

A Logic-Based Method for Business Process Knowledge Base Management¹

Antonio De Nicola^α, Michele Missikoff^α, Maurizio Proietti^α, Fabrizio Smith^{α,β}

^α IASI-CNR, Viale Manzoni 30, 00185, Rome, Italy

^β DIEI, Università degli Studi de L'Aquila, Italy

{antonio.denicola, michele.missikoff, maurizio.proietti,
fabrizio.smith}@iasi.cnr.it

Abstract. In this paper we present the BPAL platform aimed at the management of Business Process Knowledge Bases. It includes a logic-based language for BP modeling and a reasoning mechanism providing support to BP designers in several services. Firstly, the definition of a BP meta-model (MM) consisting of a set of rules that guide the BP designers in their work. Secondly, given a BP, the BPAL platform allows for the automatic verification of the compliance of a given BP w.r.t. the defined MM. Finally, the execution semantics of a BP is given in term of its instances (referred to as *traces*) to provide two basic services; checking if an execution of a BP has been carried out in accordance with the corresponding definition (process log analysis); simulating possible executions by automatic traces generation. The proposed platform is open since the meta-model can be easily modified to codify different classes of BP, to comply with specific needs of an enterprise.

Keywords: business process, knowledge bases, modeling language, Horn logic, BPAL.

1 Introduction

Business Process (BP) management is constantly gaining popularity in various industrial sectors, especially in medium to large enterprises, and in the public administration. BP modeling is a complex human activity, requiring a special competence and, typically, the use of a BP design tool. Many tools (e.g., Intalio BPMS Designer, Tibco Business Studio), today available on the market (open source or free of charge) support the design of BP diagrams and the production of XML-based representations, such as BPEL or XPD. Many of these tools, beside the generation of XML code, are able to provide, additional services, such as some forms of verification and the simulation of the designed processes. The availability of the mentioned tools has further pushed the diffusion of several languages (e.g., BPMN

¹ This work is partially supported by the Tocai Project (<http://www.dis.uniroma1.it/~tocai/>), funded by the FIRB Programme of the Italian Ministry of University and Research (MIUR).

[6], EPC [14]) used both in the academic and in the industrial realities. However, the use of these languages still present a number of drawbacks caused by the lack of a formal semantics, that leads to possible ambiguities in the defined diagrams, and of a guiding method to support the designer in producing high quality BP diagrams. Furthermore, the storage and retrieval of BPs (and their fragment) is managed in a primitive way, based on XML files or internal (opaque) representations.

In this paper we propose the BPAL (Business Process Abstract Language) [7] platform, aimed at the construction, management, and querying of a Business Process Knowledge Bases (BPKB), with a logic-based representation. The proposed approach provides: (i) a formal semantics to BPs, (ii) guidelines for BP modelers in building better quality BPs, (iii) a query language. In this paper we elaborate the first two points, while the third point, i.e. the query services, is currently under investigation.

In essence, here we illustrate the possibility to: (i) use BPAL to build a meta-model that specifies the rules to build a well-formed BP, (ii) model a BP with BPAL sentences (BPAL BP Schema); (iii) use the BPAL method to verify business process schema well-formedness with respect to the given meta-model; (iv) verify if a given process trace, i.e., the actual execution of a BP, is compliant with a well-formed BP Schema; (v) simulate a BP execution by generating all the possible traces (when finite).

The BPAL platform is characterized by both a formal foundation and a high level of practical usability. The formal foundation is rooted in the logic-based approach of the BPAL notation. The practical usability is guaranteed by the fact that BPAL platform has not been conceived as an alternative to existing BP tools but, conversely, it intends to be associated to the existing BP modeling tools enhancing their functionalities.

The rest of this paper is organized as follows. In Section 2 some relevant related works are presented. The BPAL language for business process modeling and verification is described in Section 3. Section 4 presents the BPAL meta-model and in Section 5 the execution semantics (in term of *execution traces*) of a BPAL BP schema is described. In Section 6 an overview of the BPAL platform, consisting of the well-formedness verification service, the execution log analysis, and traces generation service, is presented. Finally, conclusions in Section 7 end the paper.

2 Related Works

In the literature, much attention is given to BP modeling, since its application to the management of complex processes and systems [9] is an important issue in business organizations. It appears increasingly evident that a good support to BP management requires reliable BP modeling methods and tools. Such reliability can be achieved only if the adopted method is based on formal foundations. In this perspective, our work is related to the formal BP languages for the specification, the verification, and analysis of business processes.

Formal semantics of process modeling languages is usually defined in terms of a mapping to Petri nets [1]. Petri nets represent a powerful formal paradigm to support

automatic analysis and verification of BPs within a procedural approach. However, seen the inherent operational nature of Petri Nets, they are not well suited as a foundation for BPKBs. Furthermore, by using Petri Nets, it is difficult to provide a “meta-level” that can be used to guide and constrain business process modeling.

A different philosophy is represented by a declarative, logic-based approach [2], [3]. A logical approach appears more suited to build a BPKB, with the possibility to manipulate, query, retrieve, compose complex BP diagrams. There are proposals where a process is modeled by a set of *constraints* (business rules) that must be satisfied during execution: these proposals provide a partial representation of a BP that overlooks the procedural view, i.e., the control flow among activities. [2] proposes ConDec, a declarative flow language to define process models that can be represented as conjunction of Linear Temporal Logic formulae. This approach allows to verify properties by using model checking techniques. [3] proposes a verification method based on Abductive Logic Programming (ALP) and, in particular, the SCIFF framework [4], that is an ALP rule-based language and a family of proof procedures for specification and verification of event-based systems.

PSL (Process Specification Language) [5], is a language to support the exchange of process information among systems. A PSL ontology is organized into PSL-CORE and a partially ordered set of extensions. Axioms are first-order sentences written in the Knowledge Interchange Format (KIF). The PSL-CORE axiomatizes a set of intuitive semantic primitives (e.g., *activities*, *activity occurrences*, *time points*, and *objects*) allowing to describe the fundamental concepts of a process. The expressive power of PSL is increased by a set of extensions that introduce new terminology and the logical formalization to express information involving concepts that are not explicitly specified in PSL-Core.

[2], [3], [5] are based on rigorous mathematical foundations but they propose a paradigm shift from traditional process modeling approaches that is difficult to be understood and, consequently, to be accepted by business people. Furthermore, their focus is more on the (partial) representation of BPs aiming at guaranteeing certain formal properties, without a clear aim at the structuring and management of a BPKB. The BPAL framework is positioned among the logic-based languages but with a full capability of representing the control flow of a BP; furthermore, it aims at complementing a typical BP management tool to provide additional services. In essence, with respect to existing logic-based proposals, it is characterized by: (i) enhanced adoptability, since we propose a progressive approach where a business expert can start with the (commercial) tool and notation of his/her choice and then enrich its functionalities with BPAL; (ii) the possibility of specifying a meta-model that imposes quality characteristics to all the produced BP diagrams; (iii) the possibility to automatically verifying the well-formedness of the modeled BP diagrams; (iv) the possibility to automatically verify that a BP execution (i.e., a BP trace) has been achieved in accordance with the BP specification; finally (v) the possibility of automatically generate all the possible traces² of a given BP.

² Please note that in this work we address acyclic diagrams admitting a finite set of finite traces.

3 The BPAL Approach for BP Modeling and Verification

In this section we present the BPAL modeling language. BPAL is a rule-based formalism (grounded in Horn Logic), that provides an integrated support to the building of a BPKB, characterized by three levels: (i) the meta-level, where we define the meta-model, establishing the rules for building well-formed BPs; (ii) the schema level, where we define the BP schemas, in accordance with the given meta-model; (iii) the ground level, where we represent the BP instances, i.e., the traces that are produced by the execution of a BP. For the formalization of BPAL we use standard notions of first order logic and logic programming [8].

3.1 The BPAL Language

In this paper, for sake of brevity, we focus on a core subset of BP modeling constructs. These constructs are common to the most used and widely accepted BP languages (e.g., BPMN, UML activity diagrams, EPC) and, in particular, such a core is based on BPMN 2.0 specification [6].

From a formal point of view, the BPAL language consists of two syntactic categories: (i) a set *Const* of BPAL *constants* denoting entities to be used in the specification of a business process schema (e.g., business activities, events, and gateways) and (ii) a set *Pred* of *predicates* denoting relationships among BPAL entities (e.g., *activity(Invoicing)*). In particular, there are two kinds of predicates in BPAL: *unary* and *relational* predicates. Unary predicates specify the types of the entities of a business process schema while relational predicates describe the sequencing of *flow elements* in all possible executions of the process (e.g., *seq(Invoicing, PayingInvoice)*).

Finally, a BPAL business process schema (BP) is specified by a set of ground *facts* (i.e., atomic formulas) of the form $p(C_1, \dots, C_n)$, where $p \in Pred$ and $C_1, \dots, C_n \in Const$.

3.2 BPAL Entities

The types of the entities of a business process schema are represented by the following set of predicates:

- *flow_el(_el)*: *_el* is a *flow element*, that is, any atomic component appearing in the control flow. A flow element is either an activity or an event or a gateway;
- *activity(_act)*: *_act* is a *business activity*, the key element of the business process;
- *event(_ev)*: *_ev* is an *event* that occurs during the process execution. An event is of one of the following three types: (i) a *start* event, which starts the business process, (ii) an *intermediate* event, and (iii) an *end* event, which ends the business process. These three types of events are specified by the three predicates *start_ev(_start)*, *end_ev(_end)*, and *int_ev(_int)*;
- *gateway(_gat)*: *_gat* is a *gateway*. A gateway is either a *branch* or a *merge point*, whose types are specified by the predicates *branch_pt(_gat)* and *mrg_pt(_gat)*, respectively. A branch (or merge) point can be either a *parallel*, or an *inclusive*, or

an *exclusive* branch (or merge) point. Each type of branch or merge point is specified by a corresponding unary predicate.

3.3 Relational Predicates

BPAL provides predicates to model *parallel* (i.e., AND), *exclusive* (i.e., XOR), and *inclusive* (i.e., OR) *branching/merging* of the control flow. They are:

- $par_branch(gat, el1, el2)^3$: gat is a *parallel branch* point (i.e., a *parallel fork*) from which the business process branches to two sub-processes started by $el1$ and $el2$. The two sub-processes started by $el1$ and $el2$ are executed in *parallel*;
- $par_mrg(el1, el2, gat)$: gat is a *parallel merge* point (i.e., a *join*) where the two sub-processes ended by $el1$ and $el2$ are synchronized, that is, both sub-processes must be completed in order to proceed;
- $inc_dec^4(gat, el1, el2)$: gat is an *inclusive decision* point from which the business process branches to two sub-processes started by $el1$ and $el2$. At least one of the sub-processes started by $el1$ and $el2$ is executed;
- $inc_mrg(el1, el2, gat)$: gat is an *inclusive merge* point. At least one of the two sub-processes ended by $el1$ and $el2$ must be completed in order to proceed;
- $exc_dec(gat, el1, el2)$: gat is an *exclusive decision* point from which the business process branches to two sub-processes started by $el1$ and $el2$. Exactly one of the sub-processes started by $el1$ and $el2$ is executed;
- $exc_mrg(el1, el2, gat)$: gat is an *exclusive merge* point. Exactly one of the two sub-processes ended by $el1$ and $el2$ must be completed in order to proceed;
- $seq(el1, el2)$: the flow element $el1$ is immediately followed by $el2$.

To better present the BPAL approach, we briefly introduce a fragment of an *eProcurement* process that will be used as a running example throughout the rest of the paper. An ACME supplier company receives a purchase order from a buyer and sends back an invoice. The buyer receives the invoice and makes the payment to the bank. In the meanwhile, the supplier prepares a gift for the buyer if she/he is classified as *golden client*, otherwise he prepares a brochure. After receiving the payment clearance from the bank, the supplier sends the goods to the buyer.

The Figure 1 reports a BPMN diagram that illustrates the fragment of the *eProcurement* process from the supplier perspective. The same process is reported in Table 1 encoded as a BPAL BP Schema.

³ Note that we represent only binary branches, while they are n-ary in the general case. This limitation is made for presentation purposes and can be easily removed.

⁴ Note that *inclusive* and *exclusive* gateways, in their general formulation, are associated with a condition. For instance, exc_dec tests a condition to select the path where the process flow will continue.

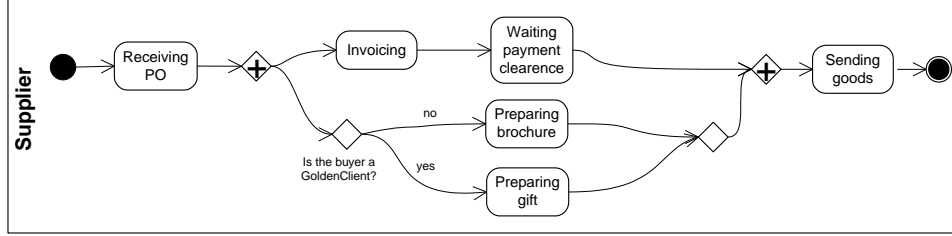


Fig. 1. BPMN specification of a fragment of an eProcurement example

<i>start_ev(Start)</i>	<i>end_ev(End)</i>
<i>activity(ReceivingPO)</i>	<i>seq(Start,ReceivingPO)</i>
<i>activity(Invoicing)</i>	<i>seq(ReceivingPO,Gat1)</i>
<i>activity(WaitingPaymentClearance)</i>	<i>seq(Invoicing,WaitingPaymentClearance)</i>
<i>activity(PreparingGift)</i>	<i>seq(Gat2,SendingGoods)</i>
<i>activity(PreparingBrochure)</i>	<i>seq(SendingGoods,End)</i>
<i>activity(SendingGoods)</i>	<i>par_branch(Gat1,Invoicing,Gat3)</i>
<i>par_branch_pt(Gat1)</i>	<i>par_mrg(WaitingPaymentClearance,Gat4,Gat2)</i>
<i>par_mrg_pt(Gat2)</i>	<i>exc_dec(Gat3,PreparingBrochure,PreparingGift)</i>
<i>exc_dec_pt(Gat3)</i>	<i>exc_mrg(PreparingBrochure,PreparingGift,Gat4)</i>
<i>exc_mrg_pt(Gat4)</i>	

Table 1. BPAL BPS of the eProcurement example

4 BPAL Meta-Model

The first service provided by BPAL is the possibility of defining a BP meta-model that will represent a guiding framework for the BP designer. In this paper, as an example, we show how to impose structured BPs by means of a BPAL meta-model. According to [10], a **strictly structured BP** can be defined as follows: it consists of m sequential *blocks*, $T_1 \dots T_m$. Each block T_i is either elementary, i.e., it is an activity, or complex. A complex block i) starts with a branch node (a parallel, inclusive or exclusive gateway) that is associated with exactly one merge node of the same kind that ends the block, *ii*) each path in the workflow graph originating in a branch node leads to its corresponding merge node and consists of n sequential blocks (simple or complex). It is worth noting that removing the structured assumption leads to several weaknesses [11]. Among them, error patterns [12] such as deadlocks, livelocks and dead activities cannot manifest in a structured BPS.

The presence of a meta-model allows us to automatically prove the first fundamental property: the fact that a BPAL process schema has been built in the respect of the meta-model guidelines, i.e., it is syntactically correct. We will refer to such a property as *well-formedness*. We formalize this notion by means of a set of

rules (i.e., a first order logic theory) MM consisting of three sets of meta-rules: (1) a set K of *schema constraints* (in the form of first order formulas), (2) a set F of *process composition rules* (in the form of Horn clauses) and (3) a set I of *inclusion axioms* among the BPAL entities. All formulas in MM are universally quantified in front and, for sake of simplicity, we omit to write those quantifiers explicitly.

The set K of schema constraints (Table 2) consists of three subsets: (i) the *domain constraints*, (ii) the *type constraints*, and (iii) the *uniqueness constraints*.

<i>Schema constraint</i>	<i>Example</i>
Domain constraints are formulas expressing the relationships among BPAL unary predicates.	A flow element cannot be an activity and an event at the same time. $activity(x) \rightarrow \neg event(x)$
Type constraints are rules specifying the types of the arguments of relational predicates.	A parallel branch is defined among a parallel branch point and two flow elements. $par_branch(x,l,r) \rightarrow par_branch_pt(x) \wedge flow_el(l) \wedge flow_el(r)$
<p>Uniqueness Constraints are rules expressing that the precedence relations between flow elements are specified in an unambiguous way:</p> <p><i>branching uniqueness constraints</i> asserting that every (parallel, inclusive, exclusive) branching point has exactly one pair of successors.</p> <p><i>merging uniqueness constraints</i> asserting that every merge point has exactly one pair of predecessors.</p> <p><i>sequence uniqueness constraints</i> asserting that, by the <i>seq</i> predicate, we can specify at most one successor and at most one predecessor of any flow element.</p>	<p>Example of sequence uniqueness constraint:</p> $seq(x,y) \wedge seq(x,z) \rightarrow y=z$ $seq(x,z) \wedge seq(y,z) \rightarrow x=y$

Table 2. BPAL schema constraints and supporting examples

The set F of *process composition rules* provides the guidelines for building a well-formed BP. Then, in formal terms, it is possible to verify if a process respects such rules by means of a predicate $wf_proc(s,e)$ which holds if the business process started by the event s and ended by the event e is *well-formed*. In Table 3, some rules are reported that inductively define what is a well-formed process (*wf-proc*) by means of the notion of sub-process and its well-formedness (*wf_sub_proc*).

The set I of *inclusion axioms* (e.g., $event(x) \rightarrow flow_el(x)$) defines a taxonomy among the BPAL entities, as informally described in Section 3.2

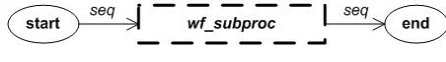
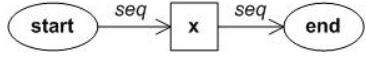

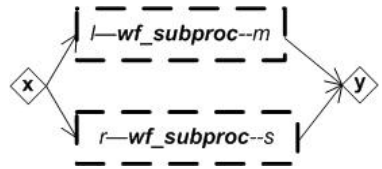
<i>Process composition rule</i>	<i>Intuitive Diagram</i>
F1. A business process schema is <i>well-formed</i> if (i) it is started by a start event s , (ii) it is ended by an end event e , and (iii) the sub-process from s to e is a well-formed sub-process constructed according to rules F2-F6: $(start_ev(s) \wedge wf_sub_proc(s,e) \wedge end_ev(e)) \rightarrow wf_proc(s,e)$	A well-formed process: 
F2. Any activity/event or sequence of two activities/events is a well-formed sub-process: $activity(x) \rightarrow wf_sub_proc(x,x)$ $int_ev(x) \rightarrow wf_sub_proc(x,x)$ $seq(x,y) \rightarrow wf_sub_proc(x,y)$	So, the simplest well-formed process is graphically represented as: 
F3. A sub-process is well-formed if it can be decomposed into a concatenation of two <i>well-formed</i> sub-processes: $wf_sub_proc(x,y) \wedge wf_sub_proc(y,z) \rightarrow wf_sub_proc(x,z)$	A well-formed sub-process: 
F4. A sub-process started by a branch point x and ended by a merge point y is well-formed if (i) x and y are of the same type, and (ii) both branches contain two well-formed sub-processes ⁵ : $par_branch(x,l,r) \wedge wf_sub_proc(l,m) \wedge wf_sub_proc(r,s) \wedge par_mrg(m,s,y) \rightarrow wf_sub_proc(x,y)$	A well-formed sub-process including merge and branch points: 

Table 3. BPAL process composition rules and supporting diagrammatic description

We are now ready to give a definition of the well-formedness of a BP schema B . We say that B is *well-formed* if:

- (i) every schema constraint C in K can be inferred from $B \cup F \cup I$, and
- (ii) for every start event S and end event E , $wf_process(S,E)$ can be inferred from $B \cup F \cup I$.

⁵ The rules F5 and F6 defining the predicate $wf_sub_process(x,y)$ in the cases where x is an inclusive or an exclusive decision gateway are similar and are omitted.

5 BPAL Execution Traces

An execution of a business process is a sequence of instances of *activities* (or *events*) called *steps*. Steps are denoted by constants taken from a set *Step* disjoint from *Const*. Thus, a possible execution of a business process is a sequence $[s_1, s_2, \dots, s_n]$, where $s_1, s_2, \dots, s_n \in \text{Step}$, called a *trace*. The *instance* relation between steps and activities (or events) is specified by a binary predicate $\text{inst}(\text{step}, \text{activity})$. For example, $\text{inst}(RPO1, \text{ReceivingPO})$ states that the step *RPO1* is an activity instance of *ReceivingPO*.

A trace is *correct* w.r.t. a well-formed business process schema **B** if it is conformant to **B** according to the intended semantics of the BPAL relational predicates (as informally described in Section 3.3). Below we present a formal definition of the notion of a correct trace. Let us first give some examples by referring to the example in Figure 1. Below we list two correct traces of the business process schema corresponding to the above BPMN specification:

- $[s, r, i, pG, w, sG, e]$
- $[s, r, i, w, pB, sG, e]$

where $\text{inst}(s, \text{Start})$, $\text{inst}(r, \text{ReceivingPO})$, $\text{inst}(i, \text{Invoicing})$, $\text{inst}(w, \text{WaitingPayment Clearance})$, $\text{inst}(pG, \text{PreparingGift})$, $\text{inst}(pB, \text{PreparingBrochure})$, $\text{inst}(sG, \text{Sending Goods})$, $\text{inst}(e, \text{End})$.

Note that the sub-traces $[i, pG, w]$ of the first trace and $[i, w, pB]$ of the second trace are the interleaving of the sub-trace $[i, w]$ with the two branches going out from the exclusive branch point.

We now introduce a predicate $\text{trace}(t)$, which holds if t is a correct trace, with respect to a BP, of the form $[s_1, s_2, \dots, s_n]$, where s_1 is an instance of a start event and s_n is an instance of an end event. The predicate $\text{trace}(t)$ is defined by a set **T** of rules (in the form of Horn clauses), called *trace rules*. These rules have a double nature, since they can be used to check correctness but also for generating correct traces. Each trace rule corresponds to a *process composition rule*, for lack of space, here we list only the trace rules corresponding to the composition rules presented in Section 4.2. The trace axioms are defined by induction on the length of the trace t .

T1. A sequence $[s1, \dots, e1]$ of steps is a *correct trace* if: (i) $s1$ is an instance of a start event, (ii) $e1$ is an instance of an end event, and (iii) $[s1, \dots, e1]$ is a correct *sub-trace* from $s1$ to $e1$ constructed according to the sets of rules T2-T6:

$$\text{start_ev}(s) \wedge \text{inst}(s1, s) \wedge \text{sub_trace}(s1, t, e1) \wedge \text{end_ev}(e) \wedge \text{inst}(e1, e) \rightarrow \text{trace}(t)$$

T2. Any instance of an activity/event or a sequence of instances of activities/events is a correct sub-trace.

$$\text{inst}(x1, x) \wedge \text{activity}(x) \rightarrow \text{sub_trace}(x1, [x1], x1)$$

$$\text{inst}(x1, x) \wedge \text{int_ev}(x) \rightarrow \text{sub_trace}(x1, [x1], x1)$$

$$\text{inst}(x1, x) \wedge \text{inst}(y1, y) \wedge \text{seq}(x, y) \wedge \text{act_or_ev_seq}([x1, y1], t) \rightarrow \text{sub_trace}(x1, t, y1)$$

where the predicate $\text{act_or_ev_seq}([x1, y1], t)$ holds iff t is the sequence obtained from $[x1, y1]$ by deleting the steps which are not instances of activities or events.

T3. A trace is correct if it can be decomposed into a concatenation of two correct sub-traces:

$$sub_trace(xl,tl,y1) \wedge sub_trace(y1,t2,zl) \wedge concatenation(t1,t2,t) \rightarrow sub_trace(xl,t,zl)$$

where the concatenation of $[x1,...,xm]$ and $[y1,y2,...,yn]$ is $[x1,...,xm,y2,...,yn]$ if $xm = y1$ and $[x1,...,xm,y1,y2,...,yn]$ otherwise.

T4. In the case where $x1$ is an instance of a parallel branch point, the correctness of a sub-trace t from $x1$ to zl is defined by the following rule⁶:

$$\begin{aligned} &inst(x1,x) \wedge inst(l1,l) \wedge inst(r1,r) \wedge par_branch(x,l,r) \wedge inst(m1,m) \wedge sub_trace(l1,t1,m1) \\ &\wedge inst(s1,s) \wedge sub_trace(r1,t2,s1) \wedge inst(y1,y), par_mrg(m,s,y), interleaving(t1,t2,t) \\ &\rightarrow sub_trace(x1,t,y1) \end{aligned}$$

where the predicate *interleaving*($t1,t2,t$) holds iff t is a sequence such that: (i) the elements of t are the elements of $t2$ together with the elements of $t2$ and (ii) for $i=(1,2)$ x precedes y in ti iff x precedes y in t .

Table 4. BPAL Trace rules

We say that a trace t is *correct* w.r.t. a BPAL BP schema B if $trace(t)$ can be inferred from $B \cup T$.

6 The BPAL Platform

In this section we briefly present the logical architecture of the BPAL platform with the key services: (1) verification of the well-formedness of a BPS, (2) log analysis, and (3) trace generation.

In the BPAL platform, the service for verifying the well-formedness of a BP is performed as depicted in Figure 2. A BPMN graphical specification of a given BP-1 business process is exported from the existing tool as XPDL file and translated into a set of BPAL ground facts by means of the service *XPDL2BPAL*, thereby producing the BP schema B . Then B is passed to the reasoning engine as a logic program together with the meta-model MM , i.e., the set F of composition rules, the set K of constraints, and the set I of inclusion rules. Please note that $B \cup F \cup I$ is a set of Horn clauses and, therefore, it is translated to a DATALOG program in a straightforward way. The schema constraints in K are translated to DATALOG queries too. The reasoning engine, aimed at inferring the well-formedness of B , is implemented by using the XSB logic programming and deductive database system⁷. The union of B , F , I , and K makes up the BPKB on which to activate the reasoning process by running the query *wf_proc* (*start,end*) for any given start event *start* and end event *end*.

⁶ For sake of concision we omit the sets T5, T6 for the inclusive and exclusive branch points.

⁷ The XSB Logic Programming System. Version 3.1, Aug. 2007, <http://xsb.sourceforge.net>.

The log analysis and trace generation services are performed by translating the knowledge base $T \cup B$ to a Prolog program in the reasoning engine. As above, this translation is straightforward, as $T \cup B$ is a set of Horn clauses.

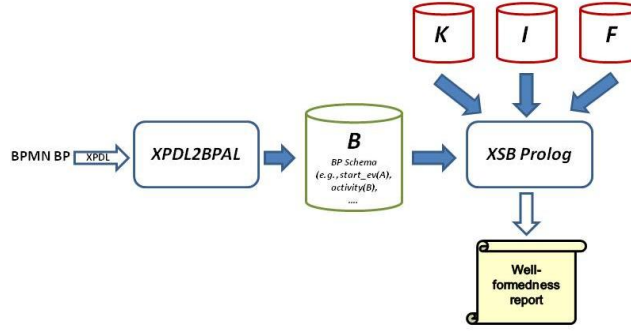


Fig. 2. Architecture of the Well-Formedness Service

The log analysis service consists in checking whether or not a given trace t is correct w.r.t. the well-formed BP B . This task is performed by the reasoning engine by running a query of the type $trace(t)$, where t is the trace to be checked (Figure 3.a).

The trace generation service consists in generating all correct traces (Figure 3.b). This task is performed by running a query of the type $trace(T)$, where T is a free variable.

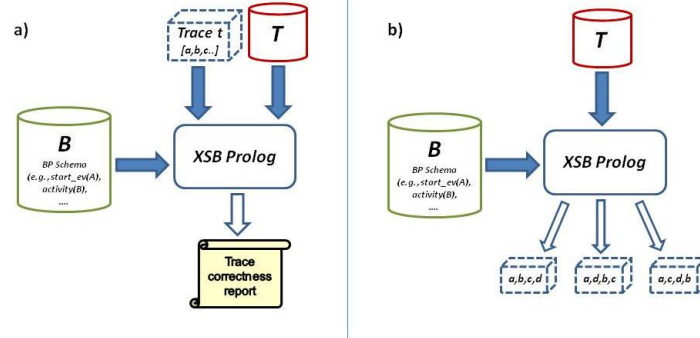


Fig. 3. a) Architecture of the Log Analysis and b) Traces Generation Service

7 Conclusions and Future Work

In this paper we presented a platform to manage a BPKB, with the intent to complement existing business modeling tools by providing advanced reasoning

services. The proposed BPKB is articulated in three levels: meta-model, BP schemas, BP execution traces. The illustrated services are: the well-formedness verification service, the log analysis, and trace generation service. The platform is based on the BPAL formal representation language. A first evaluation of the services in the eProcurement domain shows the viability of the approach.

The proposed solution establishes a logic-based framework that we intend to expand in several directions. The first direction concerns the tight integration of business ontologies (i.e., structural knowledge) represented by OPAL [13], and the behavioral knowledge, represented by BPAL. The integration with an ontology will allow advanced services for the BP query and the Business Rules (BRs) representation and management. For the latter, we intend to develop an extended framework where BPs and BRs are integrated and jointly analyzed to check if, for instance, there are processes that violate newly issued rules. A second direction will be to investigate the problem of Business Process Reengineering, with the aim to explore the possibility of manipulating a set of business processes to produce a new, optimized (e.g., in terms of process length or aggregating sub-processes that are shared by different BPs) set of reengineered BPs.

References

1. Reisig W. and Rozenberg G., editors. Lectures on Petri Nets I: Basic Models, volume 1491 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1998.
2. Pesic, M., van der Aalst, W.M.P.: A Declarative Approach for Flexible Business Processes Management. In BPM 2006 Workshops, LNCS 4103. pp. 169-180, 2006.
3. Montali, M., et al.: Verification from Declarative Specifications Using Logic Programming. In ICLP 2008, LNCS 5366, pp. 440–454, 2008.
4. Alberti, M., et al.: Verifiable agent interaction in abductive logic programming: the SCIFF framework. ACM Transactions on Computational Logics, 9(4):1-43, 2008.
5. Conrad, B., Gruninger, M.: Psl: A semantic domain for flow models. Software and Systems Modeling, 4(2):209–231, May 2005.
6. OMG: Business Process Model and Notation. Version 2.0, August 2009, <http://www.omg.org/spec/BPMN/2.0>.
7. De Nicola A., Lezoché M., Missikoff M.: An Ontological Approach to Business Process Modeling. IICAI 2007, Pune, India, 17 -19 Dicembre, 2007.
8. Lloyd, J.W.: Foundations of Logic Programming. Springer-Verlag, Berlin, 1987.
9. Dumas M., van der Aalst W., ter Hofstede A.H.M.: Process-Aware Information Systems. WILEY-INTERSCIENCE, 2005.
10. Eder J., Gruber W.: A Meta Model for Structured Workflows Supporting Workflow Transformations. In Proc. of ADBIS 2002, Bratislava, Slovakia, September 8-11, 2002.
11. Combi C. and Gambini M. Flaws in the Flow: The Weakness of Unstructured Business Process Modeling Languages Dealing with Data. LNCS, vol. 5870 Springer Berlin, 2009.
12. van Dongen B.F., Mendling J., and van der Aalst W.M.P.: Structural Patterns for Soundness of Business Process Models. In Proc. of EDOC 2006, China, 2006.
13. D’Antonio F., Missikoff M., Taglino F., Formalizing the OPAL eBusiness ontology design patterns with OWL, in Proc of I-ESA 2007.
14. Scheer A.-W., Thomas O., Adam O.: *Process Modeling Using Event-Driven Process Chains*. In “Process-Aware Information Systems”. Edited by M. Dumas, W. van der Aalst, A.H.M ter Hofstede. WILEY-INTERSCIENCE, Pages 119-145, (2005).