Refining Semantically Annotated Business Process Diagrams

Hector G. Ceballos, Eduardo Fernandez, Jorge Espinosa
ceballos@itesm.mx, yopichoi@gmail.com, arima121@gmail.com

Tecnologico de Monterrey
Av. Eugenio Garza Sada 2501
Monterrey, N.L., Mexico

Abstract. Business Process Diagrams (BPDs) provide a rich graphical expressiveness for modeling not only business processes but also the development of collective human activities. In Multiagent platforms, BPDs has been used for decoupling the modeling of agent behavior from its implementation. Additionally, the annotation of BPD actions and events with conjunctive queries has opened the door for monitoring process development by querying a knowledge base. We study the properties of this annotation scheme for the refinement of activity diagrams through the removal of redundant nodes, the detection of disjoint alternative paths and the merge/split of event nodes.

Keywords: Business Process Diagrams, Semantic Annotation, Description Logics.

1 Introduction

BPMN is a standard notation for modeling business processes widely adopted by industry that provides a rich graphical representation that can be used for common understanding of processes [13]. Furthermore, BPMN has been used for process automation with support of agent technologies given its direct correspondence with some MAS architectures [11, 9]. In these approaches the specification of agent behavior is decoupled from the implementation of agent actions (denoted as task nodes). BPMN BPDs has been also used for modeling collective human activities, observable through the effects of human actions [2].

On the other hand, the use of conjunctive queries has been previously proposed for describing the meaning of events and actions [6, 3]. In [6], actions are described by their direct effects and these annotations are used for determining the status of a process instance. In [3], each possible outcome of a random variable representing an event or an action is annotated with a conjunctive query in order to enable process monitoring.

Muehlen and Indulska performed a representational analysis of modeling languages for business processes and business rules, finding that the most complete combination is BPMN and SWRL [12]. Rules in the Semantic Web Rule Language (SWRL) are constituted by conjunctive queries [7].

This paper is organized as follows. We introduce basic notions of BPMN Business Process Diagrams and Conjunctive Queries in section 2. Then, in section 3 we introduce semantic descriptors on a BPD normal form, describe the properties provided by the use of conjunctive queries and show how these properties can be used for refining a BPD by detecting disjoint alternative paths, deleting redundant nodes, and merging/splitting nodes. We conclude with closing remarks and future work in section 4.

2 Background

Next we introduce the formal notions used for introducing conjunctive queries as semantic descriptors on top of BPMN BPDs.

2.1 Business Process Diagrams

Business Process Modeling and Notation (BPMN) provides a formal representation of Business Process Diagrams (BPDs), which basically describe a process in terms of *events* and *tasks* connected through *control flows* that indicate valid sequences in the process development. *Gateways* are special nodes connected through control flows that indicate whether the process develops in parallel (AND), alternatively (XOR) or optionally (OR). The beginning of the process is denoted by an *initial event* node and its conclusion by a set of *end event* nodes.

Figure 1 shows a subset of the graphical notation of the BPMN 2.0 specification [13]. These graphical elements are used in approaches for software engineering [14, 4] and for modeling human activities [2] through BPDs.

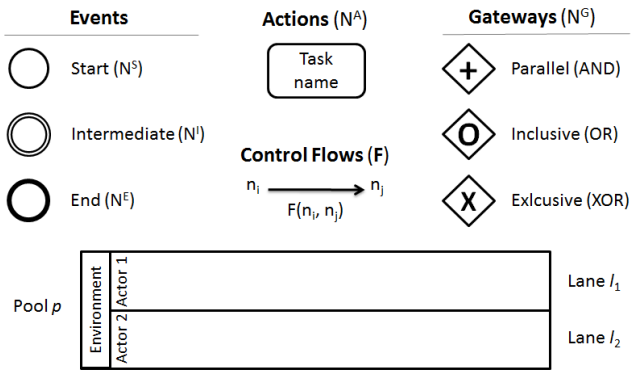

Fig. 1. BPMN graphical notation.

Industry has adopted XML file formats for authoring and exchanging BPDs such as XML Process Definition Language (XPDL)[15] and BPMN 2.0[13]. In

them, nodes representing tasks, events, gateways and control flows are represented through unique identifiers along with properties and relationships with other nodes.

2.2 DL Conjunctive Queries

We use the notation and properties of conjunctive queries given by Description Logics (DL) [1], which states that the interpretation of a query is not only given by statements specified in the query, but by constraints and definitions specified in the domain model ($\mathcal{T}$) as well. This model, known as *ontology*, is basically constituted by a set of *concepts* (used to group/classify objects) and *relations* (used to specify entity's attributes or used to relate entities among them). Ontology languages such as OWL[1] allows expressing concepts in terms of other concepts (definitions), and specifying constraints between concepts (e.g. disjointness or subsumption) and properties (e.g. reflexivity or transitivity).

A DL *conjunctive query* (CQ) has the form:

$$Q = (\mathtt{s}_1, ...\mathtt{s}_n).\{T_1, ..., T_m\}$$

where $\mathtt{s}_i$ is the set of distinguished variables, denoted $Dis(Q)$, that define the resulting binding sets (the information retrieved), and T_i is a finite set of either *concept clauses* (`s rdf:type C`) or *relation clauses* (`s r s'`), where $\mathtt{s}, \mathtt{s}' \in (N_V \cup N_C)$, `C` is a concept/class and `r` is a relation/property both defined in $\mathcal{T}$, N_V is a finite set of variables denoted $Var(Q)$, and N_C is a finite set of constants.

In SPARQL, the query language for RDF[5], a conjunctive query is coded as a SELECT query without filters. Distinguished variables ($Dis(Q)$) of a conjunctive query Q are listed in the section SELECT, whereas atoms T_i are specified in the section WHERE as statements or subject-predicate-object triplets.

Query containment between conjunctive queries can be decided automatically observing the constraints and definitions given in $\mathcal{T}$, as proposed by [8]. A query Q_1 is *contained in* a query Q_2 with respect to $\mathcal{T}$ (written $\mathcal{T} \models Q_1 \sqsubseteq Q_2$), if and only if, for every model I of $\mathcal{T}$, $Q_1(I) \sqsubseteq Q_2(I)$. In other words, $Q_1 \sqsubseteq Q_2$ and $\mathcal{K} \models Q_1$ implies that $\mathcal{K} \models Q_2$ also, i.e. the answer to Q_1 will be included in the answer to Q_2. By definition, every well-formed query Q is subsumed by $\top$, i.e. $\mathcal{T} \models Q \sqsubseteq \top$.

Queries Q_1 and Q_2 are *equivalent* if $Q_1 \sqsubseteq Q_2$ and $Q_2 \sqsubseteq Q_1$, denoted $Q_1 \equiv Q_2$. Q_1 and Q_2 are considered *disjoint* if $\mathcal{K}$ becomes inconsistent whenever $\mathcal{K} \models Q_1 \sqcap Q_2$ for any replacement of common variables in both queries, also denoted by $Q_1 \sqcap Q_2 \sqsubseteq \bot$. By definition, if $Q_1 \sqsubseteq Q_2$ then Q_1 and Q_2 are not disjoint.

Merge of two queries Q_1 and Q_2 is given by its intersection and it is denoted by $Q_1 \sqcap Q_2$, whereas the opposite operation is called *split*.

[1] OWL Web Ontology Language. `https://www.w3.org/TR/owl-features/`

3 Semantic Description of Business Process Diagrams

Now we introduce semantic descriptors for describing the meaning of lanes, actions and events, provide some basic properties for annotated nodes and show how these can be used for refining a BPD.

3.1 Semantic Descriptors

In this paper we use the formalization of BPDs and the normal form proposed in [2]. In this approach, the BPD has the purpose of illustrating alternative sequences of human actions performed by activity participants, mediated by intermediate events that the subject or other participants can observe. XOR gateways are used for representing disjoint alternatives. Activity development has a triggering condition (initial event) and a set of successful or failure outcomes (end events).

A Business Process Diagram $\mathbf{W}$ is represented by a set of pools ($\mathbf{P}$), lanes ($\mathbf{L}$), nodes ($\mathbf{N}$) and control flows ($\mathbf{F}$). Nodes ($\mathbf{N}$) allowed in the diagram are: start events (N^S), intermediate events (N^I), end events (N^E), human tasks (N^A) and gateways (N^G). A gateway $g \in N^G$ can be of type Parallel-AND (A), Optional-OR (O), or Exclusive-XOR (X), denoted respectively $type(g, \{A, O, X\})$. Unconditional sequence flows are denoted as $F(n_i, n_j) \in \mathbf{F}$, $n_i \rightarrow n_j$ for short, where $n_i, n_j \in N$. All nodes $n \in \mathbf{N}$ are situated in a lane $l \in \mathbf{L}$, denoted $in(n, l)$.

$$\mathbf{W} = \{\mathbf{P}, \mathbf{L}, \mathbf{N}, \mathbf{F}\}$$

$$\mathbf{N} = N^S \cup N^I \cup N^E \cup N^A \cup N^G$$

Definition 1 (Semantic Descriptor). *A semantic descriptor $Ann(n, Q)$ uses the conjunctive query Q for representing the meaning of a lane, an observable event, or human task $n \in (\mathbf{L} \cup N^S \cup N^I \cup N^E \cup N^A)$ in a BPD $\mathbf{W}$.*

A *lane descriptor* describes the kind of individual that play a role in the activity. It has the form $Ann(l, Q_l)$, where $l \in \mathbf{L}$ and Q_l might represent an absolute or relative role. An *absolute role* annotation is given by a $Q_l = (?role).\{?role\ a\ RoleClass\}$ where `RoleClass` indicates the type of individual, denoted by $?role$, associated to l. A relative *role annotation* is given by a $Q_l = (?role).\{?role\ rel\ ?role2\}$ where the role associated to the lane ($?role$) is defined in terms of its relationship (`rel`) with a participant represented by another lane ($?role2$).

An *event descriptor* has the form $Ann(z, Q_z)$, where $z \in (N^S \cup N^I \cup N^E)$ and Q_z is a conjunctive query describing a condition (constraints between individuals) that denotes the occurrence of the event.

A *task descriptor* has the form $Ann(x, Q_x)$, where $x \in N^A$,

$$Q_x = (?act).\{?act\ a\ TaskClass\ .\ ?act\ doneBy\ ?role\ .$$

$$?act\ prop_i\ ?value_i\},$$

`TaskClass` indicates the type of task performed or initiated by $?role$, and the task description is expressed by statements such as $?act\ prop_i\ ?value_i$. According to Activity Theory[10], task description might include the participation of other agents playing a role in the activity, the use of artifacts and the location where the task occurs. The distinguished variable $?act$ denotes the task execution (the action in act).

Definition 2 (Annotated BPD). *An annotated BPD $\mathcal{W}_\mathcal{D}$ is a BPD $\mathcal{W}$ where each node $n \in (\mathbf{L} \cup N^S \cup N^I \cup N^E \cup N^A)$ is annotated with exactly one semantic descriptor $Ann(n, Q) \in \mathcal{D}$.*

3.2 Properties of Annotated BPDs

The use of semantic descriptors allows to verify automatically if the meaning of a BPD node is equivalent to, disjoint with, or subsumed by another BPD node.

Definition 3 (Node Disjointness). *The BPD node n_1 is disjoint with the BPD node n_2, denoted $n_1 \bot n_2$, if and only if $Q_1 \sqcap Q_2 \sqsubseteq \bot$ with respect to (w.r.t.) $\mathcal{T}$, given $Ann(n_1, Q_1)$ and $Ann(n_2, Q_2)$.*

Definition 4 (Node Subsumption). *The BPD node n_1 is subsumed by the BPD node n_2, denoted $n_1 \sqsubseteq n_2$, if $Q_1 \sqsubseteq Q_2$ and $Q_2 \not\sqsubseteq Q_1$ w.r.t. $\mathcal{T}$, given $Ann(n_1, Q_1)$ and $Ann(n_2, Q_2)$.*

Definition 5 (Node Equivalence). *The BPD node n_1 is equivalent to the BPD node n_2, denoted $n_1 \equiv n_2$, if and only if $Q_1 \equiv Q_2$ w.r.t. $\mathcal{T}$, given $Ann(n_1, Q_1)$ and $Ann(n_2, Q_2)$.*

These definitions can be used for detecting redundant nodes, i.e. consecutive event nodes annotated with equivalent descriptions.

Theorem 1 (Node Redundancy). *Given two consecutive event nodes $z_1, z_2 \in (N^S \cup N^I \cup N^E)$, $z_1 \rightarrow z_2$, such that $z_2 \sqsubseteq z_1$, z_2 will hold whenever z_1 holds, making z_2 redundant in the diagram.*

Proof. During process monitoring, the occurrence of the event z_2 will be evaluated right after z_1. z_2 being subsumed by z_1 means that z_2 will hold whenever z_1 holds, hence checking the occurrence of z_2 becomes unnecessary once z_1 has occurred.

In the example of Figure 2, the event $z_{7.2}$ is *redundant* as long as $z_{7.2} \sqsubseteq z_{5.1}$, making unnecessary the observation of $z_{7.2}$. The following corollary complements Theorem 1.

Corollary 1. *Given two consecutive event nodes $z_1, z_2 \in (N^S \cup N^I \cup N^E)$, $z_1 \rightarrow z_2$, such that $z_2 \equiv z_1$, either z_1 or z_2 becomes redundant.*

Proof. Given that z_1 will hold whenever z_2 does and vice versa, either of the two can be chosen as redundant node.

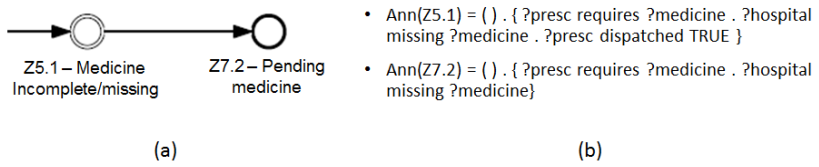

Fig. 2. Eliminating redundant nodes.

Two other properties apply to event nodes preceded by the same gateway. Two BPD events z_i and z_j are called *siblings*, denoted as $z_i||z_j$, if both of them are directly preceded by a gateway $g \in N^G$, i.e. $z_i \leftarrow g \rightarrow z_j$.

Determining that all sibling events following to a gateway g are disjoint indicates that g must be XOR-Exclusive. Figure 3 shows two alternative scenarios as result of a medical consultation, denoted by two sibling events, $z_{4.2}$ and $z_{4.3}$, with their corresponding descriptors. $Ann(z_{4.2})$ and $Ann(z_{4.3})$ are disjoint as long as the cardinality of the boolean role `followUp` is constrained to a single value in the concept `Prescription`, stated in $\mathcal{T}$ as:

$$\texttt{Prescription} \sqsubseteq \leq 1\ \texttt{followUp}$$

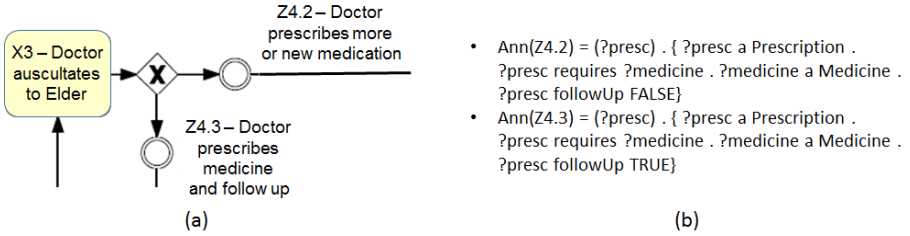

Fig. 3. Detecting XOR gateways.

Theorem 2 (Disjoint sibling events). *If every pair of sibling events $z_i||z_j$ immediately preceded by a splitting gateway $g \in N^G$ are disjoint to each other ($z_i \perp z_j$), then g must be an event-driven exclusive (XOR) gateway, i.e. $type(g, X)$.*

Proof. Once g is reached, process development will continue through g if and only if some event z_i occur after g. Being disjoint every pair of sibling events, only one of them can occur at a given time, hence the first that occur will continue process' monitoring, which is the definition of an event-driven exclusive (XOR)

gateway. In contrast, g cannot be parallel (AND) as long as once that z_i holds the condition represented by any of its sibling will not hold anymore. For the same reason g cannot be optional (OR), i.e. more than one sibling cannot hold.

On the contrary, overlaps between sibling events following to an exclusive gateway violates process development rules.

Theorem 3 (XOR-Equiv overlap). *A pair of sibling events $z_i||z_j$ immediately preceded by a splitting Exclusive-XOR gateway $g \in N^G$ such that $z_i \equiv z_j$, will provoke a violation of process monitoring after g.*

Proof. Once g is reached, process development must continue through one and only one path given by the sibling event holding next. $z_i \equiv z_j$ means that both events will hold, violating the rule imposed by the XOR gateway.

Theorem 4. *A pair of sibling events $z_i||z_j$ immediately preceded by a splitting Exclusive-XOR gateway $g \in N^G$ such that $z_i \sqsubseteq z_j$, can potentially provoke a violation of process monitoring after g.*

Proof. Observing z_j means that z_i will hold as well, enabling process development through both branches which constitutes a violation of the splitting XOR gateway rule. Nevertheless, observing z_i does not assure that z_j will hold as well, in which case the violation would not occur.

Semantic description of sibling events following XOR gateways have similar or complementary descriptions as long as they represent alternative scenarios. In these cases, annotations of sibling events have a common sub-expression that can be captured before the splitting gateway.

Definition 6 (Common Precondition). *The conjunctive query Q_g is considered a common precondition for all sibling events z_i preceded by g if and only if $Q_i \equiv Q_g \sqcap Q'_i$ for every $Ann(z_i, Q_i) \in \mathcal{D}$.*

Given that the normal form does not allow to annotate gateways, the common precondition must be represented through the introduction of an intermediate event preceding the gateway.

Theorem 5. *The observation of a precondition Q_g, included in a set of disjoint sibling events z_i following to the gateway g, can be represented through the introduction of an event node z_g such that: $n_p \rightarrow g$ is replaced by $n_p \rightarrow z_g \rightarrow g$, and $\mathcal{D} = \mathcal{D} \cup Ann(z_g, Q_g) \setminus Ann(z_i, Q_i) \cup Ann(z_i, Q'_i)$.*

Proof. Process monitoring requires observing z_g before reaching g, and then evaluating every z_i. Assuming world state does not change between the observation of z_g and the observation of z_i, the observation of Q'_i simultaneously to Q_g is by definition equivalent to observing Q_i.

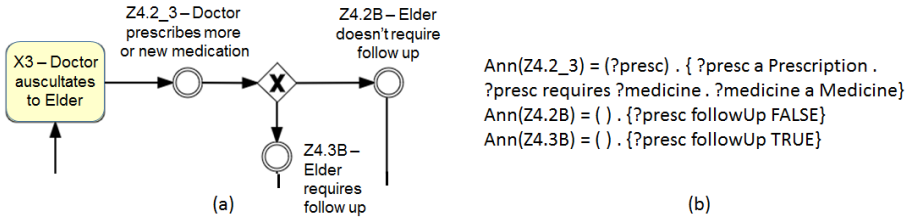

Fig. 4. Adding a common precondition.

In both alternatives shown in Figure 3 the doctor prescribes medication, but in $z_{4.3}$ the patient requires follow up whereas in $z_{4.2}$ he does not. Figure 4 shows an equivalent representation where the event $z_{4.2\_3}$ represents the common condition, and original events are replaced by the necessity of a follow up appointment ($z_{4.3B}$) or not ($z_{4.2B}$).

Properties provided by semantic descriptors also permits to validate if the workflow can be monitored properly. For instance, Theorem 2 provides a rule for checking the correspondence between XOR gateways and disjoint events.

Descriptors of nodes in the structure shown in Figure 4 must be also checked to prevent monitoring issues. For instance, if doctor's prescription (evaluated as common precondition) indicates both more medication and a follow-up appointment, and one of the exclusive alternative paths check for more medication whereas the other asks for the follow-up appointment, then monitoring could continue through both paths, but the XOR gateway forces to continue for only one (chosen arbitrarily).

Theorem 6. *A common precondition z_g subsuming at least two alternative events z_i immediately following a XOR gateway g, i.e. $z_i \sqsubseteq z_g$, will provoke a violation of process monitoring after g.*

Proof. Assuming that all sibling events z_i are evaluated immediately after the precondition z_g, once that z_g holds any z_i will hold as well. Process monitoring will follow through the branch of the first z_i evaluated, despite another branch could be followed (the one indicated by the other $z_i \sqsubseteq z_g$).

In the same circumstances but having a Parallel-AND gateway, the precondition becomes redundant.

Theorem 7. *A common precondition z_g is redundant in the diagram if it subsumes all alternative events z_i immediately following a Parallel-AND gateway g, i.e. $z_g \rightarrow g \rightarrow z_i$ and $z_i \sqsubseteq z_g$.*

Proof. Given that process monitoring continues through all branches of a splitting Parallel-AND gateway, the evaluation of z_g before every z_i is unnecessary as long as z_i will necessarily hold once that z_g holds.

3.3 Refining and Validating Annotated BPDs

Theorems 1 and 7, and corollary 1 are used for simplifying the BPD by absorbing redundant nodes. Theorem 5 is used for introducing common preconditions. Theorem 2 is used for signaling disjoint alternatives. Finally, theorems 3, 4 and 6 are used for warning about errors on process monitoring.

The following procedure can be used for validating and automatically refining the annotated BPD $\mathcal{W}_\mathcal{D}$:

1. Identify consecutive events $z_1 \to z_2$ such that $z_1, z_2 \in (N^I \cup N^E)$,
   (a) if $z_1 \sqsubseteq z_2$ then *absorb* z_2.
      i. if $z_2 \in N^E$ then $N^I = N^I \setminus z_1$ and $N^E = N^E \cup z_1$.
   (b) if $z_1 \equiv z_2$ then *absorb* z_1.
2. For each splitting gateway g immediately followed by sibling events $z_i || z_j$,
   (a) If all pairs (z_i, z_j) are disjoint $(z_i \bot z_j)$, then set $type(g, XOR)$.
   (b) If exists a common precondition Q_g such that $Q_i \equiv Q_g \sqcap Q'_i$ for each descriptor $Ann(z_i, Q_i)$ of all sibling events z_i, insert an event $z_g \to g$, $\mathcal{D} = \mathcal{D} \cup Ann(z_g, Q_g) \setminus Ann(z_i, Q_i) \cup Ann(z_i, Q'_i)$.
   (c) If $type(g, XOR)$ and $z_i \equiv z_j$, then warn: Monitoring violation after g.
   (d) If $type(g, XOR)$ and $z_i \sqsubseteq z_j$, then warn: Potential monitoring violation after g.
3. For each splitting gateway g preceded by an event z_g and followed by two or more sibling events z_i,
   (a) If $type(g, AND)$ and $z_i \sqsubseteq z_g$ for all z_i, then absorb z_g.
   (b) If $type(g, XOR)$ and $z_i \sqsubseteq z_g$ for at least two z_i, then warn: Monitoring violation after g.

In this procedure, the *absorption* of a node n_i means that all the arcs $n_j \to n_i$ must be replaced by arcs $n_j \to n_k$ for each n_k given by every arc $n_i \to n_k$ in the diagram. Then node n_i as well as all its incoming and outgoing arcs are removed from the diagram.

Given that all refinement strategies are local, the complexity of the previous procedure is bounded by the computation of query containment between annotations of sibling event nodes. The identification of a common precondition for a set of sibling events z_i requires checking if exists a common subset of triplets Q'_i in every Q_i.

As long as query containment is not a standard reasoning service provided by state-of-the-art triplestores or DL reasoners it would be necessary to develop a tool that efficiently decide query containment among a set of CQs. Such a service would decide $Q_1 \sqsubseteq Q_2$ by transforming Q_2's variables into symbols, asserting the resulting statements in a graph and asking Q_1 to it (using the desired reasoning level).

4 Conclusions

We introduced a taxonomy of semantic descriptors for annotating lanes, events and tasks in BPMN BPDs. The proposed notation not only provides a formal

description of these elements but it can be also used for mapping the business process specification to its implementation in an agent-based software.

We demonstrated how using conjunctive queries as semantic descriptors can help to detect relationships between BPMN BPD events. These relationships are then used for refining process specifications by detecting disjoint alternative paths, deleting redundant nodes, and merging/splitting nodes.

We also illustrated why human actions should not only be represented through their immediate effects, but breaking them down helps Multiagent System.to determine whether the activity takes place according to how it was modeled.

Semantic descriptors can be further used for determining common events or actions across diagrams, enabling the composition of BPDs. Diagram composition would enable to introduce the execution of MAS protocols in human activities, both modeled through BPDs. Semantic descriptors can be also further used for monitoring process/activity development by inquiring a RDF triple store representing world state.

References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, New York, NY, USA (2003)
2. Ceballos, H.G., Flores-Solorio, V., Garcia, J.P.: A Probabilistic BPMN Normal Form to Model and Advise Human Activities. In: Baldoni, M., Baresi, L., Dastani, M. (eds.) Engineering Multi-Agent Systems: Third International Workshop, EMAS 2015, Istanbul, Turkey. LNAI 9318 (2015)
3. Ceballos, H.G., Garcia-Vazquez, J.P., Brena, R.: Using activity theory and causal diagrams for designing multiagent systems that assist human activities. In: Proceedings of the 12th Mexican International Conference on Artificial Intelligence, MICAI 2013. pp. 185–198. Mexico City (Nov 2013)
4. Endert, H., Kuster, T., Hirsch, B., Albayrak, S.: Mapping BPMN to agents: An analysis. In: Agent, Web Services, and Ontologies Integrated Methodologies - International Workshop MALLOW-AWESOME 2007. pp. 43–58 (2007)
5. group, W.S.W.: SPARQL 1.1 Overview (2013), `http://www.w3.org/TR/sparql11-overview/`
6. Hinge, K., Ghosey, A., Koliadisz, G.: Process seer: A tool for semantic effect annotation of business process models. In: Proceedings of 13th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2009. pp. 54–63 (2009)
7. Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A semantic web rule language combining OWL and RuleML. W3C Member Submission 21 May 2004
8. Horrocks, I., Sattler, U., Tessaris, S., Tobies, S.: How to decide query containment under constraints using a description logic. In: Logic for Programming and Automated Reasoning, 7th International Conference, LPAR 2000, Reunion Island, France, November 11-12, 2000, Proceedings. pp. 326–343 (2000)
9. Kuster, T., Lutzenberger, M., Albayrak, S.: A Formal Description of a Mapping from Business Processes to Agents. In: Baldoni, M., Baresi, L., Dastani, M. (eds.) Engineering Multi-Agent Systems: Third International Workshop, EMAS 2015, Istanbul, Turkey. LNAI 9318 (2015)

10. Leont'ev, A.: Activity, Consciousness, and Personality. Prentice-Hall, Inc. (1978)
11. Mitakides, N., Delias, P., Spanoudakis, N.: Validating Requirements Using Gaia Analysis Models. In: Baldoni, M., Baresi, L., Dastani, M. (eds.) Engineering Multi-Agent Systems: Third International Workshop, EMAS 2015, Istanbul, Turkey. LNAI 9318 (2015)
12. zur Muehlen, M., Indulska, M.: Modeling languages for business processes and business rules: A representational analysis. Information Systems 35(4), 379–390 (2010)
13. OMG: Business Process Model and Notation (BPMN), Version 2.0 (January 2011), http://www.omg.org/spec/BPMN/2.0
14. Ouyang, C., van der Aalst, W., Dumas, M., Hofstede, A.: Translating BPMN to BPEL. BPM Center Report BPM-06-02, BPMcenter.org (2006)
15. Shapiro, R., Gagne, D.: XML Process Definition Language (XPDL). Tech. Rep. WFMC-TC-025, Workflow Management Coallition (8 2012), http://www.omg.org/spec/BPMN/2.0/