An Open Platform for Business Process Modeling and Verification[1]

Antonio De Nicola[1], Michele Missikoff[1], Maurizio Proietti[1], Fabrizio Smith[1,2]

[1] IASI-CNR, Viale Manzoni 30, 00185, Rome, Italy
[2] 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 that includes a logic-based language for business process (BP) modeling and a reasoning mechanism providing support for several tasks. 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 (well-formedness) 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 services for *i)* checking if the actual execution of a BP has been carried out in accordance with the corresponding definition, *ii)* simulating executions by trace generation. The proposed platform is open since it can easily be enhanced by adding other logic-based modeling, reasoning, and querying functionalities.

Keywords: business process, 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. Several tools[2] are today available on the market, open source or free of charge. Many of these tools are able to provide, besides a graphical editor, additional services, such as some forms of verification, simulation of the designed processes, execution or (semi) automatic generation of executable code (e.g., in the form of BPEL[3] code). The availability of the mentioned tools has further pushed the diffusion of several languages (e.g.,

[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).

[2] See for instance: Intalio BPMS Designer, Tibco Business Studio, ProcessMaker, YAWL, JBPM, Enhydra Shark

[3] Business Process Execution Language. See: http://www.oasis-open.org/specs/#wsbpelv2.0

BPMN [7], EPC [15]) used both in the academic and in the industrial realities. But, despite the growing academic interest and the penetration in the business domain, heterogeneous and ad-hoc solutions that often lack a formal semantics have been so far proposed to deal with the different perspectives that are of interest for an effective BP management: workflow modeling, business rules representation, integration of organizational and data models, data flow, query and retrieval of BP fragments, enactment, reengineering, log analysis, process mining.

This paper mainly intends to lay the formal foundations of a platform for BP modeling and verification. The proposed platform is centered around BPAL (Business Process Abstract Language) [8], a logic-based language for modeling the dynamic behavior of a business process from a workflow perspective. BPAL relies on a formalism, Horn clause logic, that is particularly well-suited for its use within a wider knowledge representation framework (for instance in conjunction with rule based ontology languages [19,20]) with an uniform semantics. The use of a logic-based uniform approach makes the BPAL platform open to further extensions and to the easy integration of more advanced functionalities, such as reasoning services for verifying consistency properties and methods for querying BP repositories, which can be supported by tools already developed in the area of logic programming.

BPAL is a rule-based formalism that provides an integrated support to the following three levels: (i) the BP meta-model, 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. The reasoning support provided by the BPAL platform allows the BP designer to: (i) verify business process schema well-formedness with respect to the given meta-model; (ii) verify if a given process trace, i.e., the actual execution of a BP, is compliant with a well-formed BP schema; (iii) simulate a BP execution by generating all possible traces (which are finitely many, whenever the BP schema is well-formed).

The BPAL platform is characterized by both a solid formal foundation and a high level of practical usability. The formal foundation is rooted in the logic-based approach of the BPAL language. 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 trace 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, as its application to the management of complex processes and systems [10] 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. The BPAL framework is positioned among the logic-based languages but, with respect to existing proposals, it is characterized by enhanced adaptability, 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.

Formal semantics of process modeling languages (e.g., the BPMN case is discussed in [1]) is usually defined in terms of a mapping to Petri nets [2]. Petri nets represent a powerful formal paradigm to support automatic analysis and verification of BPs within a procedural approach. A different approach is represented by the logic-based formalisms. A logical approach appears more suited to manipulate, query, retrieve, compose BP diagrams. Furthermore, by using Petri Nets, it is difficult to provide a “meta-level” that can be used to guide and constrain business process modeling, verifying properties at the intensional level.

As already mentioned, a different approach to formal BP specification is represented by a logic-based declarative approach [3,4]. Here 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. [3] proposes ConDec, a declarative flow language to define process models that can be represented as conjunction of Linear Temporal Logic formulas. This approach allows the BP designer to verify properties by using model checking techniques. [4] proposes a verification method based on Abductive Logic Programming (ALP) and, in particular, the SCIFF framework [5], that is an ALP rule-based language and a family of proof procedures for specification and verification of event-based systems. [3,4], 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. Such approaches are mostly intended to complement and extend fully procedural languages rather than replace them, as in the case of Declare[4], which is implemented within the YAWL[5] workflow management system.

PSL (Process Specification Language) [6], defines a logic-based neutral representation for manufacturing processes. A PSL ontology is organized into PSL-CORE and a partially ordered set of extensions. The PSL-CORE axiomatizes a set of intuitive semantic primitives (e.g., *activities*, *activity occurrences*, *time points*, and *objects*) enabling the description of the fundamental concepts of processes, while a set of extensions introduce new terminology and its logical formalization. Although PSL

[4] http://www.win.tue.nl/declare/

[5] http://www.yawlfoundation.org/

is defined in first order logic, which in principle makes behavioral specifications in PSL amenable to automated reasoning, we are not aware of PSL implementations for the modeling, verification or enactment of BPs, since it is intended mostly as a language to support the exchange of process information among systems.

Concurrent Transaction Logic (CTR) [17] is a formalism for declarative specification, analysis, and execution of transactional processes, that has been also applied to modeling and reasoning about workflows and services [18]. CTR formulas extend Horn clauses by introducing three new connectives: *serial conjunction*, which denotes sequential executions, *concurrent conjunction*, which denotes concurrent execution, and *isolation*, which denotes transactional executions. The model-theoretic semantics of CTR formulas is defined over paths, i.e., sequences of states. These formulas can be compiled for the execution[6] in a Prolog environment. Unlike a CTR formula, a BPAL process specification (i.e., the Horn clauses specifying the meta-model, the process schema, and the trace semantics) can be directly viewed as an executable logic program and, hence: (i) a BPAL specification can be queried by any Prolog system without need of a special purpose compiler, (ii) BPAL traces are explicitly represented and can be directly analyzed and manipulated, and (iii) other knowledge representation applications (e.g., ontology management systems) can easily be integrated by providing a suitable translation to logic programming.

3 The BPAL Language

BPAL is a logic-based language that has been conceived to provide a declarative modeling method capable of fully capturing the procedural knowledge in a business process. BPAL constructs are common to the most used and widely accepted BP modeling languages (e.g., BPMN, UML activity diagrams, EPC) and, in particular, its core is based on BPMN 2.0 specification [7]. Furthermore, the design principles of the language follow the MOF paradigm[7] with the four levels briefly reported below:

M3: Meta-metalevel. The top level is represented by the logical formalism that is applied to describe the lower levels. In particular we adopted Horn logic, due to its widespread popularity and the mature technological support provided by the numerous Prolog systems which are available.

M2: Metalevel. Here it is defined the meta-model, establishing the rules for building well-formed BPs.

M1: Schema level. This is the modeling level where it is defined the BP schema, in accordance with the given meta-model, that represents the business logic of the process.

M0: Trace level. This is the *ground* level, used to model the executions of a business process, in accordance with the corresponding BP schema.

[6] http://flora.sourceforge.net/

[7]OMG, (2006), Meta Object Facility (MOF) Core Specification V2.0, http://www.omg.org/docs/formal/06-01-01.pdf.

From a formal point of view, the BPAL language consists of two syntactic categories: (i) a set *Entities* of 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. Finally, a BPAL business process schema (BPS) 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 $C_1, \dots, C_n \in$ *Entities.*

The entities occurring in a BPS 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.

Furthermore BPAL provides a set of relational predicates to model primarily the sequencing of activities. Then, in case of branching flows, BPAL provides *parallel* (i.e., AND), *exclusive* (i.e., XOR), and *inclusive* (i.e., OR) *branching/merging* of the control flow. Here we adopted the standard semantics for branching and merging points:

seq(el1,el2): the flow element *el1* is immediately followed by *el2*.
par_branch(gat,el1,el2)[8]: *gat* is a *parallel branch* point from which the business process branches to two sub-processes started by *el1* and *el2* executed in *parallel*;
par_mrg(el1,el2,gat): *gat* is a *parallel merge* point where the two sub-processes ended by *el1* and *el2* are synchronized;
inc_branch (gat,el1,el2)[9]: *gat* is an *inclusive branch* 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;

[8] 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.

[9] *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.

exc_branch(gat,el1,el2): *gat* is an *exclusive branch* point from which the business process branches to two sub-processes started by *el1* and *el2* executed in mutual exclusion;

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;

To better present the BPAL approach, we briefly introduce as a running example a fragment of an eProcurement. 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 BPS.

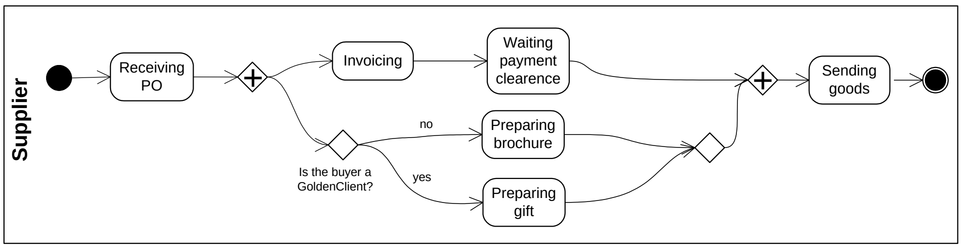


Fig. 1. BPMN specification of a fragment of an eProcurement example

Table 1. BPAL BPS of the eProcurement example

start_ev(Start)	*end_ev(End)*
activity(ReceivingPO)	*seq(Start,ReceivingPO)*
activity(Invoicing)	*seq(ReceivingPO,Gat1)*
activity(WaitingPaymentClearence)	*seq(Invoicing,WaitingPaymentClearence)*
activity(PreparingGift)	*seq(Gat2,SendingGoods)*
activity(PreparingBrochure)	*seq(SendingGoods,End)*
activity(SendingGoods)	*par_branch(Gat1,Invoicing,Gat3)*
par_branch_pt(Gat1)	*par_mrg(WaitingPaymentClearence,Gat4,Gat2)*
par_mrg_pt(Gat2)	*exc_branch(Gat3,PreparingBrochure,PreparingGift)*
exc_branch_pt(Gat3)	*exc_mrg(PreparingBrochure,PreparingGift,Gat4)*
exc_mrg_pt(Gat4)	

4 BPAL Meta-Model

The first service provided by BPAL enables the BP designer to check the compliance of a BPS with the BP meta-model, i.e., with a set of rules that constitute a guidance for the construction of the BP.

In this paper the main assumption imposed by the BPAL meta-model is the structuredness. According to [11], a **strictly structured** BP can be defined as follows: it consists of *m* sequential *blocks*, $T_1 \ldots T_m$. 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 [12]. Among them, error patterns [13] 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. We will refer to such a property as *well-formedness*.

In the rest of this section we describe the core of the meta-model of BPAL by means of a set of rules (i.e., a first order logic theory) ***MM***, which specifies when a BP is well-formed, i.e., it is syntactically correct. ***MM*** consists of three sets of meta-rules[10]: (1) a set ***I*** of inclusion axioms among the BPAL entities, (2) a set ***K*** of *schema constraints* (in the form of first order formulas), and (3) a set ***F*** of *process composition rules* (in the form of Horn clauses).

The set ***I*** of *inclusion axioms* defines a taxonomy among the BPAL entities, as informally described in Section 3. They are reported in Table 2.

Table 2. BPAL inclusion axioms

event(x) → *flow_el(x)*	*mrg_pt(x)* → *gateway(x)*
activity(x) → *flow_el(x)*	*par_branch_pt(x)* → *branch_pt(x)*
gateway(x) → *flow_el(x)*	*exc_branch_pt(x)* → *branch_pt(x)*
start_ev(x) → *event(x)*	*inc_branch_pt(x)* → *branch_pt(x)*
int_ev(x) → *event(x)*	*par_mrg_pt(x)* → *mrg_pt(x)*
end_ev(x) → *event(x)*	*exc_mrg_pt(x)* → *mrg_pt(x)*
branch_pt(x) → *gateway(x)*	*inc_mrg_pt(x)* → *mrg_pt(x)*

The set ***K*** of schema constrains (Table 3) consists of three subsets: (i) the *domain constraints*, (ii) the *type constraints*, and (iii) the *uniqueness constraints*.

[10] All formulas in ***MM*** are universally quantified in front and, for sake of simplicity, we will omit to write those quantifiers explicitly.

Table 3. BPAL schema constraints and supporting examples

Schema constraint	*Example*
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. $par\_branch(x,l,r) \rightarrow par\_branch\_pt(x) \wedge flow\_el(l) \wedge flow\_el(r)$
Uniqueness Constraints are rules expressing that the precedence relations between flow elements are specified in an unambiguous way: *branching uniqueness constraints* asserting that every (parallel, inclusive, exclusive) branching point has exactly one pair of successors. *merging uniqueness constraints* asserting that every merge point has exactly one pair of predecessors. *sequence uniqueness constraints* asserting that, by the *seq* predicate, we can specify at most one successor and at most one predecessor of any flow element.	Example of sequence uniqueness constraint: $seq(x,y) \wedge seq(x,z) \rightarrow y=z$ $seq(x,z) \wedge seq(y,z) \rightarrow x=y$

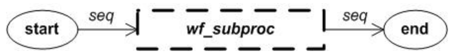
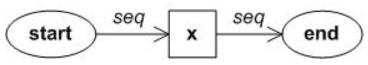
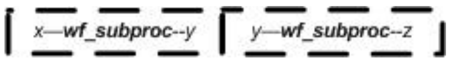
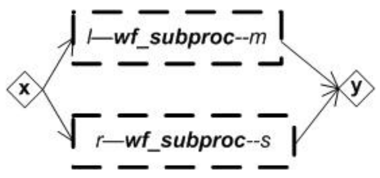
The set ***F*** of *process composition rules* provides the guidelines for building a well-formed BPS. Then, in formal terms, it is possible to verify if a process respects such rules by means of a predicate *wf_proc*(*s,e*) which holds if the business process started by the event *s* and ended by the event *e* is *well-formed*. In Table 4, 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*).

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 $\boldsymbol{B}\cup\boldsymbol{F}\cup\boldsymbol{I}$, and

(ii) for every start event *S* and end event *E*, *wf_process*(*S*,*E*) can be inferred from $\boldsymbol{B}\cup\boldsymbol{F}\cup\boldsymbol{I}$.

Table 4. BPAL process composition rules and supporting diagrammatic description

Process composition rule	***Intuitive Diagram***
F1. A business process schema is *well-formed* 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 *well-formed* 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[11]: $par\_branch(x,l,r) \wedge wf\_sub\_proc(l,m) \wedge wf\_sub\_proc(r,s) \wedge par\_mrg(m,s,y) \rightarrow wf\_sub\_proc(x,y)$	A well-formed sub-process including merge and branch points: 

5 BPAL Execution Traces

An execution of a business process is a sequence of instances of activities called *steps*; the latter may also represent instances of events. Steps are denoted by constants taken from a set *Step* disjoint from *Entities*. Thus, a possible execution of a business process is a sequence $[s_1, s_2,\ldots, s_n]$, where $s_1, s_2,\ldots, s_n \in$ *Step*, called a *trace*. The *instance* relation between steps and activities (or events) is specified by a binary

[11] 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.

predicate *inst(step,activity)*. For example, *inst(RPO1*, *ReceivingPO)* states that the step *RPO1* is an activity instance of *ReceivingPO*.

Table 2. BPAL Trace rules

T1. A sequence [*s1,…,e1*] of steps is a *correct trace* if: (i) *s1* is an instance of a start event, (ii) *e*1 is an instance of an end event, and (iii) [*s1,…,e1*] is a correct *sub-trace* from *s1* to *e1* constructed according to the sets of rules T2-T6: $\mathit{start\_ev}(s) \wedge \mathit{inst}(s1,s) \wedge \mathit{sub\_trace}(s1,t,e1) \wedge \mathit{end\_ev}(e) \wedge \mathit{inst}(e1,e) \rightarrow \mathit{trace}(t)$
T2. Any instance of an activity/event or a sequence of instances of activities/events is a correct sub-trace. $\mathit{inst}(x1,x) \wedge \mathit{activity}(x) \rightarrow \mathit{sub\_trace}(x1,[x1],x1)$ $\mathit{inst}(x1,x) \wedge \mathit{int\_ev}(x) \rightarrow \mathit{sub\_trace}(x1,[x1],x1)$ $\mathit{inst}(x1,x) \wedge \mathit{inst}(y1,y) \wedge \mathit{seq}(x,y) \wedge \mathit{act\_or\_ev\_seq}([x1,y1],t) \rightarrow \mathit{sub\_trace}(x1,t,y1)$ where the predicate *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: $\mathit{sub\_trace}(x1,t1,y1) \wedge \mathit{sub\_trace}(y1,t2,z1) \wedge \mathit{concatenation}(t1,t2,t) \rightarrow \mathit{sub\_trace}(x1,t,z1)$ 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 *z1* is defined by the following rule[12]: $\mathit{inst}(x1,x) \wedge \mathit{inst}(l1,l) \wedge \mathit{inst}(r1,r) \wedge \mathit{par\_branch}(x,l,r) \wedge \mathit{inst}(m1,m) \wedge \mathit{sub\_trace}(l1,t1,m1)$ $\wedge \mathit{inst}(s1,s) \wedge \mathit{sub\_trace}(r1,t2,s1) \wedge \mathit{inst}(y1,y), \mathit{par\_mrg}(m,s,y), \mathit{interleaving}(t1,t2,t)$ $\rightarrow \mathit{sub\_trace}(x1,t,y1)$ 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.*

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). 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]

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

where *inst(s,Start), inst(r,ReceivingPO), inst(i,Invoicing), inst(w,WaitingPayment Clearence), inst(pG,PreparingGift), inst(pB,PreparingBrochure), inst(sG,Sending Goods), inst(e,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 *trace*(*t*), which holds if *t* is a correct trace, with respect to a BP, of the form $[s_1, s_2, \ldots, s_n]$, where s_1 is an instance of a start event and s_n is an instance of an end event. The predicate *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* and, for lack of space, in Table 5 we list only the trace rules corresponding to the composition rules presented in Section 4. The trace axioms are defined by induction on the length of the trace *t*.

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

6 The BPAL Platform

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

In the BPAL platform the verification of well-formedness of a BPS is performed as depicted in Figure 2. The BPMN graphical specification of a given BP-1 business process is exported as XPDL[13] (XML Process Definition Language) and translated into a set of BPAL ground facts by means of the service XPDL2BPAL, thereby producing the BPS ***B***. Then ***B*** is passed to the reasoning engine 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 axioms. Thus, the union of ***B***, ***F***, ***I***, and ***K*** makes up a knowledge base from which we want to infer that ***B*** is well-formed. This inference task is performed by the BPAL reasoning engine which is implemented by using the XSB logic programming and deductive database system [16]. XSB extends the conventional Prolog systems with an operational semantics based on *tabling,* i.e., a mechanism for storing intermediate results and avoiding to prove sub-goals more than once. XSB has several advantages over conventional Prolog systems based on SLDNF-resolution (such as, SWI-Prolog and SICStus Prolog): (i) in many cases XSB is more efficient than conventional systems, (ii) it guarantees the termination of queries to DATALOG programs (i.e., Prolog programs without function symbols), and (iii) it often avoids to return several times the same answer to a given query.

B∪***F***∪***I*** is a set of Horn clauses and, therefore, it is translated to a Prolog program in a straightforward way. The schema constraints in ***K*** are translated to Prolog queries. For instance, the domain constraint *activity*(*x*) → ¬ *event*(*x*) is translated to the query:

[13] http://www.wfmc.org/xpdl.html

```
?- activity(X), event(X).
```

For the eProcurement example of Figure. 1, XSB answers ‘no’ to this query, meaning that there is no X which is an activity and an event at the same time. Hence, this domain constraint is inferred from ***B***∪***F***∪***I***.

Moreover, for any given start event *start* and end event *end*, we can check whether or not *wf_proc* (*start*,*end*) is inferred from ***B***∪***F***∪***I***, by running the query:

```
?- wf_proc(start,end).
```

For the eProcurement example of Figure. 1, XSB answers ‘yes’ to this query, meaning that *wf_proc* (*start*,*end*) is inferred from ***B***∪***F***∪***I*** and, thus, the well-foundedness of ***B*** is verified.

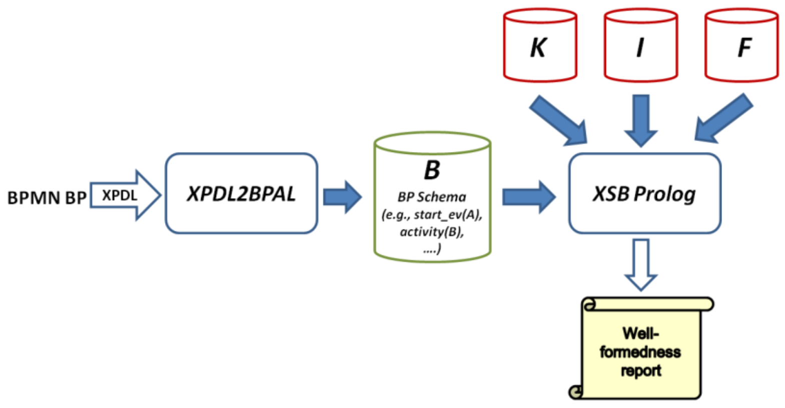


Fig. 2. Architecture of the well-formedness service

The trace analysis and trace generation services are performed by translating the theory ***T***∪***B*** to a Prolog program in the reasoning engine. As above, this translation is straightforward, as ***T***∪***B*** is a set of Horn clauses. The trace analysis service consists in checking whether or not a given trace *t* is correct w.r.t. the well-formed BPS ***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). For instance, in the eProcurement example, XSB behaves as follows:

```
?- trace([s,r,i,pG,w,sG,e]).
yes
?- trace([s,r,i,pG,pB,w,sG,e]).
no
```

Indeed, the first sequence is a correct trace and the second is not.

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 (Figure 3.a). In the eProcurement example, XSB behaves as follows:

```
?- trace(T).
T = [s,r,pG,i,w,sG,e];
T = [s,r,i,w,pG,sG,e];
T = [s,r,i,w,pB,sG,e];
T = [s,r,i,pG,w,sG,e];
T = [s,r,i,pB,w,sG,e];
T = [s,r,pB,i,w,sG,e];
no
```

meaning that the above sequences are all and only the correct execution traces of the given BPS.

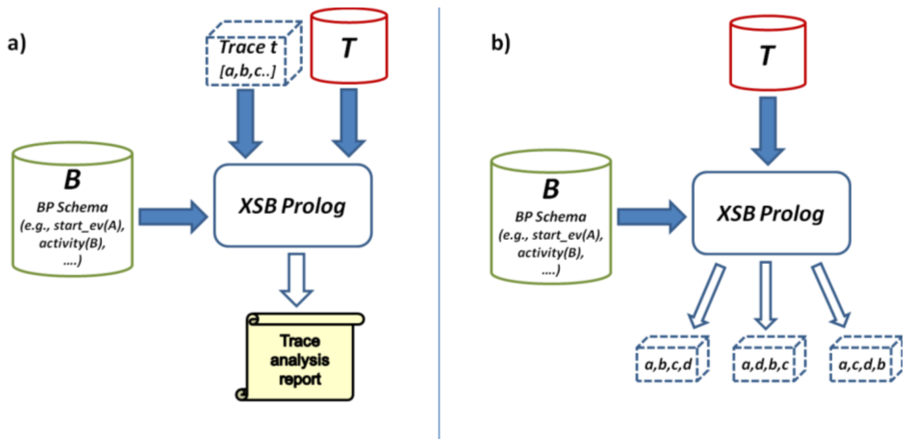


Fig. 3. a) Architecture of the trace analysis and b) traces generation service

7 Conclusions and future works

In this paper we presented a platform to complement existing business modeling tools by providing advanced reasoning services: the well-formedness verification service, the trace analysis and the trace generation service. The platform is centered around the logic-based BPAL language. A first evaluation of the services in the eProcurement domain shows the viability of the approach.

We intend to expand the BPAL platform in several directions. A first direction will be the tight integration of business ontologies (i.e., structural knowledge) represented by OPAL, and the behavioral knowledge, represented by BPAL. OPAL is an ontology representation framework supporting business experts in building a structural ontology, i.e., where concepts are defined in terms of their information structure and static relationships. In building an OPAL ontology, knowledge engineers typically start from a set of upper level concepts, and proceed according to a paradigm that highlights the active entities (actors), passive entities (objects), and transformations (processes). The latter are represented only in their structural components, without modeling the behavioral issues, delegated to BPAL. As shown in [14], a significant core of an OPAL ontology can be formalized by a fragment of OWL, relying within the OWL-RL profile [19,20] an OWL subset designed for practical implementations using rule-based technologies such as logic programming.

Another direction concerns the modeling and the verification of Business Rules (BRs). This is motivated by the fact that in real world applications the operation of an enterprise is regulated by a set of BPs that are often complemented by specific business rules. We intend to enhance the BPAL platform so as to support the integrated modeling of BPs and BRs. New reasoning services will also be needed for analyzing those integrated models to check if, for instance, there are possible executions of processes that violate any given business rule.

Since BPs play a growing role in business realities, we foresee a scenario where huge repositories of process models developed by different designers have to be managed. In such a scenario there will be the need for advanced reasoning systems aimed at query processing, for the retrieval of process fragments to be used in the design of new BP models, and at verifying that some desired properties hold. We intend to enhance the BPAL platform in such a way that we can issue several types of queries, both at intensional and extensional level. In particular, we are interested in the following three types of queries and combinations thereof. (1) Queries over BP schemas. Querying the BPS allows the search for certain patterns adopted in the design phase and the verification of constrains that descend from structural requirements to be done. (2) Queries over BP traces. Here the behavior at execution time is of interest, and the properties to be verified regard the temporal sequencing of activities in the set of correct traces. (3) Queries over the Business Ontology. Here the focus is on the domain entities (processes, actors, objects) and their relationships.

Finally, on an engineering ground, we intend to investigate the problem of Business Process Reengineering, and 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. Dijkman, R.M., Dumas, M., Ouyang, C.: Formal semantics and automated analysis of BPMN process models. In: Preprint 7115. Queensland University of Technology, Brisbane, Australia (2007).
2. 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.
3. 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.

4. Montali, M., Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verification from Declarative Specifications Using Logic Programming. In ICLP 2008, LNCS 5366, pp. 440–454, 2008.
5. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable agent interaction in abductive logic programming: the SCIFF framework. ACM Transactions on Computational Logics, 9(4):1-43, 2008.
6. Conrad, B. and Gruninger, M.: Psl: A semantic domain for flow models. Software and Systems Modeling, 4(2):209–231, May 2005.
7. OMG: Business Process Model and Notation. Version 2.0, August 2009, http://www.omg.org/spec/BPMN/2.0.
8. De Nicola, A., Lezoche, M., Missikoff, M.: An Ontological Approach to Business Process Modeling. 3rd Indian International Conference on Artificial Intelligence (IICAI 2007), Pune, India, 17 -19 Dicembre, 2007.
9. Lloyd, J.W.: Foundations of Logic Programming. Springer-Verlag, Berlin, 1987.
10. Dumas, M., van der Aalst, W., ter Hofstede, A.H.M.: Process-Aware Information Systems. WILEY-INTERSCIENCE, 2005.
11. Eder, J. and Gruber, W.: A Meta Model for Structured Workflows Supporting Workflow Transformations. In Proc. 6th East European Conference on Advances in Databases and Information Systems (ADBIS 2002), pp: 326-339, Bratislava, Slovakia, September 8-11, 2002.
12. Combi, C. and Gambini, M.: Flaws in the Flow: The Weakness of Unstructured Business Process Modeling Languages Dealing with Data. OTM Conferences, volume 5870 of Lecture Notes in Computer Science, pp. 42-59, Springer Berlin, 2009.
13. van Dongen, B.F., Mendling, J., and van der Aalst, W.M.P.: Structural Patternsfor Soundness of Business Process Models. In Proceedings of EDOC 2006, Hong Kong, China, 2006. IEEE.
14. D'Antonio, F., Missikoff, M., Taglino, F.: Formalizing the OPAL eBusiness ontology design patterns with OWL, in: Third International Conference on Interoperability for Enterprise Applications and Software, I-ESA 2007.
15. 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).
16. The XSB Logic Programming System. Version 3.1, Aug. 2007, http://xsb.sourceforge.net.
17. Bonner, A. J. and Kifer, M.: Concurrency and Communication in Transaction Logic. In Joint International Conference and Symposium on Logic Programming, 1996.
18. Roman, D. and Kifer, M.: Reasoning about the Behavior of Semantic Web Services with Concurrent Transaction Logic. In VLDB, 2007.
19. OWL 2: Profiles, http://www.w3.org/TR/owl2-profiles.
20. Grosof, B. N., Horrocks, I., Volz, R., Decker, S.: Description Logic Programs: Combining Logic Programs with Description Logic, in: Proceedings of the 12th International Conference on World Wide Web, ACM, 2003.