Formalizing Business Processes using Hybrid Programs

Suman Roy[1] and Wlodek Drabent[2]

[1] Infosys Lab, Infosys Ltd., # 44 Electronics City, Hosur Road, Bangalore 560 100
Email:Suman_Roy@infosys.com
[2] Wyższa Szkoła Informatyki i Ekonomii TWP w Olsztynie, Olsztyn, Poland;
Institute of Computer Science, Polish Academy of Sciences, Warszawa, Poland
Email: drabent@ipipan.waw.pl

Abstract. A semantic annotation of business processes with concepts from ontology has become necessity in service provisioning. There have been few work on semantically labeling business processes in terms of ontology that formalizes business process structure, business domains etc. However, dynamic behavior of a process cannot be captured by such means as ontology languages are not suitable for specifying behavioral semantics. In this work, we propose a method for labeling and specifying business processes by using hybrid programs as the knowledge representation formalism. The formalism of hybrid programs integrates normal programs (using the parlance of logic programming) with ontology specified in OWL-DL (semantic web standard).

Keywords: Knowledge representation, Business processes, Semantic web, OWL-DL, OWL with rules, Hybrid rules, Case study.

1 Introduction

A semantic annotation of business processes with concepts from ontology has become necessity in services industry. This work mainly involve two aspects, adding semantics to specify the meaning of the entities of a business process and adding semantics to specify the dynamic behavior of a process. We consider both these issues while we try to formalize business processes. While OWL-DL seems to the perfect choice for semantically annotating process diagrams [8, 9], rule-based formalisms are needed to capture dynamic behavior of processes. OWL-DL belongs to the family of Description Logic (DL) languages which offer well-understood computational and attractive decidable properties, that could be useful in settling inferences about different concepts/classes of processes. Moreover, the languages in OWL family use open world assumption under which the inability to prove a statement A does not imply that its negation $\neg A$ has to be concluded. Such a property is desirable on processes as lot of times available domain knowledge may be incomplete and certain assertions cannot be inferred without more information. However, ontology based languages fall short when it comes to expressing dynamic behavior of the processes. When we want to express properties related to control flow of the process (which is typical in the specification of functional requirements) we need to use rule based frameworks for modeling such behavior. Further we wish to exploit the feature of non-monotonic reasoning (thereby

forcing the closed world assumption on control flow behavior) available with such logics. For these reasons, we adopt the framework of hybrid programs proposed in [5, 6] which integrate OWL-DL ontology with first-order normal logic programs, to label and specify business processes.

Related work There are some ongoing research work on applying semantic web formalisms for annotating business processes using semantic web formalisms. Business processes have been tagged with semantic labels as a part of knowledge base with a view to formalize business process structure, business domains, and a set of criteria describing correct semantic marking in [8]. In another work [9], the authors propose semantic web language OWL to formalize business process diagrams, and automatically verify sets of constraints on them, that deal with knowledge about the domain and process structure. The authors in [10], attempt to analyze requirements for modeling and querying process models and present a pattern-oriented approach for process modeling and information retrieval. These semantic annotation techniques of business processes lead to the possibility of semantic validation, *i.e.,* whether the tasks in business processes are consistent with respect to each other in the underlying framework of semantic annotation. Such issues are investigated in [20], where the authors introduce a formalism for business processes, which combines concepts from work flow with that from AI. A rule-based method for modeling processes and workflows has been proposed in [14], where the authors introduce an extended Event-Condition-Action (ECA) notation for refining business rules, and perform a consistent decomposition of business processes. Cicekli and Cicekli have used event calculus, a kind of logic programming formalism for specification and execution of workflows in [3]. They express control flow graph of a work flow specification as a set of logical formulas. We have been inspired by this work, but we decide to use a combination of logic programs and OWL-DL formulas for extracting knowledge out of business processes.

2 A Motivating Example

In this section we consider an example of a bidding process (see Figure 1) which we shall later use for formulating the knowledge base. The process diagram called Business Process Diagram (BPD), is drawn using notations similar to Business Process Modeling Notation (BPMN). In this process the bidder starts by reading the description of the item followed by contacting the seller for item. Then both the activities *viz.* studying seller's credit information and reading seller's feedback are carried out in parallel. These activities lead to decision on bidding. If it is decided to bid then the bidder buys the item, and the bidder verify bidder's credentials and close the auction. The realization of this bidding can be achieved by few experts, who may wish choose requirements (constraints) on the process itself. These requirements may include different aspects of the process, as follows;

- Contacting seller is always preceded by reading item description.
- In the bidding process there must be roles of a bidder and a seller.
- The seller will close the auction.
- Aborting auction can put an end to the bidding process.

All these constraints are examples of descriptive properties of the annotated process. Naturally, a requirement analyst who is using this process for bidding an item would like

to extract relevant information about these processes. Hence it makes sense to extract a knowledge base out of such processes so that we could use the reasoning capability of the underlying formalism to support semantic requirements of the application in mind.

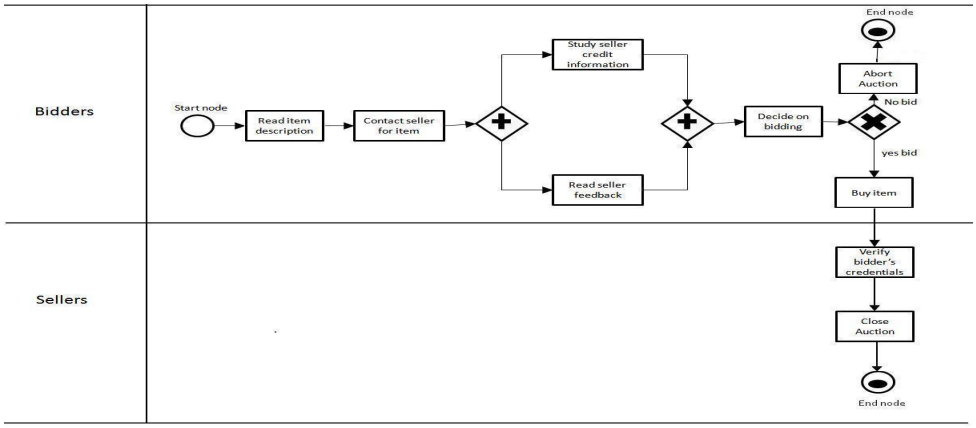

Fig. 1. An example of a bidding process

3 Logical preliminaries

In this section we discuss briefly some basics related to OWL-DL and hybrid rules.

A primer on OWL-DL W3C Web Ontology Working Group has proposed a semantic markup language called OWL (Web Ontology Language) [2]. OWL is developed as a vocabulary extension of RDF (the Resource Description Framework [13]). It shares many common features with Description Logic (DL) [19, 1]. While the syntax of OWL corresponds to that of RDF the semantics of OWL are extensions of the semantics of DL. OWL comes with three different fragments - OWL Lite, OWL-DL and OWL Full, they differ in terms of syntax and expressibility. For our work, we restrict ourselves to OWL-DL. OWL-DL is a syntactic variant of the fragment of DL, *viz.* $\mathcal{SHOIN}(\mathbf{D})$ [12, 18]. $\mathcal{SHOIN}(\mathbf{D})$ supports reasoning with datatypes, such as strings, integers through datatype roles/properties.

Hybrid programs The syntax of hybrid programs [5, 6] is defined using that of component programming language, *e.g.,* languages of normal logic programs [17] and language of OWL-DL. The alphabets of the predicate symbols of underlying logic program and OWL-DL-based logic languages are assumed to be disjoint, although they can have common variables and constants (names). A normal logic program [17] is extended with with ontological constraints to form a hybrid program. A declarative semantics of a hybrid program can be provided by extending the well-founded semantics of normal programs [6].

4 Semantic annotation of business process using ontology

The Business Process Management Initiative (BPMI) has come out with a standard Business Process Modeling Notation (BPMN) for capturing pictorial representation of

business processes. BPMN defines a Business Process Diagram (BPD) which is based on flowchart related ideas, and provides a graphical notation for business process modeling using objects like nodes, edges etc. We adopt a similar business process diagram based on a standard definition of business processes which consist of nodes like, events, activities, gateways and control flow relation linking two nodes. A *node* can be a task (also called an activity), an event, a fork (`AND-split`), a choice (`XOR-split`), a synchronizer (`AND-join`), a merge (`XOR-join`) gateway etc. In a BPD, there are *start events* denoting the beginning of a process, and *end events* denoting the end of a process. We do not take into consideration message passing, timer events etc in our model. Two processes can be located respectively within separate pools (labeled with process names), called *swim-lanes* or *roles*, which represent two participants (*e.g.*, business entities or business roles). Examples of such business process models are processing of purchase orders over the Internet, processing of insurance claims, tour planning and requisition in a company etc. A business process is *well-formed* if it has exactly one start node with no incoming edges and one outgoing edge from it, there is only one incoming edge to a task and exactly one outgoing edge to a task, each fork and choice has exactly one incoming edge and at least two outgoing edges, each synchronizer and merge has at least two incoming edges and exactly one outgoing edge, every node is on a path from the start node to some end node, and there exists at least one task in between two gateways (this is to avoid triviality). Unless otherwise mentioned, from now on without loss of generality, we shall consider only well-formed business processes.

In order to semantically annotate the BPDs and the associated domain, and to support automated reasoning on them we use the formalism proposed in [8, 9] to capture all the relevant information about the processes in the form of logical knowledge bases, which we shall call Business Process Knowledge Base (BPKB). The BPKB is implemented using W3C semantic web standard OWL (Web Ontology Language) [2]. As usual, the knowledge base can be separated into two parts: *TBox* which represents knowledge about the terminology, *i.e.*, the processes and classes, and *ABox* which contains knowledge about the individuals. A BPKB consists of the following components:

The Business Process ontology (BPO) The BPMN ontology, called BPMNO, is a formalization of the structure of a BPD as defined in [8, 9]. This ontology introduces the core elements of the process like, objects (event, activity, gateways) and sequence flows as concepts. It also includes a set of axioms about those concepts. In our case, we consider a subset of taxonomy of the graphical elements of business processes. For example, ProcOnto is a basic concept which is included in Thing. The sub elements of ProcOnto are the following concepts: Agent, Activity, Event, Document, Process etc. These elements have sub-elements which are not shown in the figure. For example, Activities can have sub-Activities, Processes can have sub-processes as sub-elements. We identify some structural patterns with processes: sequential, AND-gateways, XOR-gateways etc. These patterns provide links between activities. They are modeled as object properties/roles on activities. We assume that in business processes an activity is assigned to only an agent, and an agent can perform only one activity at one point of time. This is captured by a object property called, qualified. There is also a role relDocument which connects an activity to a document associated with it in the diagram (if shown in the

diagram). We emphasize that BPO formalizes the structural part of business processes, in particular, it specifies the basic elements and how they are connected.

Business Domain ontology (BDO) BDO captures the domain on which the particular process diagram is conceived. It provides precise semantics to the terms used to annotate the particular business process. The *TBox* axioms would include role hierarchies for agents, business documents, processes etc that are relevant to the process under consideration. Moreover, axioms in *ABox* would relate proper classes/concepts used in the process through object properties like, qualified etc. The BDO can take the form of an existing business domain ontology (*e.g.*, RosettaNet or similar standard business ontology), or a customization of an existing ontology, or a description of the domain for a particular purpose.

Business Process instances ontology (BPIO) The BPD instances contain the description of the corresponding instances of the business process, its domain and ontology. Every concept for instance, Agent, Process, Activity is given a representation as an individual of the concept. The structure of the process (the connection between different elements) is represented as instantiations of the respective roles. Further, some of the instance axioms are generated by instantiating the appropriate *ABox* axioms for the business process and domain ontology.

A BPKB can be modeled using any knowledge representation language like OWL-DL which has a complete decision procedure. Using logical reasoning over BPKB one can implement query answering service on BPD instances. The queries can involve either domain ontology, process ontology or both, for example, list the number of activities, or, find all the activities that can be performed by the agent Bidder for the process shown in Figure 1.

5 Business process specification

The ontology created from business processes cannot model the dynamic behavior (the control flow relation) of process diagrams as ontology languages are not suited to specify behavioral semantics. This kind of control flow related behavior can be better modeled by using formalisms like event calculus, logic programs etc. Specification of such a process behavior involves capturing relevant aspects of its constituent activities, the relationships among activities and their execution requirements.

For specifying process details[3], we associate predicates with important concepts about activities found in processes. We assume that there is an agent (may be, process flow manager) that coordinates the execution of the activities according to the specification of process flow. The initiation and termination of activities are triggered by events. For example the main event is the starting of an activity A denoted as $start(A)$, and the secondary event is $end(A)$ which marks the completion of an activity. We introduce the predicate $duration(A, Y)$ to denote that an activity A takes Y unit of time to get completed. We maintain a history of main events, denoted as $history(H)$ which define

[3] We adopt the following notation: upper case letter denotes variables while lower case letter constants

a prefix of a complete scenario. It can be captured by the following rules.

$$history([\,]).$$
$$history(HH) \leftarrow history(H),\ (\ step(Happens, H),\ [Happens|H] = HH \ ;\ steps(List, H),\ append(List, H, HH)\ ).$$

Predicate $step$ describes extending a history; $step(Happens, H)$ means that a history H can be extended by an item $Happens$. Similarly, $steps(L, H)$ describes extending H by a list of items. We borrow few axioms from event calculus [15] to suit our purpose. These axioms are related to notions of events, properties and periods of time for which the properties would hold. It is assumed that events initiate and/or terminate periods of time in which a property holds. These axioms reflect that with the occurrence of events, new properties hold in the new state of the world, and properties terminate when they are no longer true from the previous state. The main axiom is called persistence axiom. It states that a property P holds under certain conditions at time T.

$$holds\_at(P, T, H) \leftarrow initiates(E, P), happens3(E, T_1, H), T_1 < T, \\ not\ terminated\_between(P, T_1, T, H).$$
$$terminated\_between(P, T_1, T_2, H) \leftarrow terminates(E, P), happens3(E, TT, H), \\ T_1 \leq TT, TT \leq T_2. \qquad \textbf{Ax0}$$

The first predicate $holds\_at(P, T, H)$ denotes that the property P holds at time T in history H. The predicate $initiates(E, P)$ represents the fact that the event E initiates a period of time during which the property P holds. We assume $happens3(E, T, H)$ to denote the fact that event E happens at time T in history H. In the last rule $terminated\_between(P, T_1, T_2, H)$ represents that the property P ceases to hold at some time between T_1 and T_2 in history H due to an event which terminates it. The predicate $terminates(E, P)$ says that event E puts an end to a period during which P was true. Note the border conditions: if event E initiates P at time T_1 then P starts to be true after T_1; P not yet true at T_1, but, if event E terminates P at TT then P does not hold already at TT.

In a history, we use a term $happens(E, T)$ to record the fact that event E happens at time T. These rules query a history.

$$happens3(start(A), T, H) \leftarrow member(happens(start(A), T), H).$$
$$happens3(end(A), T_2, H) \leftarrow happens3(start(A), T_1, H), duration(A, Y), T_2 = T_1 + Y.$$

It is possible to identify a process flow with a few transition patterns such as, *sequential*, *parallel*, *conditional*, *iteration* etc. which are self-explanatory. Each of these patterns can be suitably captured using appropriate rules.

Let us now try to specify **sequential activities**. Suppose the activity A can start unconditionally, whenever activity A_0 finishes (see Figure 2). Note that we are using a prefix of dl to denote the atoms taken from process ontology. This is captured as:

$$step(happens(start(A), T), H) \leftarrow dl(sequential(A_0, A)), \\ happens3(end(A_0), T, H), not\ \ happens3(start(A), T, H).$$

Fig. 2. Sequential Activity

In this rule (and similar ones) $not\ \ happens3(start(A), T, H)$ prevents inserting an event $start(A)$ to the history twice.

For **concurrent activities**, some activities may be executed concurrently in a process flow. In particular, activities after an AND-split are scheduled to be executed in parallel. Figure 3(a) shows an AND-split. Activities $A_1, A_2, \ldots, A_n$ can start only when the activity A completes its execution, and the former activities occur concurrently. We have used predicate $and\_split(A, L)$ to denote that activity A is split into a list L of activities, $A_i, 1 \leq i \leq n$. This can be captured as follows:

$$steps(EventList, H) \leftarrow happens3(end(A), T, H), dl(and\_split(A, L)), L = [A_1|H],$$
$$not\ \ happens3(start(A_1), T, H), start\_list(L, T, EventList).$$

The following rules create the list of start events.

$$start\_list([\,], \_T, []).$$
$$start\_list([A_i|L], T, [happens(start(A_i), T)|List]) \leftarrow start\_list(L, T, List)$$

In an AND-join (see Figure 3(b)) the activity A can start when all the preceding ac-

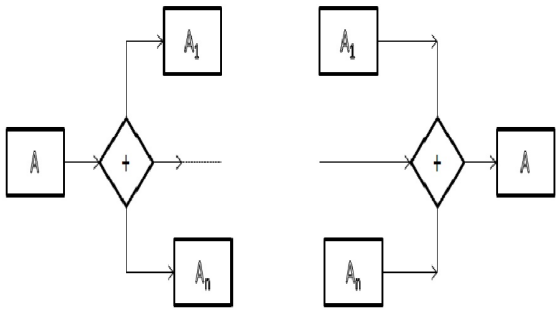

Fig. 3. (a) AND-split and (b) AND-join

tivities $A_1, \ldots, A_n$ finish. These activities may not be completed concurrently, thus we have to find the ending time of all the activities. The activity A will start at the time of the last ending activity among $A_1 \ldots$ and A_n. The predicate $and\_join(L, A)$ indicates that the activities in the list L are merged into the activity A. The predicate $endtime(L, T, H)$ says the activities in L end at T in history H.

$$step(happens(start(A), T), H) \leftarrow dl(and\_join(L, A)), endtime(L, T, H),$$
$$not\ \ happens3(start(A), T, H).$$
$$endtime([A], T, H) \leftarrow happens3(end(A), T, H).$$
$$endtime([A|L], T, H) \leftarrow happens3(end(A), T_A, H), endtime(L, T_L, H), T = \max(T_A, T_L).$$

In case of **conditional activities** some of the activities get enabled depending on certain conditions, otherwise they are not executed. The important point to notice here is that only one of the conditions should hold at the time of decision, so that only one path is taken. In an XOR-split (see Figure 4(a)), when activity A finishes, one of the activities

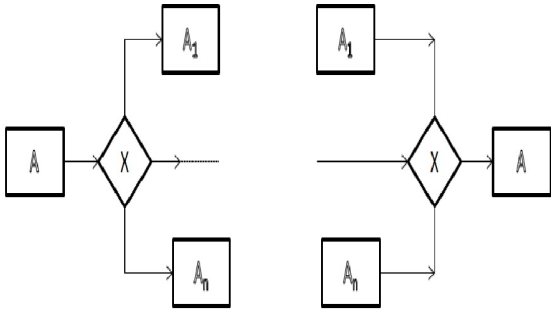

Fig. 4. (a) XOR-split and (b) XOR-join

$A_i, i \in \{1, \ldots, n\}$ can begin its execution depending on whether the condition associated with that particular activity is satisfied. As we assume the exclusiveness of the conditions we do not deal with it in the rules. This can be specified as follows:

$$step(happens(start(A_i), T), H) \leftarrow dl(xor\_split(A, L)), happens3(end(A), T, H),$$
$$member(A_i, L), dl(pair(A_i, Cond)), holds\_at(Cond, T, H),$$
$$not \;\; happens3(start(A_i), T, H)$$

Here, predicate $xor\_split(A, L)$ denotes that the activity A gets split into a set of activities $A_1, \ldots, A_n$ in the list L. The $dl(pair(A_i, Cond))$ has the obvious meaning that activity A_i is associated with the condition $Cond$ in this gateway.

In a gateway consisting of an XOR-join (Figure 4(b)), when one of incoming activities to the join is completed, the outgoing activity can start. The incoming activities need not have to be synchronized. As only one of the path is followed, the completion of one of the incoming activities is sufficient to trigger the beginning of the merged activity. We capture the fact that only the path coming from A_i is active, and leads to the activity A, by the following rule:

$$step(happens(start(A), T), H) \leftarrow dl(xor\_join(L, A)), happens3(end(A_i), T, H),$$
$$member(A_i, L), not \;\; happens3(start(A), T, H).$$

In the above, $xor\_join(L, A)$ indicates that the list L of activities get merged into the activity A. If this rule holds, A_i will be the completed predecessor activity with T as its ending time.

In this framework, **iteration of activities** can be also handled. At times, it is required to repeat a set of activities occurring in a loop. This loop may be executed a certain number of times depending on the exit condition, see Figure 5. The activities between

A_2 and A_n can be arranged as any of the transition types, and at the exit the iteration condition is checked. Iteration block can be implemented in a similar block as XOR gateways. Note that an activity can occur more than once inside a loop, but the time stamps would distinguish each occurrence.

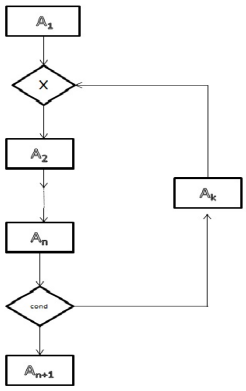

Fig. 5. Iteration of activities

For each activity a_s that starts at the initial time point 0 an axiom $happens3(start(a_s), 0, H)$ holds. There is some initial conditions/fluents marking the beginning of the process, $initiates(start(a_s), initiation)$. This will imply that $holds\_at(initiation, 0)$ will hold using axioms **Ax0**.

Completion of some of the tasks will mark the end of the process, which can be indicated by $initiates(end(A_l), closure)$ and so on, where $closure$ is a fluent. By the flow of events, one may arrive at $happens3(end(A_l), \_T, H)$. Using persistence axioms it is true that $holds\_at(closure, T, H)$. We check the history to be complete by the following rule,
$complete\_history(H) \leftarrow history(H), holds\_at(closure, T, H)$.

6 A Case study: Bidding process

Let us consider a case study of Auction bidding process (see Figure 1) to illustrate our technique for abstracting a hybrid program out of it. First, we build the corresponding ontology in OWL-DL, and as stated before we do the construction in different phases such as creating business process ontology, domain ontology, business process instances etc. BPO (see Listing 1) will consist of the main class Thing and its sub-class ProcOnto. The sub-concepts of ProcOnto are Agent, Event and Activity. The patterns in the process are modeled as roles between activities. Some sample axioms are given in Listing 1. Domain ontology reflects the domain on which the Bidding process is modeled. In particular, it depicts the role hierarchies and roles with proper classes, see some sample axioms in Listing 2. While developing ontology for the process instances all the concepts are provided with suitable instantiations, which facilitates querying later on. The link between the objects of the ontology are also instantiated through using proper

Listing 1 Bidding Process Ontology

ProcOnto $\sqsubseteq$ Thing
Agent $\sqsubseteq$ ProcOnto
Activity $\sqsubseteq$ ProcOnto
Event $\sqsubseteq$ ProcOnto
Domain(qualified) = Agent, Range(qualified) = Activity
Domain(pair) = Activity, Range(pair) = "Condition"
Domain(sequential) = Activity, Range(sequential) = Activity
Domain(and_split) = Activity, Range(and_split) = ListofActivities
Domain(and_join) = ListofActivities, Range(and_split) = Activity

Listing 2 Bidding Process Domain Ontology

Bidder $\sqsubseteq$ Agent
Seller $\sqsubseteq$ Agent
ReadItem $\sqsubseteq$ Activity
ContactSeller $\sqsubseteq$Activity
StudySeller $\sqsubseteq$ Activity
ReadSeller $\sqsubseteq$ Activity
Agent $\equiv$ ReadItem$\sqcup$ContactSeller$\sqcup\cdots\sqcup$CloseAuction
ReadItem $\sqcap$ ContactSeller $\sqsubseteq \bot$
ContactSeller $\sqcap$ StudySeller $\sqsubseteq \bot$

roles, see Listing 3 for some sample instances. Finally the hybrid rules are given in Listing 4. We admit that there is no specific built-in support for expressing lists, sequences, or ordering in OWL. However, there are many aspects in these formalisms that can be used to model many aspects of sequences (sacrificing the preciseness). In [7] two design patterns are discusses for modeling ordering using OWL-DL constructs. We indicate that such modeling techniques can be adopted for expressing lists in our case too.

7 Discussions

We have provided a methodology for designing a general framework to generate a knowledge base out of activity diagrams with roles by using a combination of OWL and rules. There are other works which combine OWL with rules, *e.g.*, Krisnadhi *et al.* [16] and Heymans *et al.* [11]. The tutorial [4] contains some overview/comparisons between such approaches.

References

1. F. Baader and W. Nutt. Basic Description Logics. In *Description Logic Handbook*, pages 43–95, 2003.
2. S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. *OWL Web Ontology Language Reference*. W3C Recommendation, 2003.
3. N. K. Cicekli and I. Cicekli. Formalizing the specification and execution of workflows using the Event Calculus. *Information Sciences*, 176(15):2227–2267, 2006.

Listing 3 Bidding Process instances ontology

```
Bidder(bidder1)
Seller(seller1)
ReadItem(readitem1)
ContactSeller(contactseller1)
VerifyBidder(verifybidder1)
CloseAuction(closeauction1)
pair(buyitem1,"'yesbid"')
pair(abortauction1,"'nobid"')
sequential(readitem1,contactseller1)
sequential(verifybidder1,closeauction1)
and_split(contactseller1,[studyseller1,readseller1])
and_join([studyseller1,readseller1],decideon1)
qualified(bidder1,readitem1)
qualified(bidder1,contactseller1)
qualified(seller1,verifybidder1)
```


4. W. Drabent. Hybrid reasoning with non-monotonic rules. In *Reasoning Web. Semantic Technologies for Software Engineering*, volume 6325 of *Lecture Notes in Computer Science*, pages 28–61. Springer, 2010.
5. W. Drabent, J. Henriksson, and J. Maluszynski. HD-rules: A hybrid system interfacing Prolog with DL-reasoners. In *Proceedings of the ICLP'07 Workshop on Applications of Logic Programming to the Web, Semantic Web and Semantic Web Services (ALPSWS)*, 2007.
6. W. Drabent and J. Maluszynski. Hybrid rules with well-founded semantics. *Knowl. Inf. Syst.*, 25(1):137–168, 2010.
7. N. Drummond, A. L. Rector, R. Stevens, G. Moulton, M. Horridge, H. Wang, and J. Seidenberg. Putting OWL in order: Patterns for sequences in OWL. In *Proceedings of the OWLED'06 Workshop on OWL: Experiences and Directions*, 2006.
8. C. D. Francescomarino, C. Ghidini, M. Rospocher, L. Serafini, and P. Tonella. Reasoning on semantically annotated processes. In *ICSOC, 6th International Conference*, pages 132–146, 2008.
9. C. D. Francescomarino, C. Ghidini, M. Rospocher, L. Serafini, and P. Tonella. Semantically-aided business process modeling. In *8th nternational Semantic Web Conference (ISWC)*, pages 114–129, 2009.
10. G. Groner and S. Staab. Modeling and query patterns for process retrieval in OWL. In *8th International Semantic Web Conference (ISWC)*, pages 243–259, 2009.
11. S. Heymans, J. de Bruijn, L. Predoiu, C. Feier, and D. V. Nieuwenborgh. Guarded hybrid knowledge bases. *TPLP*, 8(3):411–429, 2008.
12. I. Horrocks, P. F. Patel-Schneider, S. Bechhofer, and D. Tsarkov. OWL rules: A proposal and prototype implementation. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):23–40, July 2005.
13. G. Klyne and J. Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation, 2004. available at http://www.w3.org/RDF/.
14. G. Knolmayer, R. Endl, and M. Pfahrer. Modeling processes and workflows by business rules. In *Business Process Management*, volume 1806 of *LNCS*. Springer, 1998.
15. R. A. Kowalski and M. J. Sergot. A logic-based calculus of events. *New Generation Comput.*, 4(1):67–95, 1986.

Listing 4 Hybrid rules for Bidding Process

```
use 'https://sites.google.com/site/tzihan88/FreightBillingProcess.owl
?attredirects=0&d=1' as 'g'

history([]).
history(HH) :- history(H), ( step(Happens, H), [Happens|H] = HH;
                             steps(List,H), append(List,H,HH) ).
holds_at(P,T,H) :- initiates(E,P), happens3(E,T1,H), T1 < T,

terminated_between(P,T1,T2,H) :-
        terminates(E,P), happens3(E,TT,H), T1 <= T, T <= T2.
happens3(start(A),T,H) :- member(happens(start(A),T),H).
happens3(end(A),T2,H) :- happens3(start(A),T1,H), duration(A,Y), T2 is T1+Y.
step(happens(start(A),T),H) :- dl(g#sequential(A0, A)), happens3(end(A0),T,H),
        not happens3(start(A),T,H).
steps(EventList, H) :- happens3(end(A), T, H), L = [A1|H],
        not happens3(start(A1), T, H), start_list(L, T, EventList).
start_list([], _T, []).
start_list([A1|L], T, [happens(start(A1),T)|List]) :- start_list(L,T,List).
step(happens(start(A),T), H) :-
        dl(g#and_join(L,A)), endtime(L,T,H), not happens3(start(A1),T,H).
endtime([A], T, H) :- happens3(end(A), T, H)
endtime([A|L], T, H) :- happens3(end(A), TA, H), endtime(L, TL, H), T = max(TA,TL).
complete_history(H):- history(H), holds_at(closure,T,H).
initiates(start(readitem1),initiation).
initiates(end(closeauction1),closure).
initiates(end(abortauction1),closure).
happens(start(readitem1),0).
duration(readitem1,1).
duration(readseller1,1).
```


16. A. Krisnadhi, F. Maier, and P. Hitzler. OWL and rules. In *Reasoning Web, Semantic Technologies for the Web of Data - 7th International Summer School 2011, Galway, Ireland, August 23-27, 2011, Tutorial Lectures*, pages 382–415, 2011.
17. J. W. Lloyd. *Foundations of Logic Programming*. Springer Verlag, 1987.
18. B. Motik, U. Sattler, and R. Studer. Query answering for OWL-DL with rules. *Journal of Web Semantics*, 3:41–60, 2005.
19. M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.
20. I. Weber, J. Hoffmann, and J. Mendling. Beyond soundness: on the verification of semantic business process models. *Distributed and Parallel Databases*, 27(3):271–343, 2010.