ISTITUTO DI ANALISI DEI SISTEMI ED INFORMATICA
"Antonio Ruberti"
CONSIGLIO NAZIONALE DELLE RICERCHE

M. Missikoff, M. Proietti, F. Smith

QUERYING SEMANTICALLY ANNOTATED BUSINESS PROCESSES

R. 10-22, 2010

Michele Missikoff – Istituto di Analisi dei Sistemi ed Informatica del CNR, viale Manzoni 30, I-00185 Roma, Italy. Email: michele.missikoff@iasi.cnr.it.

Maurizio Proietti – Istituto di Analisi dei Sistemi ed Informatica del CNR, viale Manzoni 30, I-00185 Roma, Italy. Email: maurizio.proietti@iasi.cnr.it.

Fabrizio Smith – Dipartimento di Ingegneria Elettrica e dell'Informazione, Università degli Studi de L'Aquila, I-67040 Monteluco di Roio, L'Aquila, Italy and Istituto di Analisi dei Sistemi ed Informatica del CNR, viale Manzoni 30, I-00185 Roma, Italy. Email: fabrizio.smith@iasi.cnr.it.

This work was partially done under IASI support

ISSN: 1128–3378

Collana dei Rapporti dell'Istituto di Analisi dei Sistemi ed Informatica "Antonio Ruberti", CNR

viale Manzoni 30, 00185 ROMA, Italy

tel. ++39-06-77161
fax ++39-06-7716461
email: iasi@iasi.cnr.it
URL: http://www.iasi.cnr.it

Abstract

In this paper we present a logic-based framework for querying semantically annotated business process repositories. The proposed solution is based on a synergic use of ontologies to capture the semantics of a business scenario, and a business process modelling framework (BPAL), to represent the workflow logic. Both frameworks are grounded in a logic-based formalism (Logic Programming) and therefore it is possible to apply effective reasoning methods to query the Business Process Knowledge Base stemming from the fusion of the two. A software platform has been developed and the first extensive tests are encouraging.

Keywords: Business Process, Semantic Annotation, Ontology, Query Language

1 Introduction

Business Process (BP) management is constantly gaining popularity in various industrial sectors and in the public administration. But, despite the penetration in the business domain and the growing academic interest, there are a number of services, related to the management of large process repositories, that are not yet available in the commercial Business Process Management Systems (BPMS). For instance, beyond the functions currently offered by existing BPMS, there is the need to address more advanced issues, such as: cross-enterprise integration and collaboration, integration of organizational and data models with workflow models, query and retrieval of BP fragments, BP composition. Such advanced services are difficult to be achieved with ad-hoc solutions and, conversely, there is a growing consensus on the advantages that can derive from the adoption of automatic reasoning techniques. To this end, it is necessary to expand the formal background and the underlying theory of business processes following a logic-based approach, suitable for an effective automatic reasoning aimed at increasing the level of automation in their specification, analysis, implementation and monitoring.

Another direction to enhance the BP management is towards an integrated representation of the business context in which the processes take place. In particular, it is useful to represent the involved actors, objects, and business resources in a way that is homogeneous with the representation of BPs. Along this line, various papers have advocated the enhancement of BP management tools by means of well-established techniques from the area of the Semantic Web, like, for instance, computational ontologies [1]. The use of an ontology allows an unambiguous definition of the entities occurring in the domain, and eases the interoperability between software applications and the reuse/exchange of knowledge between human actors. However, there are still several open issues regarding the combination of workflow languages (procedural in their nature, with an execution semantics) and ontologies (declarative in their nature, with a denotational semantics), and the accomplishment of reasoning tasks involving both these components.

In this paper we present a logic-based framework that aims at providing a uniform and formal representation of both the behavioral (i.e., workflow-related) and the structural (i.e., ontology-related) domain knowledge about a business process. Our framework is also equipped with a powerful inference mechanism supported by the solutions developed in the area of Logic Programming [2], capable of efficient querying over semantically annotated business processes.

In essence, the main contributions of this work are: *i)* a method for semantic enrichment of business processes; to this end, *ii)* a tight integration of ontologies and BP repositories, with, *iii)* a unifying formal framework, centered around BPAL [3] (Business Process Abstract modeling Language) and based on a logic programming approach; then *iv)* reasoning services encompassing ontologies and BP repositories in a unique computational framework and, thanks to the latter, *v)* an effective query language and engine.

The rest of the paper is organized as follows. The BP knowledge representation framework is presented in Section 2. Section 3 describes the query support and the query language provided by the framework. Then, Section 4 describes the BPAL platform and Section 5 presents related works. Finally, conclusions in Section 6 end the paper.

2 Business Process Knowledge Representation

An advanced support to BP management requires a complex analysis of the business reality and the modeling of different kinds of knowledge. Primarily, the behavioral knowledge, i.e., the execution flow represented by the activity sequencing, but also the structural knowledge regarding the domain in which the process takes place, that includes the actors associated to activities, managed objects, and their relationships.

The complex knowledge related to a BP is stored in a *Business Process Knowledge Base (BPKB)*, sketchily depicted in Figure 1. A *BPKB* is organised in five main components: *i)* a repository of BP schemas, encoded in BPAL; *ii)* a BP meta-model; *iii)* a ground level, where the execution traces are modelled; *iv)* a Business Reference Ontology (BRO), used for the representation of a business domain; *v)* a Semantic Annotation, linking elements of a BP schema to ontology concepts.

The proposed approach provides a uniform and formal representation framework, suited for automatic reasoning and equipped with a powerful inference mechanism supported by the solutions developed in the area of Logic Programming. At the same time, it perfectly integrates with all the commercial tools that adopt BPMN [4] as a modelling notation and loosely supports OWL [5], for the definition of domain ontologies.

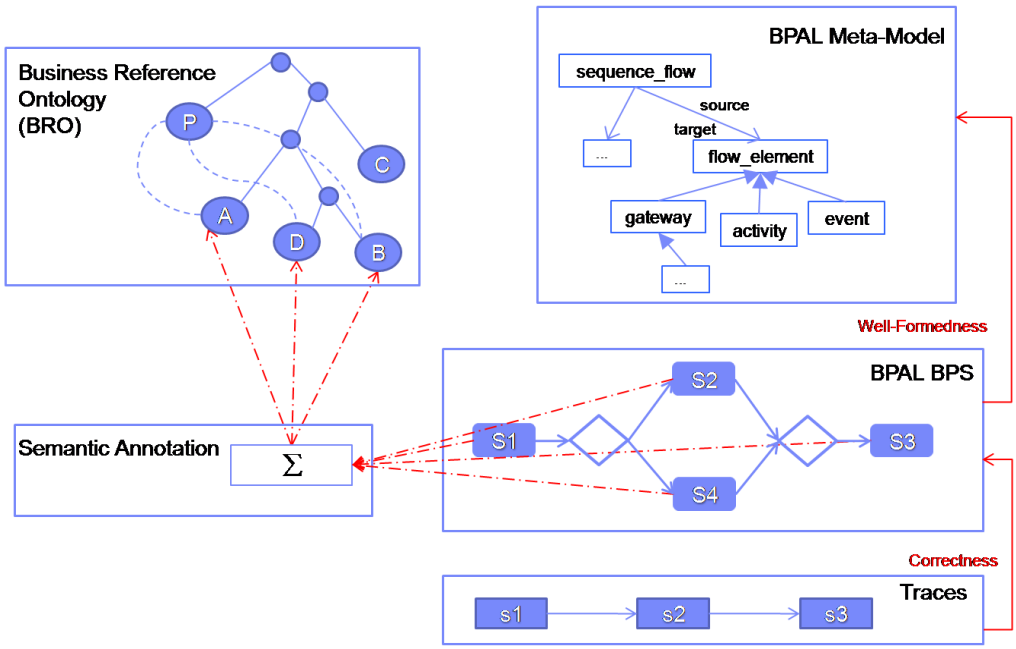

Fig. 1. Business Process Knowledge Base

Figure 2 depicts a fragment of an *eProcurement* process, that will be used as a running example throughout the paper. The activity workflow is modeled using BPMN, and its flow elements are (partially) annotated by concepts of a given BRO. An ACSE supplier, in order to process a purchase order, sends back an invoice. In the meanwhile, it sends a gift to the buyer if she/he is classified as 'golden client.' After receiving the payment clearance from the bank, the goods are delivered.

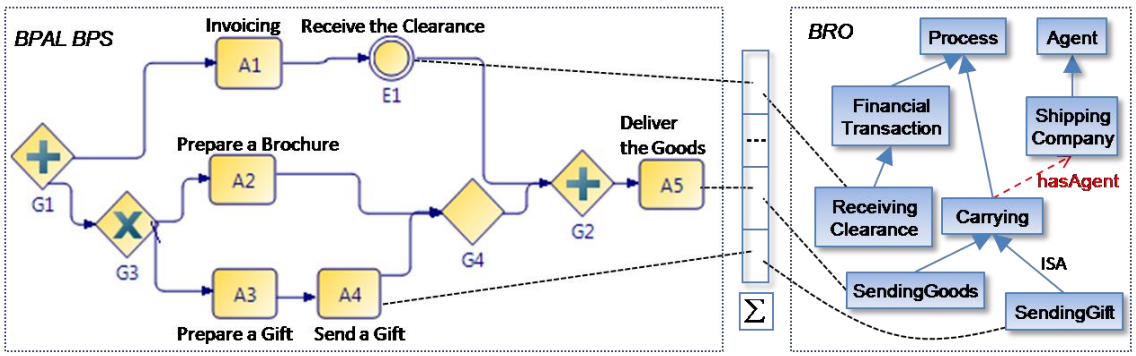

Fig. 2. Running Example: eProcurement process

2.1 BPAL Language

In this Section we present BPAL, a logic-based language (grounded in logic programming) that, having a declarative nature, is capable of fully capturing the procedural knowledge in a business process. As anticipated, BPAL constructs have been conceived along the line of the most common 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 [4].

Table 1. BPAL language

flow_el(el)	any atomic component appearing in the control flow
activity(act)	a *business activity*, the key element of the business process
event(ev)	an *event* that occurs during the process execution
start_ev(start)	an *event* which starts the business process
end_ev(end)	an *event* which ends the business process
seq(el1,el2)	the flow element *el1* is immediately followed by *el2*
branch_pt(gat)	a branch point of the control flow
merge_pt(gat)	a merge point of the control flow
par_branch (g1,el1,el2) *par_mrg(el3,el4,g2)*	the control flow branches from *g1* into two sub-processes started by *el1* and *el2,* executed in *parallel,* and synchronized in *g2*
exc_branch(g1,el1,el2) *exc_mrg(el3,el4,g2)*	the control flow branches from *g1* into two sub-processes started by *el1* and *el2,* executed in *mutual exclusion,* and merged in *g2*
inc_branch(g1,el1,el2) *inc_mrg(el3,el4,g2)*	the control flow branches from *g1* into two sub-processes started by *el1* and *el2* and merged in *g2.* At least one branch is executed

Table 1 lists some of the BPAL constructs, where the unary predicates represent the types of the flow elements occurring in a BPS and the binary and ternary predicates represent the sequencing of activities. For branching flows, BPAL provides predicates representing *parallel* (AND), *exclusive* (XOR), and *inclusive* (OR) *branching/merging* of the control flow.

A BPAL BP Schema (BPS) is specified by a set of ground *facts* (i.e., atomic assertions on individual constants) of the form $p(C_1, \ldots, C_n)$, where p is a BPAL predicate and $C_1, \ldots, C_n$. are constants denoting BPS elements (e.g., business activities, events, and gateways). A BPS is uniquely identified by a process identifier. We use the relation *bpId(id,s,e)* to assign the process identifier *id* to a process starting and ending with the *s* and *e* events, respectively.

Table 2 reports the eProcurement process fragment, depicted in Figure 2, encoded as a set of BPAL assertions.

Table 2. BPAL representation of the eProcurement process

Unary predicates		**Relational Predicates**	
int_ev(E1)	*activity(A5)*	*par_branch(G1,A1,G3)*	*seq(A1,E1)*
activity(A1)	*par_branch_pt(G1)*	*par_merge(E1,G4,G2)*	*seq(A3,A4)*
activity(A2)	*par_merge_pt(G2)*	*exc_branch(G3,A2,A3)*	*seq(G2,A5)*
activity(A3)	*exc_branch_pt(G3)*	*exc_merge(A2,A4,G4)*	
activity(A4)	*exc_merge_pt(G4)*		

2.2 Semantic Annotation through a Business Reference Ontology

The language adopted for the definition of a BRO is a fragment of OWL, falling within the OWL-RL profile. OWL-RL [5], is an OWL subset designed for practical implementations using rule-based techniques, such as logic programming [6]. Hereafter we present OWL expressions using the triple notation by means of the ternary predicate *t(s,p,o)*, representing a generalized RDF triple (with subject *s*, predicate *p*, and object *o*) and assuming the usual prefixes *rdfs* and *owl* for the RDFS/OWL vocabulary. For the semantics of an OWL-RL ontology we refer to the axiomatization described in [5] by a set of FOL rules over the predicate *t* (OWL 2 RL/RDF rules).

A *Semantic Annotation* **Σ** defines a correspondence between elements of the BPS and elements of the Reference Ontology, in order to describe the meaning of the elements of a business process in terms of a suitable conceptualization of the domain of interest. **Σ** is specified using the predicate σ, and consists of a set of ground facts of the form *σ(FlowEl,Conc)*, where *FlowEl* is a constant that denotes a flow element of a BP schema, and *Conc* is a constant used to denote a concept defined in the ontology. **Σ** is formalized by a set of axioms, such as $\sigma(x,y) \wedge t(y, rdfs{:}\,subClassOf, z) \rightarrow \sigma(x,z)$, stating that σ is preserved by the *subclass* relation. **Σ** is presented in [7] in its extended form for richer annotations of BP schemas (e.g., input, output, actors).

The content of the *BPKB* for the *eProcurement* process in Figure 2 is completed by the ground facts, encoding the (partial example of) BRO and **Σ,** as reported in Table 3.

Table 3. Semantic enrichment of the eProcurement process

Semantic Annotation Σ		
$\sigma(E1, ReceivingClearance)$	$\sigma(A4, SendingGift)$	$\sigma(A5, SendingGoods)$
Business Reference Ontology BRO		
t(ReceivingClearance,rdfs:subClassOf, FinancialTransaction) *t(SendingGoods,rdfs:subClassOf,Carrying)* *t(SendingGift,rdfs:subClassOf,Carrying)* *t(Carrying,rdfs:subClassOf,r1)*	*t(r1,owl:allValuesFrom,ShippingCompany)* *t(r1,owl:onProperty,hasAgent)* *t(Carrying,rdfs:subClassOf,Process)* *t(ShippingCompany,rdfs:subCassOf,Agent)* *…*	

2.3 BPAL Theories

On top of the BPS modelling layer, we explicitly introduce a BP meta-modelling layer. The latter is essential to: *i)* provide a clear definition of what the correct BPS are and,

furthermore, *ii)* offer a clear operational guidance to the modelling experts. Therefore, the meta-model defines how the constructs provided by the BPAL language can be used to build a BPS. In formal terms, the BPAL meta-model is defined by means of a first order logic theory ***M***, called *meta-model theory,* which specifies when a BPS is well-formed, i.e., it is correct from a syntactical point of view. In this work we assume that well-formed processes, i.e., processes built according to the meta-model, are *structured* (or *blocked*) [3]. Table 4 lists some of the predicates defined in ***M***.

Table 4. BPAL Meta-Model

wf_proc(*bpId*)	the business process *bpId* is *well-formed* (i.e., structured);
wf_subproc(bpId,s,e)	the sub-process starting with *s* and ending with *e* is well-formed
belongs(flow_el,bpid)	*flow_el* belongs to the set of flow elements of the process *bpId*
belongs(flow_el,bpid,s,e)	*flow_el* belongs to the set of flow elements of the sub-process of *bpId* starting with *s* and ending with *e*

The semantics of a BPS is given in terms of the set of its correct traces, and the explicit formalization of this notion is given by a *trace theory* ***T*** [3]. A *trace* models an execution (or instance, or enactment) of a BP as a sequence $<s_1, s_2, \ldots, s_n>$ of instances of activities (or events) called *steps*. The axioms constituting the trace theory ***T*** can be viewed as a set of rules for constructing the *traces* of a given BPS. Hence they have a double nature, since they can be used to check correctness of a *trace* w.r.t. a given BPS, but also to generate the set of correct traces of a BPS. In particular ***T*** defines the predicates:

- *trace*(*t,bpId*), which holds if *t* is a correct trace of process *bpId*;
- *sub_trace(s,t,e,bpId):* which holds if *t* is a correct sub-trace of process *bpId* starting with *s* and ending with *e.*

In order to verify properties regarding the execution semantics of a BPS, i.e. properties regarding the possible executions of a BPS, the theory ***D*** [8] provides the formalization of a set of *dependency constraints* (often referred to as *compliance rules*). *Dependency constraints* state that an activity is dependent on another activity, i.e., two activities have to occur together (or in mutual exclusion) in the process (possibly, in a given order). As examples of such constraints, in Table 5 we list some predicates defined in ***D***.

Table 5. Dependency Constraints

precedence(a,b,bpId) *precedence(a,b,bpId,s,e)*	if *b* is executed then *a* has been previously executed in BPS *bpId* (resp., the sub-process of *bpId* starting with *s* and ending with *e*)
response(a,b,bpId) *response(a,b,bpId,s,e)*	if *a* is executed then *b* will be executed in BPS *bpId* (resp., the sub-process of *bpId* starting with *s* and ending with *e*)
mutex(a,b,bpId) *mutex(a,b,bpId,s,e)*	*a* and *b* are never both executed in BPS *bpId* (resp., the sub-process of *bpId* starting with *s* and ending with *e*)
coexist(a,b,bpId) *coexist(a,b,bpId,s,e)*	*a* and *b* are always both executed in BPS *bpId* (resp., the sub-process of *bpId* starting with *s* and ending with *e*)

3 Querying a Business Process Knowledge Base

Having presented the notion of a *BPKB,* and the modeling framework BPAL, in this Section we propose a method for BP querying.

The components of the *BPKB* introduced in the previous section are formalized by a First Order Logic theory, defined as

$$\boldsymbol{BPKB} = \boldsymbol{BRO} \cup \boldsymbol{\Sigma} \cup \boldsymbol{M} \cup \boldsymbol{B} \cup \boldsymbol{T} \cup \boldsymbol{D}$$

where: ***BRO*** is a Business Reference Ontology; ***Σ*** is a semantic annotation, including a set of assertions of the form *σ(FlowEl,Conc)*; ***M*** is the theory formalizing the meta-model and the related notion of well-formedness of a BP schema; ***B*** is a BP *repository*, encoded in BPAL, i.e. a set of ground facts constructed from the BPAL alphabet; ***T*** is the theory formalizing the trace semantics of a BP schema and the notion of correctness of a trace w.r.t. that schema, ***D*** is the theory formalizing the dependency constraints.

A relevant property of the *BPKB* is that it has a straightforward translation to a logic program [2], which can be effectively used for reasoning within a Prolog environment. This translation allows us to exploit several kinds of reasoning services, within the uniform framework of logic programming. Every component of the *BPKB* defines a set of predicates that can be used for querying the knowledge base. ***BRO*** and **Σ** allow us to express queries in terms of the ontology vocabulary. ***M*** and ***B*** allow us to query the schema level of a BP, verifying properties regarding the flow elements occurring in it (*activities, events, gateways*) and their relationships (*sequence flows*). Finally ***T*** and ***D***, allow us to express queries about the behavior of a BP schema at execution time, i.e., verify properties regarding the execution semantics of a BP schema.

Hence the ***BPKB*** can be used for evaluating conjunctive queries, formulated in the Prolog syntax as clauses of the form:

$$q(\vec{x}) \text{:- } p_1(\overrightarrow{x_1}), \ldots, p_m(\overrightarrow{x_m}), not\ p_{m+1}(\overrightarrow{x_{m+1}}), \ldots, not\ p_n(\overrightarrow{x_n})$$

where $p_1, \ldots, p_n$ are predicates defined in the ***BPKB,*** $q(\vec{x})$ is the query to be evaluated by the engine, $\overrightarrow{x_1}, \ldots, \overrightarrow{x_n}$ are vectors of variables such that every x occurring in $\vec{x}$ occurs also in some $\overrightarrow{x_i}$.

3.1 The QuBPAL Query Language

Having set the methodological framework, we now introduce QuBPAL, a simple and expressive language for querying the *BPKB*. It does not require the user to understand the technicalities of the underlining logic programming platform, since QuBPAL queries are automatically translated to logic programs and evaluated by using the XSB logic programming and deductive database system [9]. More specifically, QuBPAL queries which do not involve predicates defined in ***T***, i.e., queries that do not explicitly manipulate traces, are translated to logic programs belonging to the fragment of Datalog with stratified negation [10]. For this class of programs the tabling mechanism of XSB guarantees a terminating and efficient top-down evaluation.

In the queries we use question mark to denote variables (e.g., *?x*), and we use the notation *?x::ConceptID* to indicate the semantic typing of a variable, i.e. *?x* ranges over activities and events annotated with the concept *ConceptID*. A (well-formed) BPS is denoted by *<bpId>*, where *bpId* is a business process identifier. A (well-formed) sub-process is denoted by *<bpId,start,end>,* where *start* and *end* are the flow elements (activities, events or gateways) of the BPS *bpId* that start and end the sub-process, respectively. Syntactically a query is an expression of the form:

SELECT (*empty_string* | *<?bpId>* | *<?bpId,?start,?end >* |*<?start,?end >*) *?x**
FROM *<bpId>*$^+$ | *<bpId,start,end>*$^+$
WHERE *comparison_predicate*

The **SELECT** statement defines the output of the query evaluation. If this statement contains no variables, then the query returns either *true* or *false.* Otherwise, as a target list, can be specified:

- a sequence (possibly empty) *?x** of variables occurring in the WHERE statement;
- a BPS, denoted by the process identifier *<?bpId>*;
- a sub-process of a BPS, denoted by the triple *<?bpId,?start,?end>*;
- a sub-process of a particular BPS, specified in the FROM clause, by *<?start,?end>*.

The **FROM** statement indicates the process(es) from which data is to be retrieved. If it is omitted, the whole repository is considered, otherwise can be specified:

- a non empty sequence of BP schemas, *<bpId>*$^+$;
- a non empty sequence of sub-processes, *<bpId,start,end>*$^+$.

In the **WHERE** statement one can specify an expression that restricts the set of data returned by the query. The *comparison_predicate* is a sentence built from:

- the set of the predicates defined in the $\boldsymbol{BRO} \cup \boldsymbol{\Sigma} \cup \boldsymbol{M} \cup \boldsymbol{B} \cup \boldsymbol{D}$, listed in the Tables 1-5;
- the connectives AND, OR, NOT, and the predicate = with the standard logic semantics;
- another QuBPAL query, to allow nested queries.

The arguments of the predicates constituting a query are:

- semantically typed variables. (i.e.: *?x::conceptId*);
- individual constants.

3.2 Compiling QuBPAL queries into Datalog queries

The BPAL Platform (see Section 4) provides a compiler that translates QuBPAL queries into Datalog queries, performing also some optimizations on the translated queries.

The translation of QuBPAL into Datalog is similar to the translation of SQL into Datalog [10], especially for the treatment of nested and disjunctive queries. For this reason we will not show the details of the translation and we will just describe it in a pictorial way (see Figure 3) and through the examples reported in the next section. In Figure 3 we only consider some specific cases for the SELECT, FROM, and WHERE statements. The extension to the general case is straightforward. In particular nested and disjunctive queries will be translated to multiple Datalog rules.

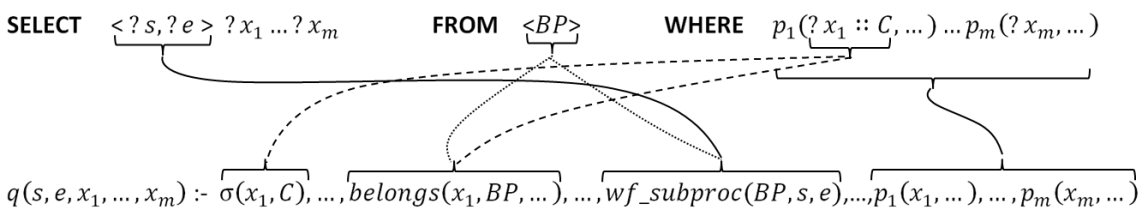

Fig. 3. Translation of QuBPAL into Datalog

Besides translating QuBPAL queries to Datalog queries, the compiler also verifies some syntactic correctness properties. Among these, the compiler checks that the QuBPAL query is *range-restricted* [19], i.e., every variable in the SELECT statement of the query also appears in a positive literal in the WHERE statement of the query. Range-restriction ensures that every variable appearing in a Datalog rule also appears in a positive literal in the premise of the rule and, therefore, only ground answers will be returned. Then, the compiler re-orders literals in the premises of the target Datalog rules so that every variable occurring in a negative literal also occurs in a positive literal on its left. This re-ordering guarantees a safe evaluation of the query by using the top-down, left-to-right strategy of XSB (i.e., only *ground* negative queries are evaluated). Finally, the compiler performs some simple transformations with the goal of optimizing the performance of query evaluation. These optimizing transformations include the re-ordering of literals and the insertion of the '!' (*cut*) predicate to eliminate unproductive choices in the query evaluation tree.

3.3 Query Examples

In this section we present some examples of query over a ***BPKB***. We provide a natural language description of the query, the corresponding formulation according to QuBPAL and the translation into Datalog rules, where constants are enclosed in quotes.

Ex1. *In order to complete the design of a new BPS for processing purchase orders, it is needed to:*
Q1: Retrieve all sub-processes that contain a 'Financial Transaction' (to be also returned by the query), and end with an activity performed by a 'Shipping Company'.

SELECT <?p,?s,?e > ?x **WHERE** activity(?e::?c) AND t(?c,owl:allValuesFrom,ShippingCompany) AND t(?c,owl:onProperty,hasAgent) AND belongs(?x::FinancialTransaction,?p,?s,?e)
q(p,s,e,x):- σ(x,'FinancialTransaction'), belongs(x,p), t(c,'owl:allValuesFrom','ShippingCompany'), t(c,'owl:onProperty','hasAgent'), belongs(e,p), activity(e), σ(e,c), wf_subproc(p,s,e), belongs(x,p,s,e)

Ex2. *In order to verify the compliance of a BPS with respect to the enterprise policy,*
Q2. Retrieve the BP schemas where a 'Financial Transaction' is not a pre-condition for the 'Carrying' of any given product:

SELECT <?p> **WHERE** NOT precedence(FinancialTransaction,Carrying,?p)
q(p):- bpId(p,s,e), not precedence('FinancialTransaction','Carrying',p,s,e)

Ex3. *In the reengineering phase, to avoid concurrent executions of a 'Financial Transaction' and a 'Carrying' activity, it is needed to*
Q3. Retrieve the sub-processes, delimited by parallel gateways, containing a 'Financial Transaction' and a 'Carrying' activity on two different branches:

SELECT <?p,?s,?e> **WHERE** par_branch(?s,?s1,?s2) AND par_merge(?e1,?e2,?e) AND ({**SELECT** <?p,?s1,?e1> **WHERE** belongs(?x::FinancialTransaction,?p,?s1,?e1)} AND { **SELECT** <?p,?s2,?e2> **WHERE** belongs(?y::Carrying,?p,?s2,?e2)} OR {**SELECT** <?p,?s1,?e1> **WHERE** belongs(?x:: Carrying,?p,?s1,?e1)} AND {**SELECT** <?p,?s2,?e2> **WHERE** belongs(?y:: FinancialTransaction,?p,?s2,?e2)})
q(p,s,e):- belongs(s,p), par_branch(s,s1,s2), belongs(e,p), par_merge(e1,e2,e), wf_subproc(p,s,e), aux(p,s1,s2,e1,e2). aux(p,s1,s2,e1,e2):- σ(x,'FinancialTransaction'), belongs(x,p,s1,e1), wf_subproc(p,s1,e1), σ(y,'Carrying'), belongs(y,p,s2,e2), wf_subproc(p,s2,e2). aux(p,s1,s2,e1,e2):- σ(x,'Carrying'), belongs(x,p,s1,e1), wf_subproc(p,s1,e1), σ(y,'FinancialTransaction'), belongs(y,p,s2,e2), wf_subproc(p,s2,e2).

If we consider the eProcurement process fragment of Section 2:

- The answer to *Q1* contains the sub-process starting with *G1* and ending with *A5* and the event *E1;*
- The answer to *Q2* contains the *eProcurement* BPS, since it may happen that *A4* is executed before *E1*;
- *Q3* returns the sub-process starting with *G1* and ending with *G2.*

4 BPAL Query Platform

This Section describes the implementation of the proposed framework and presents a preliminary evaluation of the system performance[1].

4.1 Implementation

A prototype of the proposed framework has been implemented as a Java application, interfaced with the XSB logic programming and deductive database system [9] through the Interprolog library (*http://www.declarativa.com/interprolog*). The BPAL platform is depicted in Figure 4. On the left part of this figure, enclosed in a dotted rectangle, we have grouped the components involved in the *setup phase*, when the BPKB is built.

The process repository ***B*** is populated by process schemas modeled by business experts using a BPMS capable of exporting XPDL [11]; then it is translated into BPAL by means of the service *XPDL2BPAL*. The business reference ontology ***BRO*** is imported from an OWL-RL ontology by the service *OWL2LP* that translates the ***BRO*** into a set of ground facts in the triple notation. The reasoning over the ontology is supported by the rule-set ***OWLRL***, obtained by a translation of the OWL 2 RL/RDF rules [5]. The semantic annotation **Σ** is encoded as an OWL file too, and it is similarly imported into the *BPKB*. The parsing of OWL files is based on the Jena2 toolkit (*http://jena.sourceforge.net/*). Finally the *BPKB* is completed by the logic programs encoding the meta-model theory ***M***, the trace theory ***T*** and the dependency constraints ***D***. Having populated the *BPKB*, the reasoning tasks are performed by querying the knowledge base through *QuBPAL* queries

[1] The tool and the resources used for the evaluation are available at http://leks-pub.iasi.cnr.it/bpal.

that are translated into Datalog by the service *QBPAL2LP* and evaluated by the XSB engine. The computed results can be exported through the *XpdlWriter* module as an XPDL file, for its visualization in a BPMS and its further reuse. These components are enclosed in a dotted rectangle on the right part of Figure 4.

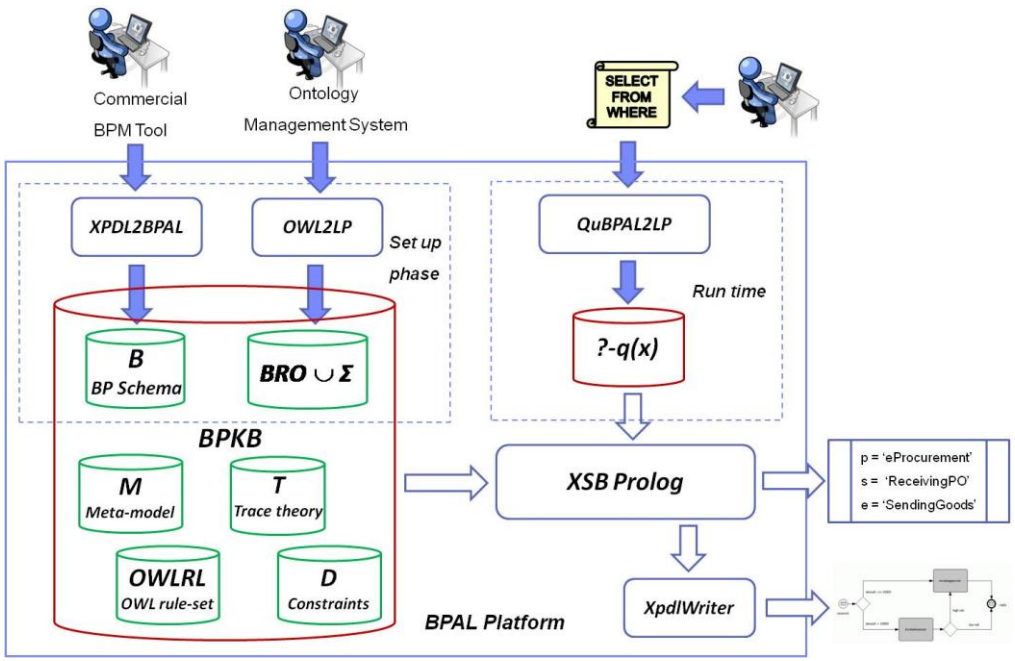

Fig. 4. Architecture of the BPAL Platform

4.2 Evaluation

In this section we report some results related to a preliminary evaluation of the system performance, to show the feasibility of the approach and its scalability. We run the tests described above on a contemporary desktop machine, equipped with an Intel Core2 E4500 CPU (2x2.20 GH) and 2GB of RAM.

The results are summarized in Table 6. Timings are expressed in seconds. We generated artificial XPDL files, describing three BPS repositories, **T1-T3,** of different size and structure. In the first part of Table 6 we report, for each repository, the number of BP schemas, the total size, i.e. the total number of flow elements, the total number of gateways and the size of the smallest and biggest BPS. We then randomly annotated such BP schemas with respect to the OWL translation of the SUMO ontology (available at http://www.ontologyportal.org/translations/SUMO.owl), containing 613 named concepts and 228 relations described by about 2200 triples. In particular for the annotation we used sub-classes of *sumo:Process.*

We tested first the set up phase, by importing into the platform each repository from an XPDL file, the ontology and the semantic annotation from OWL. Results are sown in the middle part of Table 6.

We finally performed the queries introduced in Section 3.3 against each repository. We report, for each run, the number of results obtained and the total time spent for the evaluation, including the QuBPAL query translation (*QuBPAL2LP*), the communication

overhead between Java and XSB and the export of the results as a new XPDL file (*XpdlWriter*).

Table 6. Evaluation Results

Test Data Sets						
	Nr. of BPS	*Tot. Size*	*Nr. of Gateways*		*Min BPS Size*	*Max BPS Size*
T1	50	13583	4206		247	320
T2	100	21692	6702		189	259
T3	200	32936	10126		142	207
Set up phase Evaluation						
	BPS Import		**BRO Import**		**Σ Import**	
	XPDL2BPAL	*XSB Compile*	*OWL2LP*	*XSB Compile*	*OWL2LP*	*XSB Compile*
T1	4.6	7.6	1	0.7	2.1	1.2
T2	7.8	12.8	1	0.7	3	2.1
T3	17.3	20	1	0.7	4.5	3.3
Run Time phase Evaluation						
	Q1		**Q2**		**Q3**	
	Nr. of Res.	*Time*	*Nr. of Res.*	*Time*	*Nr. of Res.*	*Time*
T1	27	2.6	14	0.9	14	10.5
T2	22	3.2	26	1.7	15	13.2
T3	68	3.4	38	3.1	10	11.6

5 Related Works

Some techniques for semantic annotation first developed in the context of the semantic web have been extended to business process management. Relevant work in this field has been done within the OWL-S [12] and WSMO [13] initiatives. Both approaches make an essential use of ontologies to describe a web service from several perspectives: (a) input, output, pre-conditions and post-conditions (OWL-S *Profile*, WSMO *Capability* and *Goal*); (b) the service behavior, through its representation as a process workflow (OWL-S *Service Model*, WSMO *Orchestration* and *Choreography*); and (c) the grounding of the service to its implementation (OWL-S *Grounding*, WSMO *Mediator*). In [14] the annotation of process models has been proposed to support the verification of semantic constraints and structural requirements involving both the knowledge about the domain and the process structure. [15] presents a reasoning framework where several ontologies model functional, behavioral, organizational and informational perspectives and the entities of a business process are then represented as instances of such ontologies. Unlike the aforementioned works, the execution semantics of a BPAL BPS is formally defined as the set of its correct traces. The latter can be explicitly represented and manipulated, and several verification tasks can also be performed. Furthermore, the *BPKB* provides an environment to effectively reason about and query in an homogeneous framework the ontological description, the workflow specification and the dynamic properties of a modeled process.

BP-QL [16] is based on an abstract representation of the BPEL (Business Process Execution Language) standard. It is a visual language, formally based on graph grammars, that allows one to query the process *specification* (i.e. the graph representation of a process workflow) of a BPEL process ignoring the run-time semantics of certain constructs such as choice or parallel execution. A graph matching approach is also shared by BPMN-Q [17], a

visual language based on BPMN. BPMN-Q also supports some templates of constraints regarding the execution semantics (e.g., precedence, response), which are verified by a model checker. This model checker takes as input a temporal logic translation of the queries and a finite state model generated from the BPMN specification. With respect to these approaches, we propose a uniform framework that allows us to deal with different reasoning tasks and combination thereof with a uniform semantics, avoiding the burden of integrating heterogeneous formalisms and tools. The use of a logic-based uniform approach guarantees *i)* good complexity properties (the translation into Datalog queries ensures polynomial data complexity [9]), *ii)* considerable scalability, *iii)* easy integration of additional reasoning services through a suitable translation to logic programming.

Concurrent Transaction Logic (CTR) [18] is a formalism for declarative specification, analysis, and execution of transactional processes. CTR formulas extend Horn clauses and their standard model-theoretic and operational semantics. Unlike CTR formulas, the *BPKB* can be directly viewed as an executable logic program. Hence any component can be queried by any Prolog system without the need for a special purpose evaluator. This implies that both BPAL traces and BP schemas are explicitly represented and can be directly analyzed and manipulated, in conjunction with other knowledge representation applications, e.g., computational ontologies. Furthermore, BPAL is derived from graph based modeling languages like BPMN, and queries are written in essentially the same way as process specifications.

Finally, program analysis and verification techniques have been applied to the analysis of processes behavior (see [19,20] for a sample). These works are based on model checking techniques where queries, formulated in some temporal logics, test if the executions of the process satisfy a certain property. Despite the efficiency of these approaches, severe limitations arise when they have to be combined with other techniques in order to verify properties encompassing also other conditions beyond the ordering/ presence/ absence of tasks in the possible execution of the process.

6 Conclusions

In this paper we presented a framework conceived to complement existing BPMS by providing advanced reasoning services. The proposed solution is based on a synergic use of ontologies to capture the semantics of a business scenario, and a business process modelling framework (BPAL), to represent the underlying application logic. Both frameworks are seamlessly connected thanks to their grounding in logics (Logic Programming) and therefore it is possible to apply effective reasoning methods to query the BPKBs encompassing the two. This solution has been conceived to be easily adopted in any context where a (commercial) tool for BPMS is already in use, since the BPAL platform smoothly integrates with any BP modeling environment capable of representing the processes in BPMN, and in particular in its XPDL linear form.

We are working to extend the proposed BPAL knowledge representation framework in several directions. First of all the query evaluation process can be strongly optimized through more elaborated transformations achieved by exploiting sophisticated program transformation techniques [21]. Then, we want to remove the restriction to blocked processes, to handle any graph-structured BP schema, and hence the verification of

behavioral properties over (possibly) infinite sets of traces. On an engineering ground, we are exploring the problem of manipulating, merging and aggregating a set of business process fragments in the contexts of BP Composition and BP Re-engineering. Finally, we are working to improve the software platform, in particular on the user interface and on the level of automation in supporting the semantic annotation of BP schemas.

7 References

1. Hepp, M., Leymann, F., Domingue, J., Wahler, A., Fensel., D.: Semantic business process management: A vision towards using semantic web services for business process management. Proc. ICEBE 2005, pp. 535--540. IEEE Computer Society, Los Alamitos, 2005.
2. Lloyd, J.W.: Foundations of Logic Programming. Springer-Verlag, Berlin, 1987. 2nd Ed.
3. De Nicola, A., Missikoff, M., Proietti, M., Smith, F.: An Open Platform for Business Process Modeling and Verification, International Conference on Database and Expert Systems Applications. Proc. DEXA 2010. LNCS 6261, pp. 66--90, Springer, 2010.
4. OMG: Business Process Model and Notation. Version 2.0, August 2009, http://www.omg.org/spec/BPMN/2.0.
5. OWL 2: Profiles, http://www.w3.org/TR/owl2-profiles.
6. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description Logic Programs: Combining Logic Programs with Description Logic. Proc. 12th Int. Conf. on World Wide Web, pp. 48--57, ACM Press, 2003.
7. Missikoff M., Proietti M., Smith F.: Linking Ontologies to Business Process Schemas. IASI-CNR, Technical Report 10-20, 2010.
8. De Nicola, A., Missikoff, M., Smith, F.: Towards a Method for Business Process and Informal Business Rules Compliance. *Journal of Software Process: Improvement and Practice*. To Appear.
9. The XSB Logic Programming System. Version 3.1, Aug. 2007, http://xsb.sourceforge.net.
10. Abiteboul, S., Hull, R., Vianu, V.: *Foundations of Databases*, Addison-Wesley, 1995.
11. XPDL 2.1 Complete Specification, Oct. 2008, http://www.wfmc.org/xpdl.html.
12. W3C: OWL-S, Semantic markup for web services. 22 November 2004, http://www.w3.org/Submission/OWL-S/.
13. Roman, D., et. al. : Web Service Modeling Ontology. *Applied Ontology*, 1(1): 77--106, IOS Press, 2005.
14. Di Francescomarino, C., Ghidini, C., Rospocher, M., Serafini, L., Tonella, P., Semantically-aided business process modeling. Proc. ISWC 2009, pp. 114--129, Washington DC, USA, 2009.
15. Markovic, I. Advanced Querying and Reasoning on Business Process Models. Proc. BIS 2008. LNBIP 7, pp.189--200, Springer, 2008.
16. Beeri, C., Eyal, A., Kamenkovich, S., and Milo, T. Querying business processes with BP-QL. *Information Systems.* 33, 6 (Sep. 2008), 477--507.
17. Awad, A., Decker, G., Weske, M.: Efficient compliance checking using BPMN-Q and temporal logic. Proc. BPM 2008. LNCS 5240, pp. 326--341. Springer, 2008.
18. Roman, D. and Kifer, M.: Reasoning about the Behavior of Semantic Web Services with Concurrent Transaction Logic. Proc. VLDB 2007, pp. 627--638.
19. Fu, X., Bultan, T., and Su, J.: Analysis of interacting BPEL web services. Proc. Int. World Wide Web Conf., pp. 621--630, ACM Press, 2004.
20. Liu, Y., Muller, S., Xu, K.: A static compliance-checking framework for business process models. *IBM Systems Journal* 46 (2007) 335--361.
21. Pettorossi, A. and Proietti, M.: Transformation of Logic Programs: Foundations and Techniques. *Journal of Logic Programming* 1994: 19, 20: 261--320.