Ontology-based Querying of Composite Services

Fabrizio Smith, Michele Missikoff, Maurizio Proietti,

National Research Council, IASI "Antonio Ruberti", Viale Manzoni 30, 00185 Rome (Italy)

{smith, missikoff, proietti}@iasi.cnr.it

Abstract. Enterprises are evolving towards advanced forms of cooperation and networking. This kind of tight cooperation requires the creation of global Business Processes (i.e., cross-enterprise composite services) composed starting from a set of existing local processes (exposed, in turn, as services) found in different enterprises. To this end, in this chapter we present an ontology-based approach for querying business process repositories for the retrieval of process fragments to be reused in the composition of new business processes. The proposed solution is based on a synergic use of a business process modelling framework (BPAL) to represent the workflow logic of business processes, and business ontologies, aimed at capturing the semantics of a business scenario. Both components are grounded in logic programming and therefore it is possible to apply effective reasoning methods to query the knowledge base stemming from the fusion of the two.

Keywords: Networked Enterprise, Business Process, Semantic Annotation, Ontology, Query Language, Composite Service.

1 Introduction

In recent years there has been an acceleration towards new forms of cooperation among enterprises, such as virtual enterprises, networked enterprises, or business ecosystems. A networked enterprise integrates the resources and Business Processes (BPs) of the participating organizations allowing them to pursue shared objectives in a tightly coordinated fashion, operating as a unique (virtual) organization (see Chapter 1 of this volume for a discussion on the evolution of business trends and related research challenges). In this context, the notions of a *Service Oriented Architecture* (SOA), viewed as a design philosophy, and of a *Web Service*, viewed as a technological approach, play a key role. SOA is the natural way of designing a software system to provide services to either end-user applications or other services distributed in a network, via published and discoverable interfaces [1]. The implementation of a SOA by means of Web Services, allows packaged functionalities to be offered as a suite of interoperable *services*, widely usable since their interfaces are defined independently of the underlying technologies (see Chapter 8 of this volume, where the requirements for building cross-organizational service-based applications are discussed).

In a Service Oriented Architecture an *orchestration* is described as a Business Process schema, i.e., a workflow graph that specifies the planned order of operations execution. A BP is hence built from a collection of services, possibly implemented and deployed as web services, each of which performs a well-defined activity within the process. Composite services can be defined by combining existing elementary services, thereby yielding higher-level services or processes. Service-oriented computing offers a means for designing global BPs (i.e., cross-enterprise composite services) by assembling existing local BPs (exposed, in turn, as services) found in different enterprises, to virtually form a single logical system. However, in practice this operation is not an easy one, since the *semantic interoperability problem* arises both at the data level and at the process level. The local BPs are often built by using different tools, according to different business logics, and using different labels and terminology to denote activities and resources. To overcome this incompatibilities, the various participating enterprises need to agree on a common view and vocabulary of the business domain (e.g., represented by a Business Reference Ontology), and provide descriptions of the local BPs according to such a common view. The potentials offered by Reference Modeling for the substantial improvement of both the efficiency and effectiveness in BP design have been widely recognized in literature (see, e.g., Chapters 3 and 5 of this volume).

Much work has been done[1] to semantically enrich BP management systems [2] by means of well-established techniques from the area of the Semantic Web and, in particular, computational ontologies [3]. An enterprise ontology provides unambiguous definitions of the entities occurring in the domain, and eases the interoperability between software applications and the reuse/exchange of knowledge between human actors.

In this frame, we propose a semantic platform to be associated with the different BP management tools adopted in the different enterprises to provide a unified, semantically enriched view of the different local BPs. Our proposal is based on a Business Process Knowledge Base (BPKB) that organizes and stores the conceptual knowledge about the process-related knowledge of the enterprises, aiming at the leveraging of the semantic heterogeneities inherent to the aggregation of independently authored resources. Then, the BPKB can be queried to retrieve individual BPs, or fragments therein. Figure 1 represents a view of the macro-architecture of the proposed framework. The main components of the BPKB are the local BP repositories ($LBPR_i$), the common set of ontologies and vocabularies constituting the Reference Ontology and the semantic annotation (Σ_i) relating the local enterprise resources to the reference ontology. Then, the BP Engineer will be able to operate on the local BPs in a unified fashion through the semantic model provided by the BPKB and, by using a number of reasoning services (notably, the BP semantic query processing), will be able to compose a cross-enterprise, global BP.

In particular, while providing a general view of our approach, in this chapter we focus on the problem of querying repositories of semantically enriched BPs. This is

[1] See, e.g., the SUPER (http://www.ip-super.org/), COIN (http://www.coin-ip.eu/) and PLUG-IT (http://plug-it.org/) European projects.

achieved through the synergic use of a logic-based BP modeling language (BPAL [4]) to represent the workflow logic, and an ontological framework (OPAL [5]) aimed at capturing the domain knowledge of a business scenario. Then, the semantic annotation allows us to query process schemas in terms of the conceptualization provided by the reference ontology, easing the retrieval of local BPs (or fragments therein) to be reused in the composition of new BPs.

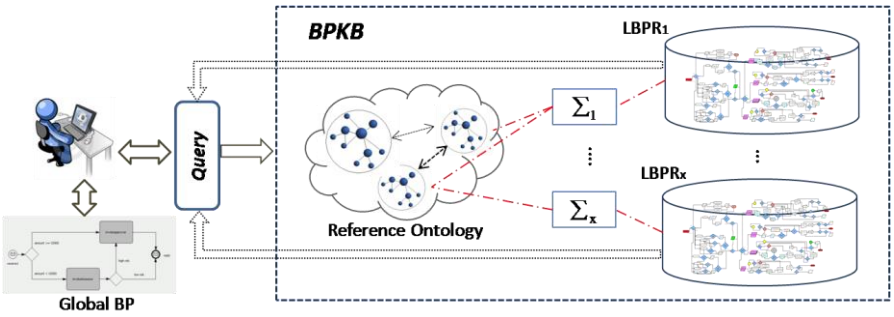

Fig. 1. Approach overview

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 (LP) [6]. At the same time it has been conceived to be used in conjunction with the existing BP management tools as an 'add-on' facility, by supporting BPMN [7], in particular its XPDL [8] linear form, as a modeling notation, and OWL [9], for the definition of the reference ontologies.

The rest of the chapter is organized as follows. The 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 software platform and Section 5 presents related works.

2 Knowledge Representation Framework

In this section we introduce the knowledge representation framework which is at the basis of the querying approach that will be proposed in Section 3. Three different perspectives will be taken into account to represent and reason about BPs: (1) the *structural specification*, which directly descends from the workflow graph associated with each BP; (2) the *behavioral semantics*, i.e., a formal description of the execution semantics of a BP, which enables the analysis of the possible enactments of a BP; (3) the *domain knowledge*, i.e., a conceptualization intended to capture the semantics of the business scenario, used to describe each individual entity participating in a BP.

In Section 2.1 the workflow graph will be formally represented within the BPAL language by defining the notion of a Business Process Schema (BPS) and its *meta-model*. In Section 2.2 we will present the behavioral semantics of a BPS, which is

defined in terms of its *execution traces.* Finally, in Section 2.3 we will present our method for defining a Business Reference Ontology and the semantic annotation within the OPAL framework. The language QuBPAL defined in Section 3 will be used to make complex queries that involve structural, behavioral, and domain-related properties.

2.1 Introducing BPAL

The Business Process Abstract modeling Language (BPAL) [4] is a logic-based language conceived to provide a declarative modeling method capable of fully capturing 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 the BPMN 2.0 specification [7]. For illustration, consider the BP depicted in Figure 2, where an orchestration specifying the handling of a purchase order in an eProcurement scenario is represented using the BPMN notation.

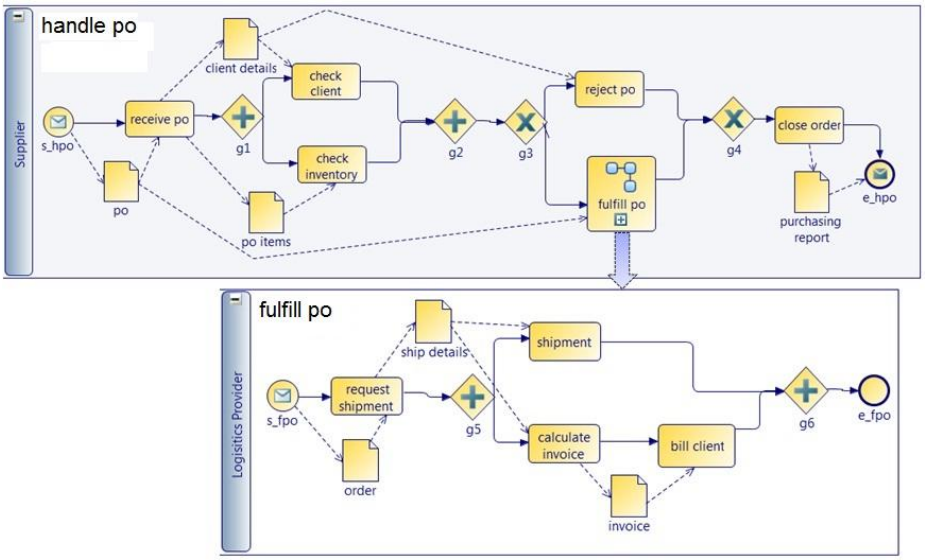

Fig. 2. A Business Process for handling purchase orders

Upon receiving the purchase order from a customer, a supplier initiates two tasks concurrently, to verify the information provided by the customer and to check product availability in the inventory. If the purchase order is accepted, then it is fulfilled. The activity *fulfill_po* is a *compound* activity (modeled as a BPMN sub-process), representing the invocation of the corresponding process, where the shipment and the invoicing are executed by a logistics provider.

In Figure 3 the core elements of the BPAL meta-model are shown in a UML class diagram for sake of readability, while the formalization, including the axiomatisation of its semantics has been presented in [4] and [10] in an extended form.

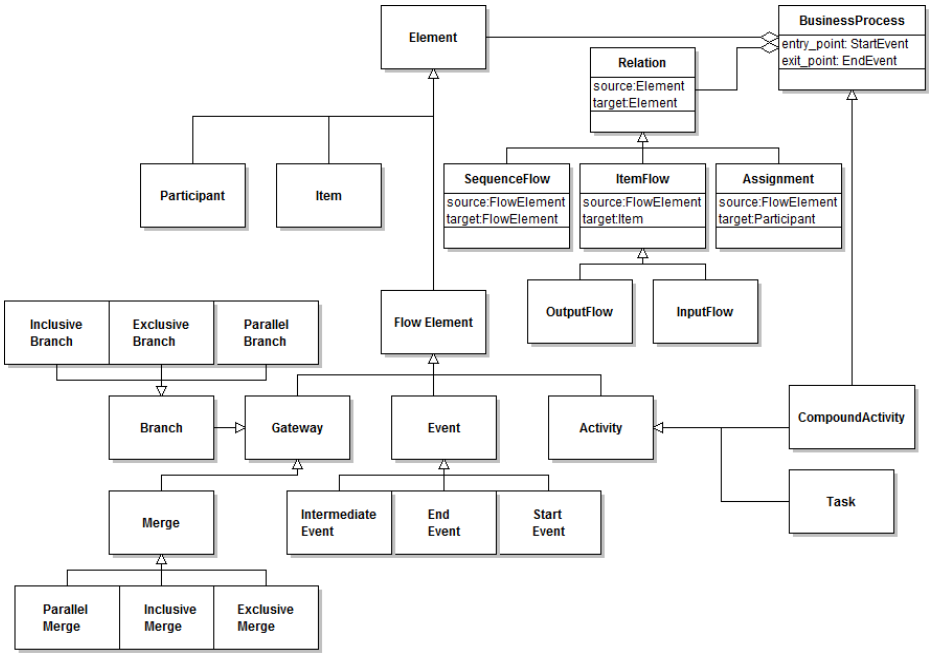

Fig. 3. BPAL core meta-model

A *business process* consists of a set of *elements* and *relations* between elements appearing in the workflow graph, and it is associated with a unique *start event* and a unique *end event* that represent the entry point and the exit point, respectively, of the process. An *activity* is the key element of the business process, representing a unit of work performed within the process. A *task* represents an atomic activity (e.g., *bill_client*), i.e., no further decomposable, while a *compound activity* represents the invocation of composite (possibly remote) process, and it is associated with a workflow graph that provides the definition of its internal structure (e.g., *fulfill_po*). A BPS can thus be viewed as a hierarchy of activities (e.g., the composite activity *fulfill_po* occurs in the process *handle_po*). The sequencing of flow elements is specified by the *sequence flow* relation and, for branching flows, BPAL provides predicates representing *parallel* (AND), *exclusive* (XOR), and *inclusive* (OR) *branching/merging* of the control flow (*gateways*). An *item* represents a physical or information object (e.g., *invoice*) that is created and used during the execution of the process. An item holds the values that are produced during the process enactment and, hence, it is regarded as a variable. An *item flow* specifies that a flow element uses as *input* or produces as *output* a particular item. An item related to an item flow originating in a start event constitutes the input of the process (e.g., *handle_po* is triggered by receiving an order), while an item related to an item flow ending in an end event constitutes

the output of the process (e.g., *handle_po* ends by sending back a final report). Finally, a *participant* is a generic notion representing a role within a company (e.g., *employee*), a department, or a business partner role (e.g., *manufacturer*) which is *assigned* to the execution of a collection of activities. It is worth noting that the *semantics* of the notions of *item* and *participant* are left intentionally underspecified, since an unambiguous definition of their meaning has to be provided in terms of a reference ontology through the semantic annotation, as shown in the next section.

Formally 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 $c_1,\ldots,c_n$ are constants denoting BPS elements (e.g., business activities, events, and gateways) and p is a BPAL predicate. In Table 1 we list some of the BPAL predicates, while in Table 2 we report the BPAL translation of the *fulfill_po* BPS depicted in Figure 2.

Table 1. Excerpt of the BPAL language

bp(*p*,*s*,*e*)	*p* is a process, with entry-point *s* and exit-point *e*
task(*a*)	*a* is a task, i.e., an atomic activity no further decomposable
comp_act(*a*,*s*,*e*)	*a* is a compound activity with entry-point *s* and exit-point *e*
seq(e_1,e_2,*p*)	a sequence flow relation is defined between e_1 and e_2 in *p*
par_branch (*g*)	the execution of *g* enables all the successor flow elements
par_merge(*g*)	*g* waits for the completion of all the predecessor flow elements
exc_branch(*g*)	the execution of *g* enables one of the successor flow elements
exc_merge(*g*)	*g* waits for the completion of one of the predecessor flow elements
inc_branch(*g*)	the execution of *g* enables at least one of the successor flow elements
inc_merge(*g*)	*g* waits for the completion of the predecessor flow elements that will be eventually executed
item(*i*)	*i* is an item
input(*a*,*i*,*p*)	the activity *a* uses as input the item *i* in the process *p*
output(*a*,*i*,*p*)	the activity *a* uses as output the item *i* in the process *p*
participant(*part*)	*part* is a participant
assigned(*a*,*part*,*p*)	the activity *a* is assigned to the participant *part* in the process *p*

Table 2. BPAL representation of the fulfill_po process

comp_act(*fulfill_po*,*s_fpo*,*e_fpo*)	*item*(*order*)
task(*request_shipment*)	*item*(*ship_details*)
task(*calculate_invoice*)	*item*(*invoice*)
task(*bill_client*)	*output*(*s_fpo*,*order*,*fulfill_po*)
task(*shipment*)	*input*(*request_shipment*,*order*,*fulfill_po*)
par_branch(*g5*)	*output*(*request_shipment*,*ship_details*,*fulfill_po*)
par_merge(*g6*)	*input*(*calculate_invoice*,*ship_details*,*fulfill_po*)
seq(*s_fpo*,*request_shipment*,*fulfill_po*)	*input*(*shipment*,*ship_details*,*fulfill_po*)
seq(*request_shipment*,*g5*,*fulfill_po*)	*output*(*calculate_invoice*,*invoice*,*fulfill_po*)
seq(*g5*,*shipment*,*fulfill_po*)	*input*(*bill_client*,*invoice*,*fulfill_po*)
seq(*g5*,*calculate_invoice*,*fulfill_po*)	*participant*(*logistics_provider*)
seq(*calculate_invoice*,*bill_client*,*fulfill_po*)	*assigned*(*request_shipment*,*logistics_provider*,*fulfill_po*)
seq(*shipment*,*g6*,*fulfill_po*)	*assigned*(*calculate_invoice*,*logistics_provider*,*fulfill_po*)
seq(*bill_clientg*,*g6*,*fulfill_po*)	*assigned*(*bill_client*,*logistics_provider*,*fulfill_po*)
seq(*g6*,*e_fpo*,*fulfill_po*)	*assigned*(*shipment*,*logistics_provider*,*fulfill_po*)

BPAL Meta-Model. On top of the BPS modelling layer, we explicitly introduce a BP meta-modelling layer, formalized by the *meta-model theory* ***M*** which defines a set of structural properties of a BPS that at this level is regarded as a labeled graph. First of

all, ***M*** defines how the constructs provided by the BPAL language can be used to build a *well-formed* BPS. Two categories of properties should be verified by a well-formed BPS[2]:

- *local* properties related to the elementary components of the workflow graph. For instance, every activity must have at most one incoming and at most one outgoing sequence flow, i.e.,

  $Y = Z \leftarrow activity(X) \wedge seq(X,Y,P) \wedge seq(X,Z,P)$

  $Y = Z \leftarrow activity(X) \wedge seq(Y,X,P) \wedge seq(Z,X,P)$

- *global* properties related to the overall structure of the process. For instance, in this work we assume that processes are *structured*, i.e., each branch point is matched with a merge point of the same type, and such branch-merge pairs are also properly nested. Such a property is formalized by a set of axioms including the ones listed below (which deal with binary gateways), where *wf_sub_proc*(*p*,*s*,*e*) holds if the sub process of *p* starting with *s* and ending with *e* is a structured block:

  $wf\_sub\_proc(P,X,X) \leftarrow task(X) \wedge occurs(X,P)$

  $wf\_sub\_proc(P,X,X) \leftarrow comp\_act(X,S,E) \wedge occurs(X,P) \wedge wf\_sub\_proc(X,S,E)$

  $wf\_sub\_proc(P,X,Y) \leftarrow wf\_sub\_proc(P,X,Z) \wedge seq(Z,W,P) \wedge wf\_sub\_proc(P,Z,Y)$

  $wf\_sub\_proc(P,X,YM) \leftarrow branch(X) \wedge merge(Y) \wedge same\_type(X,Y) \wedge seq(X,L,P)$
  $\wedge seq(X,R,P) \wedge seq(S,Y,P) \wedge seq(Z,Y,P) \wedge wf\_sub\_proc(P,L,Z)$
  $\wedge wf\_sub\_proc(P,R,S)$

Finally, ***M*** defines other properties related to the notions of paths and reachability between flow elements in the graph structure underlying a BPS. A non-exhaustive list of the predicates defined in ***M*** is given in Table 3.

Table 3. Excerpt of the BPAL meta-model

wf_proc(*p*)	the business process *p* is *well-formed*
wf_sub_proc(*p*,*s*,*e*)	the sub-process of *p* starting with *s* and ending with *e* is well-formed
occurs(*el*,*p*)	*el* occurs in (i.e., belongs to the set of flow elements of) the process *p*
occurs(*el*,*p*,*s*,*e*)	*el* occurs in the sub-process of *p* starting with *s* and ending with *e*
reachable(el_1,el_2,*p*)	there exists a path from el_1 to el_2 in *p*

2.2 Behavioral Semantics of Business Process Schemas

The behavioral 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 the *trace theory* ***TR***. A *trace* models an execution (or instance, or enactment) of a BPS as a sequence $<s_1, s_2, \ldots, s_n>$ of occurrences of activities (or events) called *steps*. The axioms constituting the theory ***TR*** 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

[2] Hereafter when axioms are presented in the form of clauses (i.e., rules), we follow an LP-like notation, with all variables intended as universally quantified and denoted by capital letters.

trace w.r.t. a given BPS, but also to generate the set of correct traces of a BPS. In particular ***TR*** defines the predicates:

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

The notion of trace provides a natural means for the verification of properties regarding the possible executions of a BPS, such as *dependency constraints* (often referred to as *compliance rules*). Dependency constraints state that the execution of an activity is dependent on the execution of another activity, i.e., two activities have to occur together (or in mutual exclusion) in the process (possibly, in a given order). We can define the semantics of dependency constraints as formulas universally quantified over the set of the correct traces of a BPS. As an example we report the semantics of the *response dependency*, represented by the predicate *resp*(*a*,*b*,*p*,*s*,*e*).

$$resp(a,b,p,s,e) \equiv_{def} \forall\, t_1,t_2,s_1,a_1,e_1\ (step(s_1,s) \wedge step(a_1,a) \wedge step(e_1,e) \wedge sub\_trace(t_1,p,s_1,a_1) \wedge sub\_trace(t_2,p,a_1,e_1) \rightarrow \exists\, b_1\ (step(b_1,b) \wedge member(b_1,t_2)))$$

where:

- *step*(a_1,*a*) states that a_1 is an occurrence of the flow element *a* in a possible execution of the process;
- the arguments *s* and *e* limit the scope of the constraint, considered in the sub-process (possibly the whole process) of *p* starting with *s* and ending with *e;*
- *member*(*s*,*t*) holds if *s* is a step in *t*.

This definition states that for every correct trace *t* of the BPS *p*, if a step a_1 of the activity *a* occurs in *t*, then a step b_1 of the activity *b* occurs in *t* after a_1. In a structured BPS, like the ones considered in this chapter, such constraint could be verified by an exhaustive exploration of the set of correct traces. However, this approach would be inefficient, especially when used for answering complex queries of the kind described in Section 3. Thus, we follow a more efficient approach which is based on defining such constraints by means of logic rules that infer the absence of a counterexample. For instance, in the response case, this amounts to prove the absence of a correct trace that leads from a step of activity *a* to a step of *e* and contains no step of *b* in between. This approach is indeed more efficient because, in order to construct a counterexample, we can avoid to actually construct all possible interleavings of the traces generated by the execution of parallel sub-processes and, in fact, we only need to perform a suitable traversal of the workflow graph. In [11] we have followed this approach, based on the encoding of suitable traversals of the workflow graph by means of logic rules, for defining the constraint templates discussed in [12]. The set of these rules constitutes the *dependency constraints theory* ***D***. In Table 4 we list some of the predicates defined in ***D***.

Table 4. Dependency Constraints

prec(*a*,*b*,*p*,*s*,*e*)	*if b is executed then a has been previously executed* in the sub-process of *p* starting with *s* and ending with *e*
resp(*a*,*b*,*p*,*s*,*e*)	*if a is executed then b will be executed* in the sub-process of *p* starting with *s* and ending with *e*
mutex(*a*,*b*,*p*,*s*,*e*)	*a and b are never both executed* in the sub-process of *p* starting with *s* and ending with *e*
coex(*a*,*b*,*p*,*s*,*e*)	*neither a nor b are executed, or they are both executed* in the sub-process of *p* starting with *s* and ending with *e*

2.3 Semantic Enrichment of Process Schemas

In order to provide an alignment of the terminology and conceptualization used in different BPs, it is required to agree on a common view of the business domain, and describe, through a *semantic annotation*, the local BPs according to such agreed common view, represented by a *Business Reference Ontology* (BRO). For the design of a BRO we consider as a reference framework the OPAL methodology [5], while for its technical implementation we commit to OWL/RDF [9], a de facto standard for ontology and meta-data sharing. Hereafter we present OWL expressions 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 *owl* and *rdfs* for the OWL/RDF vocabulary.

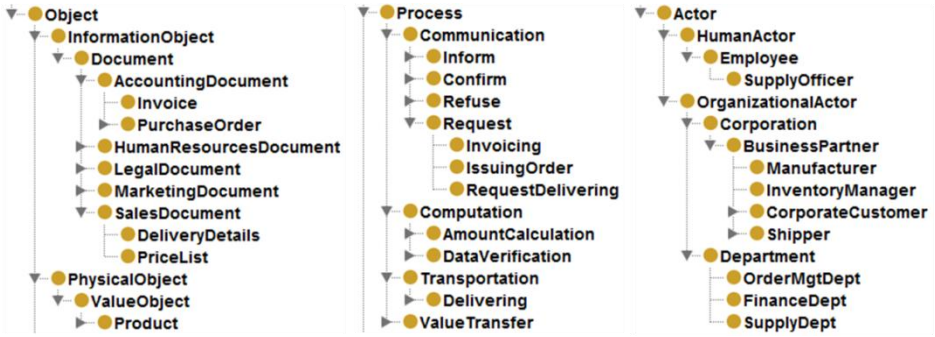

Fig. 4. Exemplary business reference ontology excerpt

OPAL Business Reference Ontology. For the definition of the BRO we consider as the reference framework OPAL (Object Process Actor Language), proposed in [5] as an ontology representation methodology based on UML and OWL, aimed at building business-oriented domain ontologies. OPAL organizes concepts in a number of conceptual categories to support the domain expert in the conceptualization process, identifying active entities (*actors*), passive entities (*objects*), and transformations (*processes*). Therefore, the top level concepts are: *i) opal*:*Process*, representing any business activity or operation aimed at satisfying a business goal and operating on a set of business objects; *ii) opal*:*Actor*, representing active elements of a business domain, able to activate, perform, or monitor a process; *iii) opal*:*Object*, representing

entities on which a business process operates. The development of an OPAL ontology is guided by a use-case driven, iterative and incremental process, derived from the large experience drawn in the software engineering area, with particular reference to the UP software development framework. OPAL has been tested and validated in several national and international projects and applications, showing its effectiveness and high acceptance among business experts.

The adoption of OPAL as a "default" component of the proposed knowledge representation framework is motivated by several reasons. First of all, OPAL is strongly focused on describing the environment in which processes are carried out from the organizations perspective, thus allowing to contextualize BPs within the enterprise assets (e.g., people, departments, resources). Then, such a description is given in terms of a limited number of high level categories (actor, object, process), which constitute a suitable conceptual counterpart for the fundamental modeling constructs identified in BPAL (i.e., participant, item, activity). Finally, a direct formalization into OWL is given. By the way, the commitment to a particular conceptual model is not a restrictive assumption. Other resources developed in the context of Enterprise Modeling, can be adopted as well, given that a suitable representation in terms of a formal language is provided.

Figure 4 shows an excerpt of an exemplary BRO related to the *handle_po* BPS, where three concept hierarchies, having root in *opal*:*Process*, *opal*:*Actor* and *opal*:*Object* respectively, are depicted.

Semantic Annotation. A *Semantic Annotation* $\boldsymbol{\Sigma}$ defines a correspondence between elements of the BPS and concepts of the BRO, in order to describe the meaning of the former in terms of a suitable conceptualization of the domain of interest provided by the latter. To establish a general semantic association between the linked entities inherent in their meaning, we define the relation $\sigma \subseteq BpsEl \times Concept$, where *BpsEl* is an element of a BPS, and *Concept* is either

- a named concept defined in the BRO, e.g. *Shipper*;
- a concept defined by a class expression, e.g. $Shipper \sqcap InventoryManager$.

We do not impose that every BPS element is annotated, nor that every concept is involved in the annotation of some BPS element. On the other hand, different BPS elements could be annotated with respect to the same concept, to provide an alignment of the heterogeneous terminologies and conceptualizations used in different BP schemas, e.g., both the items *order* and *po* actually refer to the same notion, suitably defined in BRO terms as *PurchaseOrder*.

Even though the conceptualization introduced in a BPS differs on scope and purpose from the one provided by a reference domain ontology, some criteria may be introduced to put them in relation. For instance, since the vocabulary introduced in a BPS is intended to be a specialization of the one introduced in the reference ontology, σ is preserved by the subsumption relation, i.e., $\sigma(El,A) \leftarrow \sigma(El,C) \wedge t(C,rdfs{:}subClassOf,A)$. Then, in order to relate the different kinds of BPAL elements to the very general concepts introduced by the top-level categories of OPAL, the annotation is further constrained as follows: *i)* an *activity* has to be annotated with a subclass of *opal*:*Process*, i.e., $t(C,rdfs{:}subClassOf,opal{:}Process) \leftarrow \sigma(A,C) \wedge activity(A)$;

ii) an *item* has to be annotated with a sub-class of *opal*:*Object*, i.e., $t(C,rdfs{:}subClassOf,opal{:}Object) \leftarrow \sigma(I,C) \wedge item(I)$; *iii)* a *participant* has to be annotated with a sub-class of *opal*:*Actor*, i.e., $t(C,rdfs{:}subClassOf,opal{:}Actor) \leftarrow \sigma(P,C) \wedge participant(P)$. Axioms as the ones presented above are very general and domain independent, and are intended as a starting point for further extensions where more specific constraints are formulated to accommodate the requirements of the particular addressed domain.

A semantically enriched business process is hence a BPS defined according to BPAL, whose elements are linked to concepts defined in a reference ontology through a semantic annotation. In order to ease the exchange of meta-data and their reuse, we encode such semantic structure as an RDF model, as exemplified in the above listings, which refer to the ontology in Figure 4. In the example we report an RDF description related to the *reject_po* task, which is annotated with the complex concept *bro*:*Refuse* $\sqcap$ $\exists$*opal*:*content.bro*:*PurchaseOrder* $\sqcap$ $\exists$*opal*:*receiver.bro*:*Customer*, representing the action of notifying to a customer the rejection of an issued order.

```
<rdf:Description rdf:about="reject_po">
 <rdf:type rdf:resource="bpal:Task"/>
 <bpal:input rdf:resource="client_details"/>
 <bpal:assigned rdf:resource="supplier"/>
 <bpal:occurs rdf:resource="handle_po"/>
 <bpal:sigma>
   <owl:Class>
     <owl:intersectionOf rdf:parseType="Collection">
       <owl:Class rdf:about="bro:Refuse"/>
       <owl:Restriction>
         <owl:onProperty>
           <owl:ObjectProperty rdf:ID="opal:content"/>
         </owl:onProperty>
         <owl:someValuesFrom rdf:resource="bro:PurchaseOrder"/>
       </owl:Restriction>
       <owl:Restriction>
         <owl:onProperty>
           <owl:ObjectProperty rdf:ID="opal:receiver"/>
         </owl:onProperty>
         <owl:someValuesFrom rdf:resource="bro:Customer"/>
       </owl:Restriction>
     </owl:intersectionOf>
   </owl:Class>
 </bpal:sigma>
 <bpal:model_ref> http://acme/ACME.xpdl#_123</bpal:model_ref>
</rdf:Description>
```

In this frame, other meta-data definitions can be easily handled, such as references to WSDL operations describing concrete service implementations or to data types defined in XML files. For instance, the above description also include a reference to the BP fragment (*model_ref*) in the original process schema (e.g., an XPDL file) to keep the link between the annotated BPS fragment and its annotation information, in order to allow other systems to process this piece of information.

3 QuBPAL Query Platform

In this section we describe our querying approach, based on the knowledge representation framework described in the previous section. In this framework we are able to define a Business Process Knowledge Base (BPKB) as a collection of logical theories that formalize a repository of semantically enriched business process schemas. The interpretation of these theories as logic programs presented in Section 3.1 provides a powerful inference support, at the basis of the query language we will introduce in Section 3.2 and 3.3.

3.1 Business Process Knowledge Base

In Section 2 we introduced the two main components of the Business Process Knowledge Base, namely *i)* a repository of BPs represented according to BPAL, *ii)* a business reference ontology defined according to OPAL that, together with the semantic annotation, provides a representation of the domain knowledge associated with the BPs. In order to achieve a uniform and formal representation, suited for reasoning on the above structures, we define a BPAL BPS repository ***BPR*** as a First Order Logic theory of the form:

$$\boldsymbol{BPR} = \boldsymbol{B} \cup \boldsymbol{M} \cup \boldsymbol{TR} \cup \boldsymbol{D}$$

where: *i)* ***B*** is a set of BP schemas defined in BPAL, *ii)* ***M*** and ***TR*** are the BPAL core theories formalizing the meta-model and the trace semantics, respectively, *iii)* ***D*** is the dependency constraint theory, introduced with the purpose of efficiently verifying properties regarding the possible executions of a BPS.

Table 5. Inference examples on a BPAL BPS repository

BPR $\vdash$	***BPR*** $\vdash$ *not*
wf_proc(*handle_po*)	*wf_sub_proc*(*handle_po*,*g1*,*g6*)
wf_sub_proc(*handle_po*,*g1*,*g4*)	*occurs*(*reject_po*,*handle_po*)
occurs(*bill_client*,*fulfill_po*)	*reachable*(*reject_po*,*shipment*,*handle_po*)
occurs(*bill_client*,*handle_po*)	*prec*(*bill_client*,*close_order*,*handle_po*,*s_hpo*,*e_hpo*)
occurs(*bill_client*,*handle_po*,*g3*,*g4*)	*resp*(*check_client*,*bill_client*,*handle_po*,*s_hpo*,*e_hpo*)
reachable(*receive_po*,*shipment*,*handle_po*)	*mutex*(*check_client*,*shipment*,*handle_po*,*s_hpo*,*e_hpo*)
prec(*check_client*,*bill_client*,*handle_po*)	*coex*(*check_client*,*bill_client*,*handle_po*,*s_hpo*,*e_hpo*)
resp(*bill_client*,*close_order*,*handle_po*)	
mutex(*reject_po*,*shipment*,*handle_po*)	
coex(*shipment*,*bill_client*,*handle_po*)	

A relevant property of the theory ***BPR*** is that it has a straightforward interpretation as a logic program [3], providing an operational semantics which enables logical inference taking advantage of the tools developed in the area of logic programming (LP). In this frame, all the properties defined in the aforementioned theories can be used for querying the theory ***BPR***. In particular, the predicates defined by the meta-model theory ***M*** and by the BP schemas ***B*** allow us to query the schema level of a BP, verifying properties regarding the elements occurring in it (e.g., *activities, items, gateways*) and their relationships (e.g., *sequence flows*), while ***TR*** and ***D*** allow us to ex-

press queries about the behavior of a BP at execution time, i.e., verify properties regarding the execution semantics of a BPS. Table 5 presents some examples of inferences regarding the *handle_po* BPS, where $\boldsymbol{BPR} \vdash L$ means that L can be inferred by $\boldsymbol{BPR}$[3].

For the representation of the business reference ontology we adopt a fragment of OWL, falling within the OWL 2 RL [9] profile. OWL 2 RL is an OWL subset designed for practical implementations using rule-based techniques. The semantics of OWL 2 RL is defined through a partial axiomatisation of the OWL 2 RDF-Based Semantics in the form of first-order implications (OWL 2 RL/RDF rules), and constitutes an upward-compatible extension of RDF and RDFS. In the EKB, the semantic resources encoded in OWL/RDF are represented by means of the ternary predicate $t(s,p,o)$, and reasoning is supported by including a set of FOL rules encoding the OWL 2 RL/RDF rule-set.

Finally, a Business Process Knowledge Base is formalized by a logic program ***BPKB***, defined by putting together the theories introduced so far:

$$\boldsymbol{BPKB} = \boldsymbol{BPR} \cup \boldsymbol{BRO} \cup \boldsymbol{\Sigma}$$

where: *i)* ***BPR*** is a BPAL BP repository; *ii)* ***BRO*** is an OPAL Business Reference Ontology, encoded as a set of assertions of the form $t(s,p,o)$ and including the OWL 2 RL/RDF rule-set; *iii)* $\boldsymbol{\Sigma}$ is a semantic annotation, including a set of assertions of the form $\sigma(BpsEl,Concept)$.

Hence the logic program ***BPKB*** can be used for evaluating conjunctive queries, formulated as clauses of the form:

$$q(\vec{X}) \leftarrow p_1(\vec{X}_1) \wedge \ldots \wedge p_m(\vec{X}_m) \wedge not\ p_{m+1}(\vec{X}_{m+1}) \wedge \ldots \wedge not\ p_n(\vec{X}_n)$$

where $p_1,\ldots,p_n$ are predicates defined in ***BPKB***, $q(\vec{X})$ is the query to be evaluated by the inference engine, $\vec{X}_1,\ldots,\vec{X}_n$ are vectors of variables such that every X occurring in $\vec{X}$ occurs also in some $\vec{X}_i$.

3.2 The QuBPAL Query Language

Having set the theoretical framework, we can now introduce QuBPAL, an expressive language for querying a BPKB. It does not require the user to understand the technicalities of the underlying logic programming platform, since QuBPAL queries are automatically translated to logic programs and evaluated by using standards LP engines (in particular, we refer to the XSB logic programming and deductive database system[4]). More specifically, QuBPAL queries which do not involve predicates defined in ***TR***, i.e., queries that do not explicitly manipulate traces, are translated to Datalog programs with stratified negation [14]. For this class of programs, proof procedures based on tabled resolution, such as the one implemented in the XSB system, guarantee a polynomial sound and complete top-down evaluation.

---

[3] Formally, L is true in the *Perfect Model* [13] of the stratified program ***BPR***, i.e., $Perf(\boldsymbol{BPR}) \models L$.

[4] The XSB Logic Programming System. Version 3.1, Aug. 2007, http://xsb.sourceforge.net

In the queries we use question mark to denote variables (e.g., *?x*), and we use the notation *?x::c* to indicate the semantic typing of a variable, i.e., $\sigma(X,c)$. A (well-formed) BPS is denoted by <*bpId*>, where *bpId* is a business process identifier. A (well-formed) sub-process is identified by <*bpId,start,end*>, where *start* and *end* are the flow elements of the BPS *bpId* that start and end the sub-process, respectively. Syntactically a query is an expression of the form:

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

The **SELECT** statement defines the output of the query evaluation. A boolean query, which evaluation returns either *true* or *false*, is specified by the symbol **<>**, and contains no variables. Otherwise, as a target list, it can be specified:

- a BPS, denoted by <*?bpId*>;
- a well-formed sub-process, denoted by <*?bpId,?start,?end*>;
- a sequence (possibly empty) *?x** of variables occurring in the WHERE statement.

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

- a sequence of BPS or sub-process identifiers (<*bpId*>|<*bpId,start,end*>)$^{+}$.

In the **WHERE** statement it can specified an expression that restricts the set of data returned by the query. Here, complex properties combining structural, behavioral and domain knowledge can be formulated. The *comparison_predicate* is a sentence built from:

- the set of the predicates defined in the EKB;
- 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 appearing in a query are:

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

3.3 Compiling QuBPAL Queries into LP Queries

The BPAL Platform (see Section 4) provides a compiler that translates QuBPAL queries into LP queries and also performs suitable query optimizations.

The translation of QuBPAL into LP clauses is similar to the translation of SQL into Datalog [14], especially for the treatment of nested and disjunctive queries. In Figure 5 we describe in a pictorial way the translation, considering 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 into multiple clauses.

Besides translating QuBPAL queries into LP queries, the compiler also verifies some syntactic correctness properties. Among these, the compiler checks that the QuBPAL query is *range-restricted* [14], 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 clause also appears in a positive literal in the premise of the clause and, therefore, only ground answers will be returned. Then, the compiler re-orders literals in the premises of the target clause 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 (i.e., only *ground* negative queries are evaluated) by using the top-down, left-to-right strategy usually implemented by LP engines. 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.

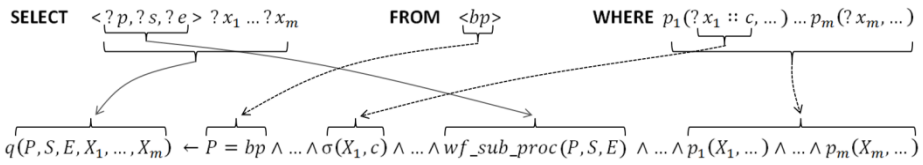

Fig. 5. Translation of QuBPAL into LP queries

3.4 Query Examples

In this section we present some examples of queries over a BPKB. We provide a natural language description of the query, and the corresponding formulation according to QuBPAL.

Ex. 1. The following query searches for processes where a *purchase order* is processed and provides services for the *invoicing* and the *delivering* of goods. Considering our running example, both *handle_po* and *fulfill_po* are returned by the query evaluation.

SELECT <?p>
WHERE occurs(?a1,?p) AND input(?a1,?po::bro:PurchaseOrder,?p) AND occurs(?a2::bro:Invoicing,?p) AND occurs(?a3::bro:Delivering,?p)

Ex. 2. The following query is a refinement of the previous one. It searches for sub-processes that *i)* start with an activity that processes a *purchase order*, and *ii)* both a *delivering* and an *invoicing* are eventually executed (response dependency). Considering our running example, the sub-process starting with *request_shipment* and ending with *g6* is returned.

SELECT <?p,?s,?e>
WHERE input(?s,?po::bro:PurchaseOrder,?p) AND resp(?s,?a1::bro:Invoicing,?p,?s,?e) AND resp(?s,?a2::bro:Delivering,?p,?s,?e)

Ex. 3. The following query searches for every sub-process that is executed as an alternative to one where an *invoicing* and a *delivering* coexist in every possible execution. This example shows the use of negation and nested queries, delimited by curly brackets. Considering our running example, the sub-process of *handle_po* constituted by the activity *reject_po* only is returned.

```
SELECT <?p,?s,?e>
WHERE exc_branch(?b) AND seq(?b,?s,?p) AND seq(?e,?m,?p) AND exc_merge(?m)
AND seq(?b,?s1,?p) AND seq(?e1,?m,?p) AND NOT ?s1 = ?s AND
{SELECT <?p,?s1,?e1> WHERE coex(?x::bro:Invoicing,y::bro:Delivering,?p,?s1,?e1)}
```

Ex. 4. The following query retrieves every *item* representing a *business document* that is processed on a path from the receiving of a *purchase order* to an *invoicing*. Considering our running example, all the items are returned, except for *purchasing_report.*

```
SELECT <?p> ?d
WHERE input(?a1,?po::bro:PurchaseOrder,?p) AND reachable(?a1,?a2,?p) AND
output(?a2,d::bro:Document,?p) AND reachable(?a2,?a3::bro:Invoicing,?p)
```

4 Implementation

This section describes the BPAL Platform, a tool implementing the proposed framework. The BPAL Platform provides the BPKB Editor to assist the user through a graphical interface in the definition of a BPKB, where semantically enriched BPs are represented, and the BPAL Reasoner, based on a LP engine (XSB), able to operate on the BPKB. A functional view of the application is depicted in Figure 6.

4.1 BPAL Reasoner

The BPAL Reasoner has been implemented as a Java application, interfaced with the XSB logic programming engine. The main components of the application are shown in Figure 6, and briefly described below.

- *BPMN2BPAL*. This module offers an interface to import BPAL process schemas into the BPKB from BPMN process models. The input BPMN processes can be acquired from both XPDL files (supported, e.g., by TIBCO[5] and Enhydra Shark[6]) and *.bpmn* files (supported, e.g., by Intalio Process Modeler[7]).
- *RDF2LP*. Reference ontologies and semantic annotations can be imported into the BPKB from OWL/RDF files by means of the *RDF2LP* module. As discussed in Section 2.3, semantic resources are represented in the BPKB through the predicate *t*(*s*,*p*,*o*), encoding a generic RDF statement. This kind of encoding allows for dealing

[5] http://developer.tibco.com/business_studio
[6] http://www.together.at/prod/workflow/tws
[7] http://www.intalio.com/process-designer

indifferently with RDF, RDFS and OWL (restricted to the RL profile). The parsing of OWL/RDF files is based on the Jena2 toolkit[8].

- *BPKB Manager*. This module provides functionalities to populate and update a BPKB, handling the interactions with the LP engine. During the set-up phase, when the BPKB is built, it is responsible for feeding the XSB engine with the logic programs encoding the BPKB, i.e.: *i)* the BPAL BP schemas imported through the *ImportBPS* interface; *ii)* the semantic resources imported through the *ImportRDF* interface; *iii)* the BPAL core theories (*meta-model*, *trace* and *dependency constraints* theories) discussed in Section 2; *iv)* the OWL 2 RL/RDF rule-set to support reasoning over the semantic resources, as discussed in Section 3.1.

- *XSB Prolog*. The application is connected with the XSB system through the Interprolog library[9], a Java/Prolog interface. XSB extends 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. In our setting, XSB has profound advantages over Prolog systems based on SLDNF-resolution: *i)* it allow us to evaluate programs with negation according to the perfect model semantics, *ii)* tabling ensures the termination of query evaluation over a BPKB, since the sizes of subgoals and answers produced during an evaluation are always finite (*bounded term-size* property), *iii)* for queries falling within the stratified Datalog fragment of LP, tabling can achieve the optimal data complexity (i.e., polynomial time) for query evaluation, guaranteeing at the same time a correct and complete evaluation.

- *Query Manager*. Having populated the BPKB, inference is essentially performed by posing queries to the XSB engine. The QueryManager exposes functionalities to translate QuBPAL queries into LP queries, evaluate them by means of the underlining XSB engine, and collect the results. The latter can be exported both in a textual form and as new XPDL files, for their further reuse in an external BPMS.

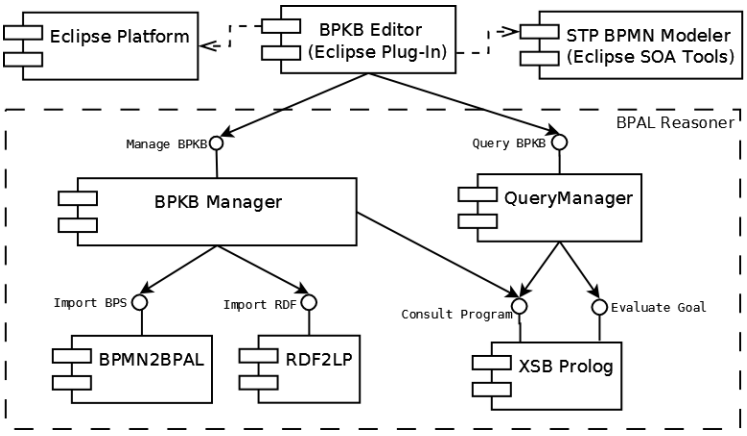

Fig. 6. Functional view of the BPAL Platform

[8] http://jena.sourceforge.net/

[9] http://www.declarativa.com/interprolog

4.2 EKB Editor

The BPKB Editor is implemented as an Eclipse Plug-In[10] which includes the BPAL Reasoner. It provides functionalities for managing persistent resources and a graphical user interface (GUI), where several widgets allow to define a BPKB and to interact with the BPAL Reasoner to exploit the reasoning facilities. The graphical editor for BPMN processes is based on the STP BPMN Modeler developed within the Eclipse SOA Tools Platform[11] which has been included as part of the plug-in.

A screen-shot of the GUI is depicted in Figure 7. The left panel (Figure 7.a) provides a tree view of the resources available in the workspace, including BP schemas and ontologies. The central panel (Figure 7.b) is the BP editor, provided by the STP BPMN Modeler, comprising an editor and a set of tools to model BP diagrams using the BPMN notation. On the right (Figure 7.c), the ontology browser allows for the visualization of OWL ontologies, published on the Internet or locally stored. The bottom panel (Figure 7.d) is the annotation editor, for the annotation of BP elements with respect to the reference ontology. The resulting semantic annotation can be saved and loaded from RDF files. The top-central panel (Figure 7.e) is the query prompt, that provides the users with a direct access to the BPAL reasoner through the query mechanism. Results can be consulted in the 'Result Panel', (Figure 7.f) and, when process fragments are included in the query, the latter can be exported as a new XPDL file for their further re-use.

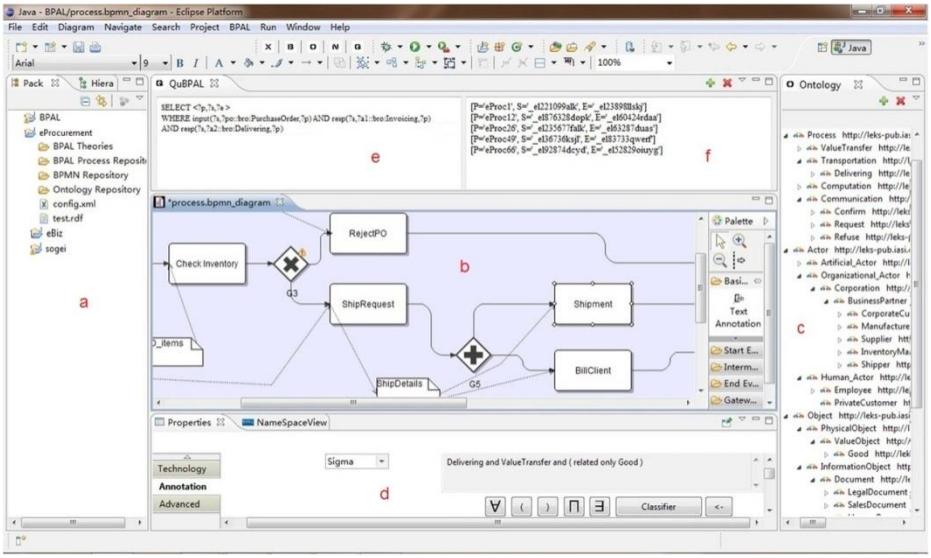

Fig. 7. EKB Editor GUI

[10] http://www.eclipse.org/

[11] http://www.eclipse.org/bpmn/

5 Related Work

A first body of related works proposes the extension to business process management of techniques developed in the context of the semantic web. Within the SUPER project[12] several ontologies to model functional, organizational, informational, and behavioral perspectives have been developed, and in [15] a querying framework based on such ontologies is presented. The approach is limited to the use of semantic annotations, which act as a sort of index, for the retrieval of processes and process fragments (i.e., a subset of activities) related to a given ontology concept. No forms of structural or behavioral reasoning is addressed in the queries. In [16] SPARQL queries, formulated through a visual language, are evaluated against business processes represented trough a BPMN meta-model ontology annotated with respect to domain ontologies. Other approaches based on meta-model ontologies have been discussed, e.g., [17,18]. Unlike the aforementioned works, where the behavioral aspects are hidden or abstracted away, properties defined in terms of the execution semantics can be considered in a QuBPAL query as well (e.g., in the form of dependency constraints). Hence, the BPKB provides a homogeneous framework where one can evaluate complex queries that combine properties related to the ontological description, the workflow structure, and the behavioral semantics of the modeled processes.

Relevant work regarding the semantic enrichment of Web Services has been done within the OWL-S [19] and WSMO [20] initiatives. Both approaches provide service definitions from two perspectives: from a functional perspective a service is described as a black-box in terms of its functionality, pre-conditions and effects, input and output; from a process perspective, the service internal behavior is modeled as a composition (or orchestration) of other services. Such solutions strongly differ from ours on scope and purpose and can be considered complementary to our work. Indeed, although they have been successfully exploited for discovering, composing (mainly through AI planning and automated synthesis techniques) and invoking electronic services over the Web, they do not provide semantics for orchestrations, or constraints between component services and within single services.

Other approaches for BP querying are based on graph matching, through visual languages [21,22] grounded in graph grammars. BP-QL [21] is based on an abstract representation of the BPEL (Business Process Execution Language) standard, and allows one to query the process structure (i.e. control and data flow) of a BPEL process, ignoring the run-time semantics of certain constructs such as conditional or parallel execution. Relevant features provided by this approach are the support for hierarchical workflows and the possibility to control the granularity of the query, considering certain components as black-boxes or exploring recursively their internal structure. Similarly, BPMN-Q [22] is a visual language based on BPMN which supports graph-based query processing. Both approaches allow the user to query the graph representation of a process workflow in an intuitive way, but they need to be combined with external tools to reason about properties of the behavioral semantics. For instance, BPMN-Q supports also some templates of behavioral constraints (e.g.,

[12] http://www.ip-super.org/

precedence, response) which are verified by a model checker, fed by a temporal logic query and a finite state machine obtained by the state space generation of a Petri net equivalent to a given BPMN model. Our framework not only provides a method for querying the structure of the workflow graph and its behavior, but, due to the logic-based representation, it also integrates additional reasoning services avoiding the burden of dealing with heterogeneous formalisms and tools. Indeed, a very relevant advantage of QuBPAL is the possibility of formulating queries involving the knowledge represented in domain models formally encoded by means of ontologies. QuBPAL queries can be posed in terms of the ontology vocabulary, which offers a "global view" of the processes annotated with it, hence *i)* decoupling queries from specific processes, *ii)* overcoming semantic heterogeneities deriving, e.g., from different terminologies, *iii)* allowing queries to be posed at different generality levels by taking advantage of the semantic relations defined in the ontology, such as *subsumption*.

Finally, [23,24] present other approaches based on logic programming for modeling and reasoning on workflows. Anyway, these works mainly focus on the verification and on the enactment of BPs, while they have not been so far applied to the problem of querying.

6 Conclusions

In this chapter we presented a framework for querying repositories of semantically enriched business processes, conceived to ease the retrieval of process fragments to be used in the design of cross-enterprise composite services. The proposed solution is based on the synergic use of BPAL, a logic-based language adopted for modeling the structure and the behavior of business processes represented accordingly to a workflow perspective, and business ontologies, providing a conceptualization of the business scenario. Both components are seamlessly connected thanks to their grounding in first order logic (in particular, logic programming) and therefore it is possible to apply effective reasoning methods to query the knowledge base encompassing the two. A preliminary evaluation of the implemented reasoning engine was conducted to prove the feasibility of the approach [25]. In particular, the rule-based implementation of the OWL reasoner and the effective goal-oriented evaluation mechanism of the LP engine shown good response time and significant scalability.

To make our approach applicable in real-world scenarios, besides a sound formal foundation we strive also for practical usability, by supporting widely used and accepted standards and technologies. We adopt BPMN as a graphical modeling notation, and its XML linear forms to import and manipulate BP models, possibly designed through external BP Management Systems. For what concerns the ontology representation, we have committed to OWL, the current de-facto standard for ontology modeling and meta-data exchange. By doing this, we allow the already existing and widespread technologies, like those based on BPMN, to be maintained, and 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 the formal framework we provide.

We are working to extend the proposed knowledge representation framework in several directions. First of all, we want to increase the expressivity of the approach by supporting a larger number of workflow patterns [2], to ease its adoption in conjunction with commercial tools for BPMS. Then, the query evaluation process can be strongly optimized through more elaborated transformations achieved by exploiting sophisticated program transformation techniques [26]. 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 re-engineering and automatic process composition. 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.

Acknowledgments. This work has been partly funded by the European Commission through the ICT Project BIVEE: Business Innovation and Virtual Enterprise Environment (No. FoF-ICT-2011.7.3-285746). The authors wish to acknowledge the Commission for its support. We also wish to acknowledge our gratitude and appreciation to all BIVEE project partners for their contribution during the development of various ideas and concepts presented in this paper.

References

1. Papazoglou, M.P.: Web Services: Principles and Technology. Pearson. Prentice Hall, 2007.
2. ter Hofstede, A.H.M., van der Aalst, W.M.P., Adams, M., Russell, N.: Modern Business Process Automation: YAWL and its Support Environment. Springer, 2010.
3. 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, 2005.
4. De Nicola, A., Missikoff, M., Proietti, M., Smith, F.: An Open Platform for Business Process Modeling and Verification. Proc. DEXA 2010, LNCS 6261, pp. 66-90. Springer, 2010.
5. De Nicola A, Missikoff M, Navigli R: A Software Engineering Approach to Ontology Building. Information Systems, 34:258-275, 2009.
6. Lloyd, J.W.: Foundations of Logic Programming. Springer-Verlag, Berlin, 1987.
7. OMG: Business Process Model and Notation, http://www.omg.org/spec/BPMN/2.0, 2010.
8. Workflow Management Coalition: XML Process Definition Language, http://www.wfmc.org/xpdl.html, 2008.
9. Hitzler, P. et al.: OWL 2 Web Ontology Language. W3C Recommendation, http://www.w3.org/TR/owl2-primer, 2009.
10. Missikoff, M., Proietti, M., Smith, F.: Linking Ontologies to Business Process Schemas. IASI-CNR Technical Report, R. 10-20, 2010.
11. De Nicola, A., Missikoff, M., Smith, F.: Towards a Method for Business Process and Informal Business Rules Compliance. Journal of Software Maintenance and Evolution: Research and Practice. Published online in Wiley Online Library, doi:10.1002/smr.553, 2011.
12. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. Proc. ICSE'99, pp. 411-420. ACM, 1999.
13. Przymusinski, T. C.: On the Declarative Semantics of Deductive Databases and Logic Programs. In: Jack Minker (Ed.): Foundations of Deductive Databases and Logic Programming, pp. 193-216. Morgan Kaufmann, 1988.
14. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, 1995.

15. Markovic, I. Advanced Querying and Reasoning on Business Process Models. Proc. BIS 2008, LNBIP 7, pp.189-200. Springer, 2008.
16. Di Francescomarino, C., Tonella, P.: Crosscutting Concern Documentation by Visual Query of Business Processes. Business Process Management Workshops 2008, LNBIP 17, pp. 18-31. Springer, 2009.
17. Haller, A. Gaaloul, W., Marmolowski, M.: Towards an XPDL Compliant Process Ontology. Proc. IEEE Congress on Services, pp.83-86. IEEE Computer Society, 2008.
18. Yun Lin. Semantic Annotation for Process Models: Facilitating Process Knowledge Management via Semantic Interoperability. PhD thesis, Norwegian University of Science and Technology, 2008.
19. Burstein, M., et al.: OWL-S, Semantic Markup for Web Services. W3C Member Submission, http://www.w3.org/Submission/OWL-S/, 2004.
20. 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.
21. Beeri, C., Eyal, A., Kamenkovich, S., and Milo, T.: Querying business processes with BP-QL. Information Systems. 33(6):477-507, 2008.
22. Awad, A., Weidlich, M., Weske, M.: Visually specifying compliance rules and explaining their violations for business processes. Journal of Visual Languages and Computing, 22:30-55, 2011.
23. Montali, M., et al: Verification from Declarative Specifications Using Logic Programming. Proc. ICLP 2008, LNCS 5366, pp. 440–454. Springer, 2008.
24. Roman, D. and Kifer, M.: Reasoning about the Behavior of Semantic Web Services with Concurrent Transaction Logic. Proc. VLDB 2007, pp. 627-638. VLDB Endowment, 2007.
25. Missikoff, M., Proietti, M., Smith, F.: Querying Semantically Enriched Business Processes. Proc. DEXA 2011, LNCS 6861, pp. 294-103. Springer, 2011.
26. Pettorossi, A. and Proietti, M.: Transformation of Logic Programs: Foundations and Techniques. Journal of Logic Programming 19:261-320, 1994.