the eECG.

- To examine the validity of H1, each process step of the process instances is used to examine the different process executions variability.
- To examine the validity of H2, the dispositions between different process executions are used to compare the quality of the process outcome.

Within one week, the experiment was performed with 21 professional emergency operators of the ILS-LU. We had 16 male and 5 female operators with professional experience between 0 and 28 years (average 9). Within approx. 45 minutes, each operator got a short introduction on using the new user interface and performed the three prepared cases (A, B, C). This results in 63 performed and recorded ECP executions. Since the three cases were performed in mixed order, the cases are presented with a letter (case) and number (order) like A1 in the following.

8.5 Experimental Results and Analysis of H1

Besides the fact that flexibility is often described as a significant capability for KiPs, we could not find any established methodology to quantify flexibility for a process. Especially with a view to the ODD-BP approach, we cannot calculate one single number for a whole process since a process is not predefined classically but results from a permanent process planning procedure. Alternatively, we can think about flexibility (Flex) as the number of execution options (ExOpts), which are offered at one specific process step (S).

While flexibility expresses the number of execution-options for a process step, we also need to define a terminology to express the number of the utilization of such flexibility over a set of process executions. For this purpose, we use the term variability (Var) to express the intensity in which the users have taken advantage of the offered flexibility. $Var(S)$ has a range between 1, no variations occurred, and $Flex(S)$ as maximum, when all execution options were used equally over a set of executed processes. With these definitions in mind, we define the following formula:

$$Var_S = \sum_{i=1}^{|ExOpts(S)|} \frac{numOccur(ExOpt_i, S)}{maxOccur(S)} \tag{18}$$

Formula 18 defines the average variability of a certain process step Var_S over a set of process executions.

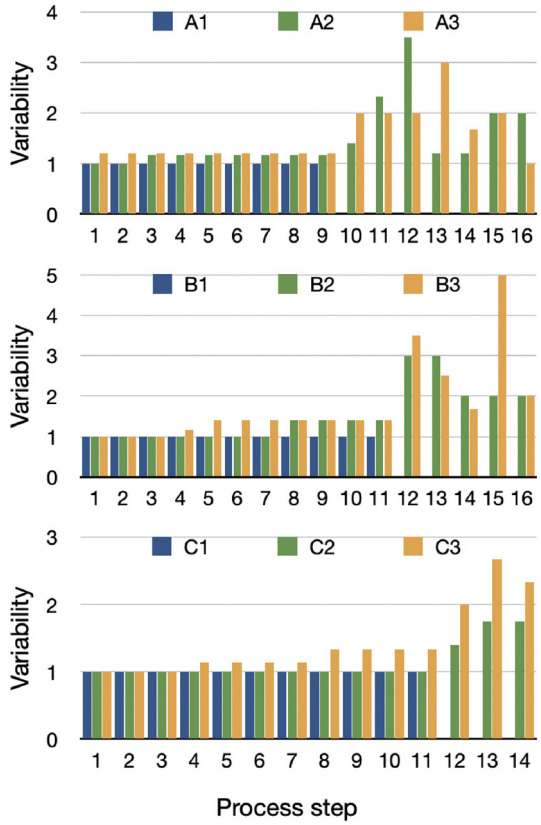

Fig. 12 Measured average variability $Var(S)$ for each process step S

- $|ExOpts(S)|$ returns the number of different execution-options performed at a certain process step S.
- $ExOpt_i$ represents one specific execution option.
- $numOccur(ExOpt_i, S)$ returns the number of executions of an ExOpt at a certain process step S.
- $maxOccur(S)$ returns the number of the most frequent performed ExOpt for a process step S.

Figure 12 presents the measured average variability for all process steps and all cases. Since A1, B1, and C1 were executed according to the established bECG, no flexibility was offered and thus, the measured variability is exactly 1. In the experiments where the cases were executed with the eECG (A2, A3, B2, B3, C2, C3), the operators took advantage of the offered flexibility to a certain degree that results in the measured average variability. It is worth noting that the initial steps (1–9 for case A and 1–11 for case B and C) offered some flexibility in the eECG, but the operators in the majority followed their well-trained path. After these initial steps, the bECG offered no further process support, while in the eECG, the SPEs appeared. Since the SPEs were unknown to the operators, they had no pre-trained path to follow. As a result, the offered flexibility turned into a higher measurable variability.

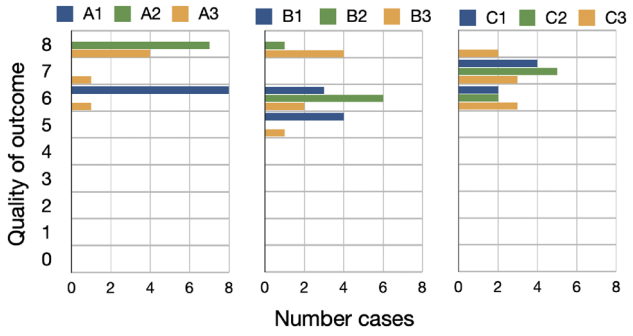

Fig. 13 Process outcome—quality of disposition

The evaluation of the recorded process instances verifies H1 and confirms that the ODD-BP approach supports a flexible process execution compared with the state-of-the-art approach.

8.6 Experimental Results and Analysis of H2

As stated in Sect. 8.4, each case (A, B, C) has its own optimal set of 8 possible emergency services. We calculated the quality of the process outcome by comparing each case's selected emergency services with the optimal service set. Each alignment counts with 1, while any misalignment counts with 0. Added up, a perfect service disposition always has a quality of the outcome of 8, while any lower value indicates an increasing distance to the optimal result.

Figure 13 presents the measured quality of outcome for each of the three cases (A, B, C). Each bar's width (x-axis) indicates how many process instances of a particular type reached a specific quality level (y-axis). Overall, the cases performed at first (A1, B1, C1—blue bars) following the bECG, often reached a less optimal result than the cases (A2, A3, B2, B3, C2, C3). This second group of cases used the eECG with the support of adapted process goals and the guidance of the SPEs and thus reached more often a higher rated process result.

The evaluation of the ranked process outcome supports the second hypothesis H2.

9 Conclusion

This paper describes an approach that combines the process metamodel with domain-relevant knowledge and data about process definitions and situational facts of process instances. As a result, all knowledge that somehow influences a process execution is semantically integrated into one unified knowledge base. We have shown that the approach allows us to infer the executability of tasks and the relevance of process elements utilizing OWL2 and SWRL rules. Thus, a typical workflow engine is not required and can be replaced by a reasoner. Limitations caused by the OWA can be overcome with SPARQL-queries and specialized agents and are subject to further research. As a confirmation of its name, the ODD-BP approach is driven by data and by an ontology.

The new approach was evaluated in an emergency call center, where 21 operators used an enhanced emergency call guideline to execute 63 simulated emergency calls. In this use case, the evaluation verifies a flexible process execution while operating data driven by considering the evolving state of process knowledge. Additionally, we can confirm that the process results' quality can be significantly improved through an AI contribution based on an expert system embedded into the domain ontology.

With our ongoing evaluation of the presented use case, we will examine further capabilities of ODD-BP like its adaptability to deal with the emergent, non-repeatable, and unpredictable characteristics of such KiPs. Besides these more functional aspects, the user's acceptance of such an AI-supported process execution is essential, especially when the amount of knowledge in the embedded expert system increases and the behavior appears unpredictable to the user. An ontology-driven approach like ODD-BP offers the base to explain its behavior (XAI), which can be utilized to gain user acceptance, an aspect we also intend to address with our future work.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Abecker A, Bernardi A, van Elst L, Lauer A, Maus H, Schwarz S, Sintek, M (2001) FRODO: a framework for distributed organizational memories: DFKI Document D-01-01. DFKI GmbH
2. Bechhofer S (2020) OWL reasoning examples. http://owl.man.ac.uk/2003/why/latest/
3. Betz S, Klink S, Koschmider A, Oberweis (2006) A automatic user support for business process modeling. In: Proceedings of the workshop on semantics for business process management, pp 1–12
4. Bhattacharya K, Gerede C, Hull R, Liu R, Su J (2007) Towards formal analysis of artifact-centric business process models. In: Lecture notes in computer science (including subseries Lecture Notes

in Artificial Intelligence and Lecture Notes in Bioinformatics), vol 4714, pp 288–304
5. Bhattacharya K, Hull R, Su J (2009) A data-centric design methodology for business processes. In: Handbook of research on business process modeling, pp 503–531
6. Bock C, Fokoue A, Haase P, Hoekstra R, Horrocks I, Ruttenberg A, Sattler U, Smith M (2012) OWL 2 web ontology language structural specification and functional-style syntax (second edition): W3C recommendation . https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/
7. Cohn D, Hull R (2009) Business artifacts: a data-centric approach to modeling business operations and processes. IEEE Data Eng Bull 32:3–9
8. Davenport TH (2005) Thinking for a living: how to get better performances and results from knowledge workers. Harvard Business School Press, Boston
9. Dengel A (2012) Semantische Technologien: Grundlagen—Konzepte—Anwendungen. Spektrum Akademischer Verlag
10. Di Ciccio C, Marrella A, Russo A (2015) Knowledge-intensive processes: characteristics, requirements and analysis of contemporary approaches. J Data Semant 4(1):29–57. https://doi.org/10.1007/s13740-014-0038-4
11. Fan S, Hua Z, Storey VC, Zhao JL (2016) A process ontology based approach to easing semantic ambiguity in business process modeling. Data Knowl Eng 102:57–77. https://doi.org/10.1016/j.datak.2016.01.001
12. Gaur A, Richariya V (2008) A layered approach for intrusion detection using meta-modeling with classification techniques. Int J Comput Technol Electron Eng (IJCTEE) 1(2):161–168
13. Giordano L, Theseider Dupré D (2018) Enriched modeling and reasoning on business processes with ontologies and answer set programming. Lect Notes Bus Inf Process 329:71–88. https://doi.org/10.1007/978-3-319-98651-7_5
14. Heinrich B, Bewernik M, Henneberger M, Krammer A, Lautenbacher F (2008) SEMPA—Ein Ansatz des Semantischen Prozessmanagements zur Planung von Prozessmodellen. Wirtschaftsinformatik 50(6):445–460
15. Horridge M (2011) A practical guide to building OWL ontologies using Protégé 4 and CO-ODE tools edition 1.3 . http://mowl-power.cs.man.ac.uk/protegeowltutorial/resources/ProtegeOWLTutorialP4_v1_3.pdf
16. Hull R, Damaggio E, Fournier F, Gupta M, Nigam A, Sukaviriya P, Vaculin R (2010) Introducing the guard-stage-milestone approach for specifying business entity lifecycles (257593), pp 1–22
17. Ly LT, Rinderle-Ma S, Göser K, Dadam P (2012) On enabling integrated process compliance with semantic constraints in process management systems: requirements, challenges, solutions. Inf Syst Front 14(2):195–219. https://doi.org/10.1007/s10796-009-9185-9
18. Maletzki C, Rietzke EGLBRKN (2019) Utilizing ontology-based reasoning to support the execution of knowledge-intensive processes. In: BPM 2019—AI4BPM, p 12
19. Neto AT, Bussamra FLdS, e Silva HAdC (2014) A new metamodel for reinforced panels under compressive loads and its application to the fuselage conception. Latin Am J Solids Struct 11(2):223–244. https://doi.org/10.1590/S1679-78252014000200005
20. Object Management Group O, Marin MA (2016) Introduction to the Case Management Model and Notation (CMMN). http://www.omg.org/spec/CMMN/1.1
21. Reyes-Alvarez L, Molina-Morales D, Hidalgo-Delgado Y, Del Mar Roldán-García M, Aldana-Montes JF (2014) Exploring incremental reasoning approaches based on module extraction. In: CEUR Workshop Proceedings, vol 1219, pp 1–12. https://doi.org/10.13140/2.1.2000.8322
22. Rietzke E, Bergmann R, Kuhn N (2017) Adaptive business process visualization for a data and constraint-based workflow approach. In: CEUR Workshop Proceedings: LWDA, vol 1917
23. Rietzke E, Bergmann R, Kuhn N (2018) Semantically-oriented business process visualization for a data and constraint-based workflow approach. In: Teniente E, Weidlich M (eds) Business process management workshops. Lecture notes in business information processing, vol 308. Springer, Cham, pp 142–150. https://doi.org/10.1007/978-3-319-74030-0
24. Rietzke E, Bergmann R, Kuhn N (2019) ODD-BP—an ontology- and data-driven business process model. In: Lernen, Wissen, Daten, Analysen—LWDA, p 12
25. Rosemann M, Zur Muehlen M (1998) Evaluation of workflow management systems—a meta model approach. Australas J Inf Syst 6(1):1–20. https://doi.org/10.3127/ajis.v6i1.322
26. Thomas O, Fellmann M (2009) Semantic process modeling—design and implementation of an ontology-based representation of business processes. Bus Inf Syst Eng 1(6):438–451. https://doi.org/10.1007/s11576-009-0201-y
27. Vaculin R, Hull R, Heath T, Cochran C, Nigam A, Sukaviriya P (2011) Declarative business artifact centric modeling of decision and knowledge intensive business processes. In: 15th IEEE International EDOC Conference Workshops. IEEE Computer Society, Los Alamitos, CA, pp 151–160. https://doi.org/10.1109/EDOC.2011.36
28. Workflow Management Coalition (1999) Workflow Management Coalition Terminology & Glossary. https://www.wfmc.org/docs/TC-1011_term_glossary_v3.pdf