A Business Process Knowledge Base for Composite Services Development[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,
fabrizio.smith}@iasi.cnr.it

Abstract. In this paper we present a semantic approach to complex service composition. The proposal is based on a synergic use of an ontological framework (OPAL), to capture the semantics of a business scenario, and a business process modelling framework (BPAL), to represent the underlying application logic. Both frameworks are grounded in a logic-based formalism and therefore it is possible to apply effective reasoning methods to make inferences over a BPKB (Business Process Knowledge Base) stemming fron the fusion of the two. Particular attention is dedicated to the BPAL framework, based on a MOF architecture, allowing a comprehensive modelling method that spans from the ground level (BP traces), to the BP schema modelling level, to the meta-modeling level (design princliples). On top, the meta-metamodelling level is represented by the logic-based formalism (Horn rules). Finally, we show how the BPKB can be queried, to support the complex service composition, and the complex service can be checked for correctness.

Keywords: business process, BPMN, horn logic, BPAL, query language, composite service, orchestration.

1 Introduction

A composite service is described as a process schema that put together other basic or composite services. Service orchestration relates to the execution of a business process that in turn, after a suitable 'packaging', becomes a composite service. A service orchestration is then modeled by a graph (the flow structure), which defines the order of execution among the nodes in the process. In general, the graph may include *activities*, *gateways*, and *event* nodes, where activity nodes represent the invocation of a basic or composite services, gateways specify the alternatives and rules controlling the execution flow, while event nodes enable service processes to send and receive several types of events. A well-known and widely adopted

[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).

executable language for service orchestration is BPEL4WS [1], but here, as an exemplary notation, we refer to BPMN [2], since it is more intuitive, provide a graphical notation, and its block structured subset can be easily translated to BPEL4WS.

The focus of this paper is on complex service composition. In particular, we propose a methodological framework and a tool that support the service designer in assembling a complex service, by defining the business logic, and verifying if the assembled complex service is compliant with a number of pre-defined properties.
In this context, we intend to offer also a tool that manages a repository of business process schemas (BPS), supporting the service designer in searching for the BPS (or fragments of it) and, once the desired BPS has been assembled, in verifying its correctness (with respect to certain criteria, see later). In this frame, we consider particularly relevant the *reuse of BPS*. A business expert should query a repository of composite services in order to retrieve schemas (or process fragments, or atomic components) to be used in the design of a new service orchestration, specifying features and properties that the retrieved artifacts must exhibit.
In this scenario, we consider important that the proposed method exhibits the following features:

- Strong support in capturing the complexity of the business reality: besides the behavior of a business process, i.e., the execution flow represented by the activity sequencing, there are other relevant aspects of a structural nature regarding the domain in which the process take place, such as actors associated to activities, managed objects, and their relationships;
- Grounding of the modeling framework to a formal and expressive representation language, in order to avoid ambiguities and allow reasoning over the process descriptions;
- Providing a reasoning mechanism to prove the correctness of a BPS;
- Providing a query language sufficiently expressive to formally capture the user requests;
- Providing a mechanism to evaluate queries in an effective manner;
- Providing a reasoning mechanism to prove the correctness of the query answers.

In Figure 1 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. In the meanwhile, the supplier sends a gift to the buyer if she/he is classified as ‘golden client.’ After receiving the payment clearance from the bank, the supplier sends the goods to the buyer.

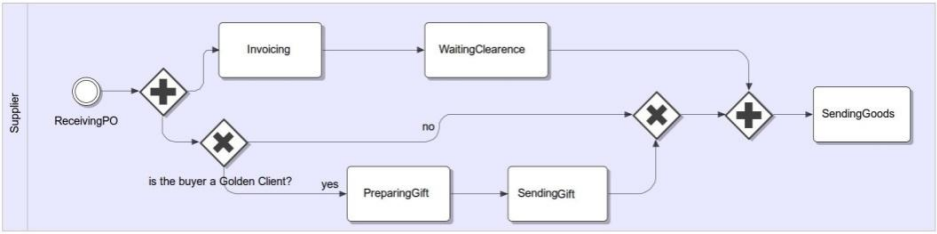

Fig. 1. BPMN specification of a fragment of an eProcurement example

2 Business Process Knowledge Representation

An effective design of a business processes requires a complex analysis of the business reality and the modeling of different kinds of knowledge. Primarily, the behavioral knowledge, but also the structural knowledge regarding the domain in which the process take place, such as actors associated to activities, managed objects, and their relationships.

In order to provide a uniform and formal representation (suited for automatic reasoning) of both behavioral and structural knowledge we rely on an expressive, logic-based representation technique.

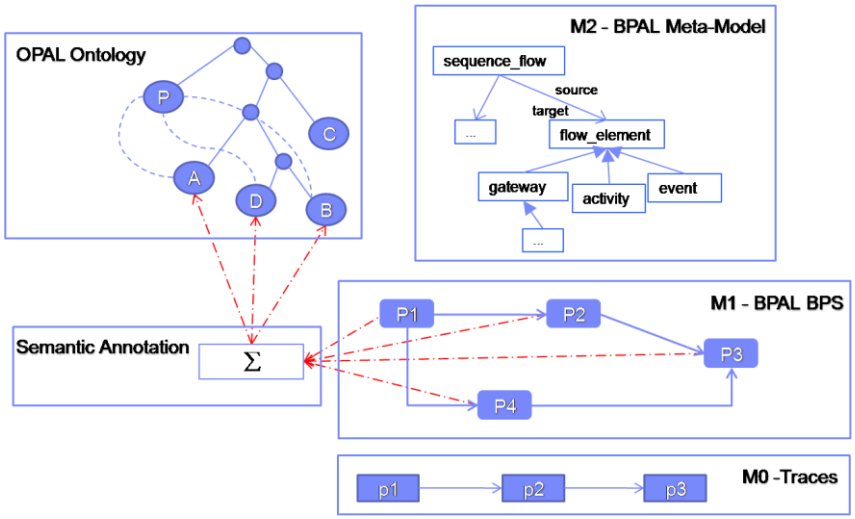

Fig. 2. Business Process Knowledge Base

Furthermore, we intend to systematically address the global modelling framework in a unitary vision. To this end, the overall approach that we assumed is based on the MOF paradigm [3] with the four levels sketchily reported below.

M3: Meta-metalevel. The top level is represented by the logical formalism that we apply to describe the following levels. In particular we adopted Horn Logic,

seen its wide adoption and the mature technological support provided by the numerous Prolog systems existing in our community.

M2: Metalevel. Here we specify the basic formation rules that guide the complex service designer in the specification of the BPS.

M1: BPS. This is the modeling level where the service designer actually define the diagram that represents the business logic of the complex service.

M0: BP trace. This is the ground level, used to model the traces that are produced by the execution of a complex service, in accordance with the corresponding BPS.

For the formalization of the framework we use standard notions of first order logic and logic programming [4].

The rich knowledge about the business processes and the context they operate in is stored in a *Business Process Knowledge Base (BPKB)* depicted in Figure 2. In the rest of this section we present the main components of the BPKB, namely: *i)* OPAL (Object, Process, Actor modelling Language), an ontological framework for the structural representation of a business domain; *ii)* BPAL (Business Process Abstract Language), to represent the behavioural knowledge of a business process (metamodel, schema and traces); and *iii)* the Semantic Annotation, constituting a bridge among the aforementioned components.

2.1 OPAL

OPAL [5] 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.

Therefore, the top level concepts are: *i) opal:Process*, representing any business activity or operation aimed at the accomplishment of a business goal, operating on a set of business objects; *ii)* o*pal:Actor*, representing active elements of a business domain, able to activate, perform, or monitor a business process; *iii) opal:Object*, representing an entity on which a business process operates. As shown in [5], a significant core of an OPAL ontology can be formalized by a fragment of OWL, relying within the OWL-RL profile. OWL-RL [6], is an OWL subset designed for practical implementations using rule-based technologies such as logic programming [7].

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*, *owl*, *xsd*, plus *opal* for the pre-defined primitives of OPAL. For the semantics of an OWL-RL ontology we refer to the axiomatization described in [6] by a set of FOL rules over the predicate *T* (OWL 2 RL/RDF rules). In Table 1 some axioms of a business reference ontology ***BRO*** related to the *eProcurement* process of Figure 1 are reported.

T(bro:Supplier, rdfs:subClassOf, opal:Actor)	The *Supplier* is an actor
T(bro:Delivering, rdfs:subClassOf, opal:Process) T(bro:SendingGift, rdfs:subClassOf, bro:Delivering) T(bro:SendingGoods, rdfs:subClassOf, bro:Delivering)	*SendingGoods* and *SendingGift* are specializations of the *Delivering* process
T(bro:Delivering, rdfs:subClassOf, bro:r1) T(bro:r1, owl:allValuesFrom, bro:Supplier) T(bro:r1, owl:onProperty, bro:PerformedBy)	Every *Delivering* is performed by a *Supplier*
T(bro:Gadget, rdfs:subClassOf, opal:Object) T(bro:Product, rdfs:subClassOf, opal:Object)	*Gadgets* and *Products* are business object of the domain
T(bro:SendingGoods, rdfs:subClassOf, bro:r2) T(bro:r2, owl:allValuesFrom, bro:Product) T(bro:r2, owl:onProperty, opal:OperateOn)	A *Sending Good* activity involves only *Products*
T(bro:SendingGift, rdfs:subClassOf, bro:r3) T(bro:r3, owl:allValuesFrom, bro:Gadget) T(bro:r3, owl:onProperty, opal:OperateOn)	A *Sending Gift* activity involves only *Gadgets*

Table 1. Excerpt of a business reference ontology

2.2 BPAL

The Business Process Abstract Language (BPAL) [8] is a logic-based language (grounded in Horn Logic) 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 [1].

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 $p(C_1, \ldots, C_n)$, where $p \in Pred$ and $C_1, \ldots, C_n \in Entities$.

The entities occurring in a BP are represented by a set of unary predicates. They are illustrated in Figure 3, and organized into a hierarchy showing the BPAL predicates together with the corresponding BPMN notation.

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): *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): *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;
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;
bpId(id,start,end): to assign the process identifier *id* to a process given its *start* and *end* events.

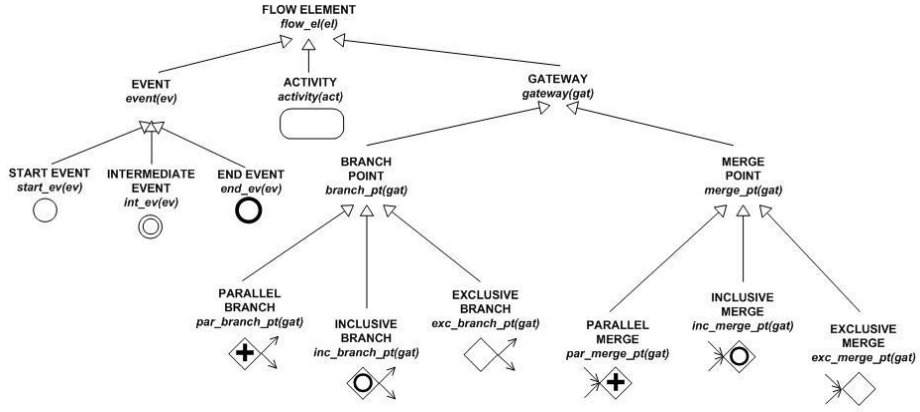

Fig. 3. Hierarchy of the BPAL unary predicates

2.3 Semantic Annotation

A Business Reference Ontology is intended to provide a semantic representation of the business context in which the business processes take place. A semantic annotation is a correspondence between elements of the BPS and elements of the BRO achieved with the 'sigma' predicate. In our case, the *Semantic Annotation* of a BPAL BPS consists of a set of assertion of the form *σ(Act,Conc)*, where *Act* is a constant used to denote an activity or an event of a BPAL BPS, and *Conc* is a constant used to denote a concept defined in the OPAL ontology. This relation allows a bridge to be built between an OPAL ontology and a BPAL BPS, specifying the meaning of the entities of a business process in term of a suitable conceptualization of the domain of interest. The definition of σ mainly requires that: *i)* σ is preserved by the *subclass* relation, i.e.$\sigma(x,y) \wedge T(y, rdfs{:}\, subclassOf, z) \rightarrow \sigma(x,z)$, and *ii)* every activity or event must be annotated with a sub class of *opal:Process.*

The process fragment of Figure 1 is reported in Table 2 encoded as a BPAL BPS. In the BPAL translation we assume to have an available reference ontology ***BRO*** in order to perform also the semantic annotation step. To keep the notation lightweight

we use assertions of the form *activity (A1::bro:Invoicing)* to denote that the activity *A1* is annotated with the concept *Invoicing*, i.e. $\sigma(A1, bro{:}\,Invoicing)$.

int_ev(E1::bro:ReceivingPO)	*par_branch_pt(G1)*	*exc_branch(G3,G4,A3)*
activity(A1:: bro:Invoicing)	*par_merge_pt(G2)*	*exc_merge(G3,A4,G4)*
activity(A2:: bro:WaitingClearence)	*exc_branch_pt(G3)*	*seq(E1,G1)*
activity(A3:: bro:PreparingGift)	*exc_merge_pt(G4)*	*seq(A1,A2)*
activity(A4:: bro:SendingGift)	*par_branch(G1,A1,G3)*	*seq(A3,A4)*
activity(A5:: bro:SendingGoods)	*par_merge(A2,G4,G2)*	*seq(G4,G5)*

Table 2. Annotated BPAL BPS of the eProcurement example.

2.4 BPAL Metamodel

In order to provide a clear modelling guidance, we explicitly introduce a specification of a business process metamodel [8] (the level M2 in the MOF hierarchy). The meta-model of BPAL is defined by means of a first order logic theory ***M***, which specifies when a business process schema is well-formed, i.e., it is correct from a syntactical point of view.

The theory ***M*** consists of two sets of formulas: a set ***K*** of first order formulas, called *schema constraints*, and a set ***F*** of Horn clauses, called *formation axioms.*

Schema constraints are formulas expressing:

1. the relationships among BPAL unary predicates; e.g. that activities, events, and gateways belong to pairwise disjoint sets, i.e.: $activity(x) \rightarrow \neg\, event(x) \wedge \neg\, gateway(x)$;
2. the typing of the relational predicates; e.g. for the predicate *par_branch*, we have $par\_branch(x,l,r) \rightarrow par\_branch\_pt(x) \wedge flow\_el(l) \wedge flow\_el(r)$;
3. an unambiguous specification of the precedence relations; e.g. for the *seq* predicate we can specify at most one successor and at most one predecessor of any flow element: $seq(x,y) \wedge seq(x,z) \rightarrow y{=}z,\ \ seq(x,z) \wedge seq(y,z) \rightarrow x{=}y$.

Formation Axioms provide the guidelines for building a well-formed BPS. The main assumption imposed by the BPAL meta-model is the structuredness. 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 an event, 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).

Then, to verify if a process respects such restriction, ***F*** defines the predicates:
wf_proc(*bpId*), which holds if the business process *bpId* is *well-formed* (i.e., structured);
wf_subproc(bpId,start,end), which define the well-formedness of the sub-process starting in *start* and ending in *end*, i.e. the sub-process is an elementary or complex block according to the above definition.

Furthermore ***F*** defines properties regarding the BPS, like:

belongs(flow_el,bpid) which holds if *flow_el* belongs to set of flow elements of the process *bpId*;
belongs(flow_el,bpid,start,end) which holds if *flow_el* belongs to set of flow elements of the sub-process of *bpId*, starting in *start* and ending in *end*.

We are now ready to give a definition of the well-formedness of a BPS ***B1.*** We say that ***B1*** is *well-formed* if:

(i) every schema constraint *C* in ***K*** can be inferred from ***B1***$\cup$***F***, and
(ii) *wf_proc*(*bpId*) can be inferred from ***B1***$\cup$***F***.

2.5 BPAL Traces

An execution (or instance, or enactment) 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 *Entities* (see Section 2.2). Thus, a possible execution of a business process is a sequence $[s_1, s_2, \ldots, s_n]$, called a *trace*, where $s_1, s_2, \ldots, s_n \in$ *Step*. The *instance* relation between steps and activities (or events) is specified by a binary predicate *stepOf(step,activity*). For example, *stepOf(RQ, ReceivingQuotation)* states that the step *RQ* is an instance of the *ReceivingQuotation* activity.

A trace is *correct* w.r.t. a well-formed business process schema ***B1*** if it is conformant to ***B1*** according to the intended semantics of the BPAL relational predicates (as informally described in Section 2.2). Below we list two correct traces of the process fragment of Table 2 (the instances are identified by small letters corresponding to the capital letters of the corresponding activity name):

- [*e1,a1,a2,a5*]
- [*e1,a3,a1,a4,a2,a5*]

The trace semantics of a BPS [8] is defined by a set ***T*** of Horn clauses, called *trace axioms*, which can also be viewed as rules for constructing correct traces. ***T*** defines the predicates

1. *trace*(*t,bpId*), which holds if *t* is a correct trace of the process *bpId*;
2. *sub_trace(s,t,e):* which holds if *t* is a correct sub-trace from *s* to *e.*

We say that a trace *t* is *correct* w.r.t. ***B1*** if *trace*(*t,bp*) can be inferred from ***B1***$\cup$***T***. These rules have a double nature, since they can be used to check correctness but also to generate correct traces.

At the trace level we can formalize *dependency constraints*, expressing the dependencies among tasks (events or activities) in the possible executions of the modeled process. We report here some examples of constraints, and their formalization within ***T***, where *i)* the arguments *s* and *e* limit the scope of the constraint, considered within the sub-process starting in *s* and ending in *e*, and *ii) member(s,t)* holds is *s* is a step in *t.*

Precedence: a task A precedes B if every execution of B follows the execution of A.

$$prec(a,b,s,e) \equiv_{def} \forall t1,t2,s1,b1,e1\ (stepOf(s1,s) \wedge stepOf(b1,b) \wedge stepOf(e1,e) \wedge sub\_trace(s1,t1,b1) \wedge sub\_trace(b1,t2,e1) \rightarrow \exists a1\ (stepOf(a1,a) \wedge member(a1,t1)))$$

Response: a task B responds to A if every time A is executed, activity B has to be executed after it.

$$resp(a,b,s,e) \equiv_{def} \forall\, t1,t2,s1,a1,e1\ (stepOf(s1,s) \wedge step(a1,a) \wedge stepOf(e1,e) \wedge sub\_trace(s1,t1,a1) \wedge sub\_trace(a1,t2,e1) \rightarrow \exists\, b1\ (stepOf(b1,b) \wedge member(b1,t2)))$$

MutualExclusion: A and B are not compatible and cannot be executed together.

$$mutex(a,b,s,e) \equiv_{def} \neg\exists\, t,s1,e1,a1,b1\ (stepOf(s1,s) \wedge stepOf(e1,e) \wedge stepOf(a1,a) \wedge step(b1,b) \wedge sub\_trace(s1,t,e1) \wedge member(a1,t) \wedge member(b1,t))$$

To conclude the section we give the formal definition of Business Process Knowledge Base:

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

where: ***BRO*** is an OPAL Business Reference Ontology, ***M*** is the Meta-Model theory, ***T*** is the Trace theory, ***B*** is a set of BPAL Business Process Schemas, and **Σ** is the Semantic Annotation. It is worth noting that ***BPKB*** can be translated in a straightforward way into a logic program in order to be effectively used for reasoning within a Prolog environment.

3 Querying a Business Process Knowledge Bases

Business Processes play a growing role in business realities and they are seen as important assets for organizations. In a near future, 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 desiderd properties hold. In the BPAL framework we can capture several types of queries, both at intentional and extensional level.

Queries over BP schemas. The execution semantics of certain constructs is not considered (e.g., gateways), but a BP is considered as a graph that satisfies some properties regarding the flow elements (*activities*, *events*, *gateways*) and their relationships (*sequence flows*). 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. Queries of this type are based on the predicates introduced by BPAL (Section 2.2) and by the meta-model theory ***M*** (Section 2.4).

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* (e.g. the *dependency constraints* introduced in Section 2.5). Queries of this type are based on the predicates introduced by the trace theory ***T*** (Section 2.5).

Queries over the Business Ontology. Here the focus is on the domain entities (processes, actors, objects) and their relationships. Queries of this type are based on

the generic ternary predicate *T* used for the definition of the business reference ontology.

In this scenario the role of the semantic annotation **Σ** is orthogonal to the above query classification, since it basically allows us to express queries in terms of the ontology vocabulary, decoupled from the business processes. This gives a great advantage in particular when an enterprise has to deal with a huge number of business processes. In fact, in such a scenario, it is possible to formulate queries in general terms related to the specific domain that the ontology describes, without knowing exactly what are the BPs it will impact on.

3.1 Query Language

In order to provide the user with a simple and expressive query language that does not require to understand the technicalities of the underline engine, we propose a simple abstract syntax, that can be directly translated into Prolog [4] rules.

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 the variable *?x,* 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 [*?x1,….,?xn*] <*?bpId*> <*?bpId,?start,?end* >
FROM <*bpId*> | <*bpId,start,end*> | *
WHERE *comparison_predicate*

The **SELECT** statement defines the output of the query evaluation, i.e. the following target list:

- a list [*?x1,….,?xn*] of variables occurring in the WHERE statement;
- a BPS, denoted by *<?bpId>;*
- a sub-process of a BPS, denoted by the triple *<?bpId,?start,?end>*.

The **FROM** statement indicates the process(es) from which data is to be retrieved:

- a particular BPS, *<bpId >;*
- a particular sub-process of a BPS, *<bpId,start,end >;*
- the whole repository, *.

In the **WHERE** statement it can be specified an expression which restricts the data returned by the query. The *comparison_predicate* is a sentence built from:

- the set of the ***BPKB*** predicates, defined in the:
    - BPAL, e.g., *flow_el(el), bpId(p,s,e)*;
    - meta-model ***M***, e.g., *wf_proc(bpId), belongs(el,bpId)*;
    - trace theory ***T***, e.g., *prec(a,b,s,e), mutex(a,b,s,e)*;
    - OPAL ontology, i.e., *T(s,p,o).*

  where arguments are:
    - *semantically typed variables. (i.e.: ?x::conceptId);*

  - *constants* denoting entities in the BPKB.
- the connectives AND, OR, NOT, = with the standard logic semantics.

As stated in Section 2 a ***BPKB*** can be directly encoded as a logic program, and used within a Prolog engine for evaluate conjunctive queries, formulated in the Prolog syntax as rules 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 $p_{1,}..,p_n$ are predicates defined in the ***BPKB,*** $q(\vec{x})$ is the goal 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 $\overrightarrow{x_i}$.

3.2 Query Examples

In this section we present some examples of query over a ***BPKB***. We report a natural language description of the query, the corresponding formulation according to the language described in the previous section and the translation into Prolog rules.

Ex1: *In order to avoid multiple dispatching of products to the same supplier within the processing of a purchase order,*
Q1. Retrieve the processes that contain more than one "Delivering" activity:
SELECT <*?p*>
FROM *
WHERE belongs(?x::Delivering,?p) AND belongs(?y::Delivering,?p) AND NOT ?x=?y

q(p):- belongs (x,p), σ (x,Delivering), belongs(y,p), σ(y, Delivering), x/=y.

Ex2. *In order to complete the composition of a service for processing purchase orders, it is needed to:*
Q2: Retrieve any sub-process that starts with the receiving of a purchase order, contains an activity of invoicing and ends with the delivery of the goods
SELECT <*?p,?s,?e* >
FROM *
WHERE flow_el(?s::ReceivingPO) AND flow_el(?e::Delivering) AND belongs(?s,?x::Invoicing,?s,?e)

q(p,s,e):- wf_sub_proc(p,s,e), σ(s,ReceivingPO), σ(e,Delivering), belongs(x,p,s,e), σ(x,Invoicing), belongs(s,p), belongs(e,p).

Ex3. *In order to verify the compliance of the BPS P with respect to the enterprise policy, we need to*
Q3. Verify if in the BPS the receiving of the bank clearance may follow any kind of product delivering:
FROM <P>
WHERE bpId(P,?s,?e) AND prec(?x::WaitingClearence,?y::Delivery,?s,?e)

q:- bpId(P,s,e), not aux_ q(s,e).
aux_q(s,e):- σ(x, WaitingClearence), σ(y, Delivering), not prec(x,y,s,e).

If we consider the eProcurement process fragment of Section 2.3 annotated with the business domain ontology ***BRO*** of Section 2.1,

- *Q1* retrive the eProcurement process, since both *SendingGift* and *SendingGoods* are *Delivering* activities;
- *Q2* match with the sub-process delimited by *e1* and *a5;*
- The answer of *Q3* is *false*, since it may happen that *SendingGift* is executed before *WaitingClearence.*

3.3 BPAL Query Platform

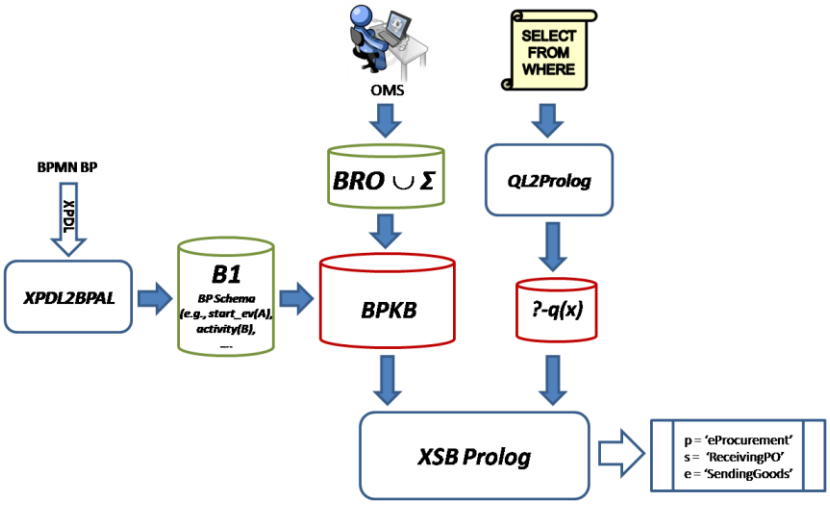

Fig. 4. Logical Architecture of the Query Platform

The prototype of the proposed framework has been implemented as a Java application, interfaced with the XSB and deductive database system [9]. The ***BPKB*** is fed as shown in Figure 4. The process repository is populated by process models represented by XPDL [10](XML Process Definition Language) and translated into a set of BPAL ground facts by means of the service XPDL2BPAL. The business ontology is imported from the OPAL Ontology Management System (OMS) Athos[11], that has been extended to allow the annotation of BP schemas and to export OPAL ontologies directly in the triple notation. To implement the terminological reasoning over the ontology a subset of the OWL 2 RL/RDF rules [6] have been included in the ***BPKB***. Finally the queries, expressed in the abstract syntax described in Section 3.1 are translated to Prolog rules and evaluated as goals against the ***BPKB***.

4 Related Works

Our work is related to two main research areas, namely the semantic enrichment of service models by means of ontology based approaches and the querying of business processes.

Semantic Web Services. 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) "what a service does", in terms of input, output, pre-conditions and post-conditions (OWL-S *Profile*, WSMO *Capability* and *Goal*); (b) "how a service works", where the service behavior is modeled as a process workflow (OWL-S *Service Model*, WSMO *Orchestration* and *Choreography*); and (c) the grounding of the modeled service to detailed specifications of message formats, protocols, and so forth, normally expressed in WSDL (OWL-S *Grounding*, WSMO *Mediator*). With respect to the BPAL framework, the main difference is in the modeling of the service behavior (b). Conversely to OWL-S and WSMO, the execution semantics of BPAL is formally defined and several verification tasks are provided. Furthermore is also defined an environment to reason with and query both the ontological description of the services and their dynamic properties. Finally, regarding (a), there are inherent differences in the ontological representation of a service, since in OWL-S and WSMO the notion of process instance (i.e., a particular execution of a service), is not modeled, while the ground level is constituted by the concrete services (possibly an implementation of a Web Service) that the ontology models.

Business Process Query. BP-QL [14] 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 to query the process *specification* (i.e. the graph representation of a process workflow) of a BPEL process, rather than possible runs, ignoring the run-time semantics of certain constructs such as choice or parallel execution. A similar perspective is also shared by [15] and [16], where queries are posed over the process definition from a structural point of view and the evaluation is performed by searching a match to the query graph in the process graph. With respect to these approaches, we allow more expressive queries, where also the dynamic properties of the process are taken into account, together with the domain knowledge provided by the ontology.

We end this section with some relevant works in the emerging area of Semantic Business Process Management [17], that aims at improving the level of automation in the management of business processes by adopting the most significant results from the area of semantic web. To this end the annotation of process models [18,19] has been proposed to support the verification of semantic constraints and structural requirements involving both knowledge about the domain and the process structure. [20] 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 on ontologies. The semantic annotation of a process model, at an abstract level, can be seen as a further

application of BPAL, since a core (blocked) subset of BPMN and the OWL-RL profile are fully supported by the framework for reasoning.

5 Conclusion

In this paper we presented the main ideas of a platform conceived to support the composition of complex services. The proposed platform consists of several parts: (i) an ontological framework, OPAL, to capture the semantics of the business scenario; (ii) a business process modeling framework, to capture the application logic; (ii) a reasoning engine, based on Horn logic, that operates on the two above structures in an integrated way; (iv) a BP query language, developed on top of the reasoning engine; finally, (v) a verification mechanism, tightly connected to the latter.
The paper presents the first version of the proposed solution that we intend to elaborate further, working in several directions:

- Extend the framework to handle any graph-structured process schemas (arbitrary looping, no blocked assumption) and hence the verification of behavioral properties (e.g. *dependency constraints*) over (possibly) infinite sets of infinite traces.
- The literature has been also investigated the query and verification at *run time* (i.e. performed over a running instance of the process during its enactment) and the *a-posteriori* analysis (i.e., log mining) over the information stored during the execution. The extension of the proposed framework to allow querying of running and executed traces is considered as a future work.
- Business Rules (BRs). In real world applications the operations of an enterprise is regulated by a set of BPs that are often integrated by specific business rules. We intend to develop an extended the framework where BPs and BRs are integrated and jointly analyzed to check if, for instance, there are processes that violate a business rule.
- 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

6 References

1. Thatte, S. (Ed.). Business Process Execution Language for Web Services Version 1.1. http://www.ibm.com/developerworks/library/specification/ws-bpel. 2003.
2. OMG: Business Process Model and Notation. Version 2.0, August 2009, http://www.omg.org/spec/BPMN/2.0.
3. OMG, (2006), Meta Object Facility (MOF) Core Specification V2.0, http://www.omg.org/docs/formal/06-01-01.pdf.
4. Lloyd, J.W.: Foundations of Logic Programming. Springer-Verlag, Berlin, 1987.

5. D'Antonio, F., Missikoff, M., Taglino, F.: Formalizing the OPAL eBusiness ontology design patterns with OWL. In the 3rd I-ESA Conference, 2007.
6. OWL 2: Profiles, http://www.w3.org/TR/owl2-profiles.
7. B. N. Grosof, I. Horrocks, R. Volz, S. Decker: Description Logic Programs: Combining Logic Programs with Description Logic, in: Proceedings of the 12th International Conference on World Wide Web, ACM, 2003.
8. Missikoff M., Proietti M., Smith F.: A Logic-Based Method for Business Process Knowledge Base Management. SEBD 2010, Rimini, Italy, June 2010.
9. The XSB Logic Programming System. Version 3.1, Aug. 2007, http://xsb.sourceforge.net.
10. XPDL 2.1 Complete Specification, Oct. 2008, http://www.wfmc.org/xpdl.html.
11. ATHENA, D.A3.2 "Updated version of the Ontology Authoring and Management System with semantic search functions", ATHENA IP, deliverable, 2005.
12. W3C: OWL-S, Semantic markup for web services. 22 november 2004, http://www.w3.org/Submission/OWL-S/.
13. Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D.: Web Service Modeling Ontology. Applied Ontology, 1(1): 77-106, IOS Press, 2005.
14. Beeri, C., Eyal, A., Kamenkovich, S., and Milo, T. 2008. Querying business processes with BP-QL. *Information Systems.* 33, 6 (Sep. 2008), 477-507.
15. Ahmed Awad: BPMN-Q: A Language to Query Business Processes. EMISA 2007.
16. Ruopeng Lu and ShaziaWasim Sadiq. Managing Process Variants as an Information Resource. In Lecture Notes in Computer Science, vol.4102, pages 426–431. Springer, 2006
17. Hepp, M., Leymann, F., Domingue, J., Wahler, A., Fensel., D.: Semantic business process management: A vision towards using semantic web services for business process managemen. In: ICEBE 2005, pp. 535–540. IEEE Computer Society, Los Alamitos (2005).
18. Dimitrov, M., Simov, A., Stein, S., Konstantinov, M.: A BPMO based Semantic Business Process Modelling Environment. In Proc. of the Workshop on Semantic Business Process and Product Lifecycle Management at the ESWC, volume 251 of CEUR-WS, 2007.
19. Di Francescomarino C., Ghidini C., Rospocher M., Serafini L., Tonella P., Semantically-aided business process modeling, 8th International Semantic Web Conference (ISWC 2009), Washington DC, USA, 2009.
20. Markovic, I. Advanced Querying and Reasoning on Business Process Models. In Proc. of the International Conference on Business Information Systems, Innsbruck, Austria, 2008.