Working Paper, Version of August 16, 2003, Submitted to Conference

Beyond Monotonic Inheritance: Towards Semantic Web Process Ontologies[α]

Abraham Bernstein[+]
[+]Department of Information Technology
University of Zurich
Zurich, Switzerland
avi@acm.org
http://www.ifi.unizh.ch/~bernstein

Benjamin N. Grosof[*]
[*]Sloan School of Management
Massachusetts Institute of Technology
Cambridge, MA, USA
bgrosof@mit.edu
http://www.mit.edu/~bgrosof

α: NB: Order of authorship is alphabetic.

[α] NB: order of authorship is alphabetic.

Abstract

Semantic Web Services (SWS), the convergence of Semantic Web and Web Services, is the emerging next major generation of the Web, in which e-services and business communication become more knowledge-based and agent-based. In the SWS vision, service descriptions are built partly upon *process ontologies* – widely shared ontological knowledge about business processes – which are represented using Semantic Web techniques for declarative knowledge representation (KR), e.g., OWL Description Logic or RuleML Logic Programs.

In this paper, we give the first approach to solving a previously unsolved, crucial problem in representing process ontologies using *SW KR*: how to represent *non-monotonic* inheritance reasoning, in which at each (sub)class in the class hierarchy, any inherited property value may be *overridden* with another value, or simply *cancelled* (i.e., not inherited). Non-monotonic inheritance is an important, heavily-used feature in pre-SWS process ontologies, e.g., ubiquitous in object-oriented (OO) programming. The advantages of non-monotonicity in inheritance include greater reuse/modularity and easier specification, updating, and merging. We focus in particular on the Process Handbook (PH), a large, influential, and well-used process ontologies repository that is representative in its features for non-monotonic inheritance. W3C's OWL, the currently dominant SW KR for ontologies, is fundamentally incapable of representing non-monotonicity; so too is First Order Logic. Using instead another form of leading SW KR – RuleML – we give a new approach that successfully represents the PH's style of non-monotonic inheritance. In this *Courteous Inheritance* approach, PH ontology knowledge is represented as prioritized default rules expressed in the Courteous Logic Programs (CLP) subset of RuleML.

A prototype of our approach is in progress. We aim to use it to enable SWS exploitation of the forthcoming open-source version of the PH.

More Introduction: SWS, KR, Ontologies, and Inheritance

In the SWS vision (see http://www.swsi.org), semantic service descriptions, i.e., process knowledge represented using Semantic Web (SW) KR, will enable various kinds of reasoning in support of several *key tasks* including advertising/discovery, matchmaking, negotiation/contracting, composition, execution monitoring, etc., of Web Services. Some early, relatively simple SWS prototypes have already been developed (The DAML Services Coalition 2001) (Grosof et al. 2003a). To realize the fuller glory of the vision, however, SWS developers will need large amounts of process knowledge that is represented declaratively[1], i.e., in semantically clean fashion cf. SW KR. A key part of process knowledge is *ontological*: definitional knowledge in the form of class hierarchies (where some (sub)classes are specializations of superclasses) and property information associated with those classes. A central kind of reasoning in ontologies is inheritance, in which each subclass (specialization) inherits the values of the properties associated with its superclasses (generalizations). In *monotonic* inheritance, a subclass can *add* new properties and/or property values. Inheritance is useful since it enables reuse: to specify a new subclass's set of associated property info – one can just specify the *differences* from its superclass(es)'s property info.

In practice, however, users of inheritance often – perhaps, usually – desire *non-monotonic*[2] inheritance, which is more general. In non-monotonic inheritance, unlike monotonic inheritance, one can *change* what's inherited: either to replace/override (i.e., *modify*) or to cancel (i.e., *delete*) an inherited property value. Non-monotonic inheritance is a.k.a. *default* inheritance or inheritance *with exceptions*. Permitting inheritance to be non-monotonic enables more reuse/modularity. When inheritance is restricted to be monotonic, if one needs to change (override or cancel) some inherited property info, then one has to manually/explicitly repeat the specification of much or all of the property info that one wants

[1] See Appendix A for review of "declarative" in basic KR terminology
[2] See Appendix A for review of "monotonic" and "non-monotonic" in basic KR terminology

to reuse from previously-defined classes, without benefit of inheritance. Non-monotonic inheritance thus enables easier specification: not only initially but also especially during updating and merging (e.g., in maintenance). Non-monotonic inheritance is found ubiquitously in OO programming languages and design tools, including Java, C++, and the PH. OWL (Van Harmelen et al. 2003) can represent expressively complex kinds of monotonic inheritance but is fundamentally incapable of representing any kind of non-monotonic reasoning, including non-monotonic inheritance. Ditto for classical / First Order Logic.

The Process Handbook and its Ontology Representation

The Process Handbook (PH) has been under development at the MIT Center for Coordination Science (CCS) for over ten years, including the contributions of a diverse and highly distributed group of over 40 scientists, students and industrial sponsors (Malone et al. 1999; Malone et al. 2003). The goal of the PH project is to develop a repository and associated conceptual tools that help users effectively retrieve and exploit the process knowledge relevant to their current challenges. The PH is large: it contains, to date, descriptions of over 5000 processes, 12000 entities, and 38000 properties/values. The PH has been used by a number of practical industrial process designers, partly via a dedicated commercial spinoff (Phios Corp.). The PH's process ontologies have been shown, at a research level, to be useful in a variety of domains such as business process reengineering (Bernstein 1998; Malone et al. 1999), e-contracting (Grosof et al. 2002), business process automation (Bernstein 2000), software design (Dellarocas 1996), etc. This provides evidence that PH process ontologies could be used in future to help supply the semantic service descriptions for the key SWS tasks discussed earlier. The PH has been influential as well in the arena of industry standards for process representation/modeling: it spawned Process Interchange Format (PIF) (Lee et al. 1996) which in turn spawned Process Specification Language (PSL) (Schlenoff et al. 2000), a draft NIST/ISO standard.

Overall, the PH includes a class hierarchy with associated properties, and features non-monotonic inheritance of the kind discussed earlier. Experience within the PH project has reconfirmed findings (MacLean et al. 1990) that allowing *cancellation* aids ease of understanding/specification. The PH, like nearly all process representation techniques, also uses the notion that a process can be decomposed into *parts*, called "sub-activities".

Figure 1 illustrates an example PH ontology that employs non-monotonic inheritance. Here, the generic activity (i.e., process) called "Sell product" is decomposed into sub-activities like "Identify potential customer," "Inform potential customer," and "Deliver Product" (among others). Formally, the decomposition relationship is represented as a property "has-task". Because this property has mutiple values associated with the same subject activity (here "Sell product"), a corresponding *slot* is defined for each value, e.g., "1," "2," and "4," respectively, assuming we would name the slots in the Figure from left to right with ascending numbers. "has-task" is thus a *slotted* property; it has arity 3, rather than arity 2 as do ordinary properties. Both "Sell by mail order" and "Sell in retail store," which are specializations of "Sell Product," inherit those has-task relationships with their respective slot numbers. Some relationships are *modified* in the specialized processes and, therefore, colored grey. In "Sell by mail order," for example, the has-task arc with slot name "1" connects to "Obtain mailing list" since that replaces "Identify potential customer." For "Sell in retail store" the has-task arc with slot name "2" was even *deleted* (shown with a red X in Figure).

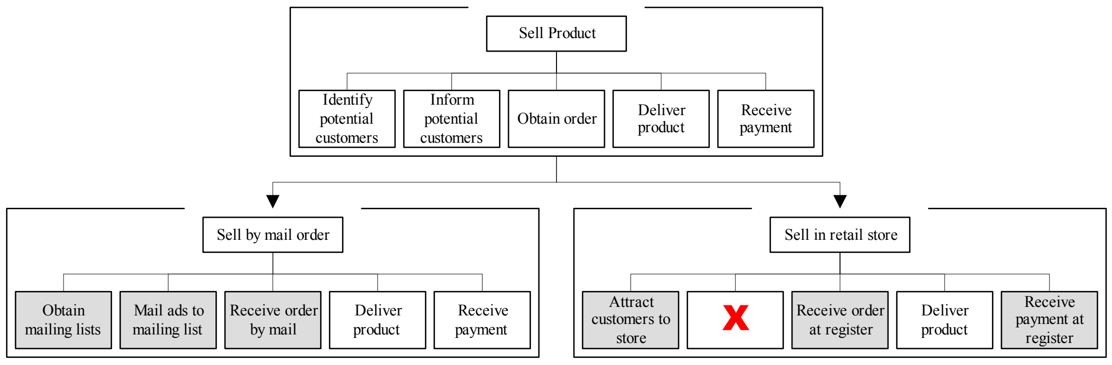

Figure 1: An example of inheritance in the specialization hierarchy.

The PH adheres to the meta-model shown in Figure 2. All classes are subclasses of a common root/top-level ancestor class called *Entity*. Other high-level classes are shown in the boxes in the Figure. *Activities* represent processes (or tasks). *Resources* represent things used in Activities. *Ports* stand for the interfaces that Activities use to exchange Resources with their environment (e.g., with other Activities). *Dependencies* denote a need for coordination when a resource is shared by more than one Activity. *Exceptions* flag things that can go wrong in an Activity. *Attributes* are used to provide more details about all Entities through specific *Values*. Various relationships, such as the *has-task* property, connect these classes.

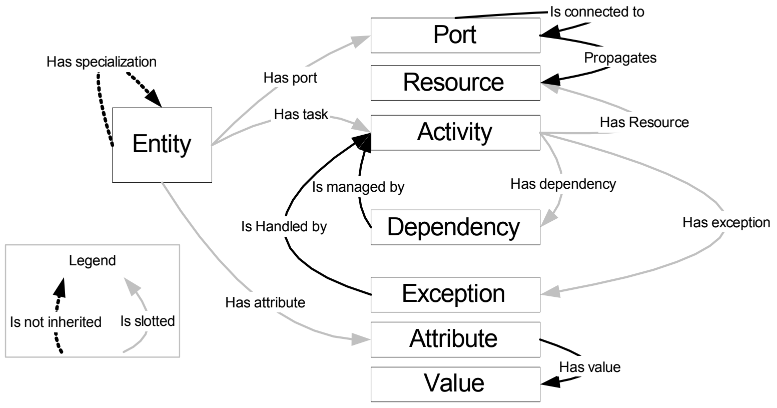

Figure 2: The process handbook classes and their relationships (specialization relationships from all classes to Entity is omitted)

New Approach: Courteous Inheritance, using CLP RuleML

We have developed an approach to representing the Process Handbook's process ontology knowledge, including its non-monotonic inheritance behavior, by using an existing SW KR: RuleML (http://www.ruleml.org), specifically its Courteous Logic Programs (CLP) subset (Grosof et al., 1999). Due to space limitations, in this paper we briefly describe the basic part of this approach, which we call "*Courteous Inheritance*". More details are in a forthcoming longer version of this paper.

Each rule in CLP is a prioritized default. CLP expressively extends Ordinary Logic Programs (roughly, declarative pure Prolog), but is tractably compileable to it. Rule labels (optional) identify rules for prioritized conflict handling. A rule label has the form of a logical term. A syntactically reserved (but otherwise ordinary) predicate "overrides" specifies the prioritization ordering: overrides(lab1,lab2) means that (any rule labeled) "lab1" has strictly higher priority than "lab2". A pairwise mutual exclusion integrity constraint ("mutex") specifies the scope of conflict; it can be conditional. RuleML, like most XML, is fairly verbose. For brevity and ease of human-readability, we give our RuleML example rules in a Prolog-like syntax that maps straightforwardly to RuleML. More precisely, this syntax is IBM CommonRules V3.0 "SCLPfile" format, extended to support URI's as logical predicate (and function) symbols (see, e.g., (Grosof et al. 2003a)). "←" stands for implication, i.e., "if". ";" ends a rule statement. A fact is a special case of a rule; it omits the "← …". The prefix "?" indicates a logical variable. "/* … */" encloses a comment. "<…>" encloses a rule label (name). "~" stands for negation-as-failure. "MUTEXHEAD ← … " specifies a mutex. In it, "MUTEXHEAD" can be read roughly as "logical contradiction". " 'q " stands for the individual that is the name of q, where q is a predicate (e.g., class or property).

Our courteous inheritance approach represents PH ontology knowledge, e.g., subclass relationships and property values, as CLP rules. The approach also represents the PH's inheritance principles as rules, including rules about prioritization and cancellation. The approach includes both background knowledge and particular knowledge. Figure 3 gives details, in the form of sample rules that represent the various kinds of PH ontology knowledge. Taken together, these rules formalize non-monotonic inheritance of slotted properties. Not shown are the rules for inheritance of ordinary (i.e., 2-ary) properties; these are similar but simpler. Rules 1-20 give the basic approach, which permits modification but not cancellation. Adding Rules 21-26 provides cancellation as well (Rule 11 is then omitted).

```
/* particular:  specify c2 is a subclassof c1, c3 subclassof c2 */
 1  subclassof('c2,'c1);
 2  subclassof('c3,'c2);
/* particular:  specify a default value v1 for slot f1 of property ps,
for subject class c1 */
 3  <ds1>   ps('c1,f1,v1);
 4  assoc-class(ds1,'c1);
 5  abouts(ds1,'ps,f1);
 6  value-of(ds1,v1);
/* background:  define subclassof's transitive closure */
```

```
 7  subclassof-tc(?x,?z) ← subclassof(?x,?y) and subclassof-tc(?y,?z);
 8  subclassof-tc(?x,?y) ← subclassof(?x,?y);
/* background:  declare name of each property ps (slotted) */
 9  slotted-property('ps);
/* background:  specify each slotted property is partial functional */
 10 MUTEXHEAD ← ps(?x,?n,?y) and ps(?x,?n,?z) GIVEN notEquals(?y,?z);
/* background:  basic inheritance for each property ps (slotted) */
 11 <inherit-s('ps,?cb,?n,?v)>
            ps(?ca,?n,?v) ← ps(?cb,?n,?v) and subclassof-tc(?ca,?cb);
 12 assoc-class(inherit-s('p,?cb,?n,?v),?cb);
 13 abouts(inherit-s('ps,?cb,?n,?v),'ps,?n);
 14 value-of(inherit-s('ps,?cb,?n,?v),?v);
/* background:  specificity prioritization based on source (slotted) */
 15 <specificity-of-source>
        overrides(?r2,?r1) ← more-specific-source(?r2,?r1);
 16 more-specific-source(?r2,?r1)
        ← assoc-class(?r1,?y1) and assoc-class(?r2,?y2)
            and subclassof-tc(?y2,?y1)
             and abouts(?r1,?z,?n) and abouts(?r2,?z,?n);
/* background:  it is a conflict for one inherited value to be a strict
specialization of another */
 17 MUTEXHEAD ← ps(?c,?n,?vx) and ps(?c,?n,?vy)
                      GIVEN subclassof-tc(?vx,?vy)and notEquals(?vx,?vy);
 /* background:  specificity prioritization based on value (slotted) */
 18 <specificity-of-value>
      overrides(?r2,?r1) ← more-specific-value(?r2,?r1);
 19 more-specific-value(?r2,?r1)
        ← abouts(?r1,?z,?n) and abouts(?r2,?z,?n)and value-of(?r2,?x)
            and value-of(?r1,?y) and subclassof-tc(?x,?y);
 20 overrides(specificity-of-source, specificity-of-value);
/* background:  inheritance for each property ps, with cancellation
(slotted) ; these rules 21-22 replace the “basic inheritance” rule 11 */
 21 <inherit-s('ps,?cb,?n,?v)>
            ps(?ca,?n,?v) ← ps(?cb,?n,?v) and subclassof-tc(?ca,?cb)
            and ~ cancel(inherit-s('ps,?cb,?n,?v));
 22 <inherit-s('ps,?cb,?n,?v)>  ncancel(inherit-s('ps,?cb,?n,?v));
/* background:  ncancel specifies non-cancellation */
 23 MUTEXHEAD ← cancel(?r) and ncancel(?r);
/* particular:  cancel inheritance of (slotted) property ps for slot f1
at subject class c2 */
 24 <cs2>   cancel(?r) ← abouts(?r,'ps,f1);
 25         assoc-class(cs2,'c2);
 26         abouts(cs2,'ps,f1);
```

Figure 3: CLP representation of the PH and its inheritance behavior

A given PH process ontology knowledge base is thus represented as a CLP RuleML rulebase. The entailed conclusions of that rulebase then capture (i.e., correspond closely to) the inferences sanctioned by the PH's inheritance scheme. Any rule system/tool that implements CLP RuleML inferencing (there are several such already existing, and more being built) can then practically perform these inferences.

Example illustrating Courteous Inheritance Approach: Selling

Next, we show how to represent the PH process ontology knowledge from Figure 1 in our approach. Figures 4-7 give details, in the form of four particular-knowledge rulesets (modules), one per process class. In the comments, “cut” means omitted for the sake of brevity and focus.

```
1     <t1> has-subtask('sell-product,g1,'identify-potential-customers);
2          assoc-class(t1,'sell-product);
3          abouts(t1,'has-subtask,g1);
4          value-of(t1,'identify-potential-customers);
5     <t2> has-subtask('sell-product,g2,'inform-potential-customers);
6     /* cut: rules about t2 analogous to rules 2-4 */
7     <t3> has-subtask('sell-product,g3,'obtain-orders);
8     /* cut: rules about t3 analogous to rules 2-4 */
9     <t4> has-subtask('sell-product,g4,'deliver-product);
10    /* cut: rules about t4 analogous to rules 2-4 */
11    <t5> has-subtask('sell-product,g5,'receive-payment);
12    /* cut: rules about t5 analogous to rules 2-4 */
```

Figure 4: Ruleset for Subtasks of “Sell product”

```
13    subclassof('sell-by-mail-order,'sell-product);
14    <m1> has-subtask('sell-by-mail-order,g1,'obtain-mailing-lists);
15         assoc-class(m1,'sell-by-mail-order);
16         abouts(m1,'has-subtask,g1);
17         value-of(m1,'obtain-mailing-lists);
18    <m2> has-subtask('sell-by-mail-order,g2,'mail-ads-to-mailing-
   list);
19    /* cut: rules about m2 analogous to rules 12-14 (or to 2-4) */
20    <m3> has-subtask('sell-by-mail-order,g3,'receive-order-by-mail);
21    /* cut: rules about m3 analogous to rules 12-14 (or to 2-4) */
```

Figure 5: Ruleset for “Sell by mail order” as subclass of “Sell product” with necessary changes

The CLP RuleML rulebase then entails, as desired, that the fourth subtask (slot g4) of “Sell by mail order” is “Deliver product”, i.e., it has been inherited without change from “Sell product”. Likewise, the fifth subtask is also inherited without change.

```
22    subclassof('sell-in-retail-store,'sell-product);
      /* the 2nd subtask / g2 slot is CANCELLED rather than replaced */
23    <r2> cancel(?r) <- abouts(?r,'sell-in-retail-store,g2);
24         assoc-class(r2,'sell-in-retail-store);
25         abouts(r2,'has-subtask,g2);
26    /* cut: rules about the other subtasks/slots that are modified
                (i.e., the 1st/g1, 3rd/g3, 5th/g5) */
```

Figure 6: Ruleset for “Sell in retail store” as subclass of “Sell product” (only cancellation shown)

Let there additionally be a new class “Sell clothes in retail store” that is a specialization of “Sell in retail store”:

```
subclassof('sell-clothes-in-retail-store,'sell-in-retail-store);
```

Figure 7: Ruleset for “Sell clothes in retail store” as subclass of “Sell in retail store”

Then, as desired, the CLP RuleML rulebase (particular plus background knowledge) entails that the subtasks of “Sell clothes in retail store” are the same as for “Sell in retail store”, i.e., they are inherited without change.

Discussion, Conclusions, and Future Work

We gave a new approach, Courteous Inheritance, that for the first time formalizes the PH's process ontology knowledge, *including* its *non-monotonic* inheritance principles, in terms of a leading *Semantic Web KR*: RuleML, specifically its CLP subset. Ours is further, more generally, (to our knowledge) the first approach to representing the non-monotonic inheritance aspect of process ontology reasoning using a SW KR. This achieves a significant step towards meeting the requirements of SWS for process ontologies as a foundation, since practical process ontologies typically rely heavily on non-monotonic inheritance similar to (or somewhat simpler than) the PH's.

By contrast, the previous approaches to represent process ontologies using SW KR are based on OWL (DAML-S effort) or FOL (PIF and PSL efforts), and thus are fundamentally incapable of representing non-monotonicity. Part of the PH's process knowledge had previously been represented in PIF (Malone et al. 2003).

As (Grosof et al. 2003b) have shown, however, CLP RuleML actually overlaps expressively with OWL: the monotonic Description Logic Programs (DLP) KR is an expressive subset of both. An interesting direction of current work, extending DLP and in the spirit of our approach here, is to represent non-monotonicity of inheritance for general-purpose Semantic Web ontologies, not just process ontologies. Another such direction is to represent in OWL the monotonic fragment of the PH's (knowledge and) inheritance.

Moreover, aside from the *form* of the PH's knowledge and reasoning, the *content* of the PH constitutes a large and interesting repository of process knowledge. The PH project is planning to release an open-source version ("OpenPH"), expected in 2004. A prototype of our approach is in progress. We aim in future work to use it to aid exploitation for SWS of the OpenPH.

Bringing the PH to the Semantic Web party will enable its large knowledge base to be integrated relatively easily with a variety of other sources of knowledge and to help develop powerful knowledge-based applications, e.g., SWS. More details are in a forthcoming longer version of this paper.

Appendix A: Review of some basic KR Terminology

In a *declarative* KR, a given set of premises entails (sanctions) a set of conclusions, independently of how inferencing is performed, i.e., independent of control strategy / algorithm / implementation, e.g., regardless of whether inferencing is done backward (query answering) or forward (data driven), or whether inferencing is incomplete (wrt entailment). Different KR's permit different kinds of premises to be expressed, and have different principles for entailing conclusions.

A KR *S* is said to be (logically) *monotonic* when adding new premises never results in retracting old conclusions, i.e., iff $\forall$P1,P2. P1 $\subseteq$ P2 $\Rightarrow$ Concl(P1) $\subseteq$ Concl(P2) where each Pi is a set of premises, $\subseteq$ stands for subset-of, and Concl maps a set of premises into its corresponding set of conclusions (that are entailed according to *S*). Classical logic, used for the foundations of mathematics, is monotonic. However, typical patterns of human pragmatic reasoning are, in general, *non*-monotonic (i.e., not monotonic).

References

Bernstein, A. "The Product Workbench: An Environment for the Mass-Customization of Production-Processes," Workshop on Information Technology and Systems (WITS), Helsinki, Finland, 1998.

Bernstein, A. "How can cooperative work tools support dynamic group processes? Bridging the specificity frontier," Computer Supported Cooperative Work, ACM Press, Philadelphia, PA, 2000.

Dellarocas, C. "A coordination Perspective on Software Architecture: Towards a design Handbook for Integrating Software Components," Ph.D. Thesis, *Department of Electrical Engineering and Computer Science*, Massachusetts Institute of Technology, Cambridge, MA, 1996, p. 288.

Grosof, B., Labrou, Y., and Chan, H.C., "A Declarative Approach to Business Rules in Contracts: Courteous Logic Programs in XML", Proc. 1st ACM Conf. on Electronic Commerce (EC-99), 1999.

Grosof, B., and Poon, T.C. "Representing Agent Contracts with Exceptions using XML Rules, Ontologies, and Process Descriptions," Proc. 12th Intl. Conf. on World Wide Web (WWW-2003), Budapest, Hungary, 2003; extended working paper submitted to journal.

Grosof, B., Horrocks, I., Volz, R., and Decker, S., "Description Logic Programs: Combining Logic Programs with Description Logic", Proc. 12th Intl. Conf. on World Wide Web (WWW-2003), Budapest, Hungary, 2003.

Lee, J., Grunninger, M., Jin, Y., Malone, T., Tate, A., and Yost, G. "The PIF Process Interchange Format and Framework Version 1.1," Workshop on Ontological Engineering, ECAI '96, Budapest, Hungary, 1996.

MacLean, A., Carter, K., Lövstrand, L., and Moran, T. "User-tailorable Systems: Pressing the Issues with Buttons," Human Factors in Computing Systems, ACM-SIGCHI, Seattle, Washington, 1990.

Malone, T.W., Crowston, K., Lee, J., Pentland, B., Dellarocas, C., Wyner, G., Quimby, J., Osborn, C., Bernstein, A., Herman, G., Klein, M., and O'Donnell, E. "Tools for Inventing Organizations: Toward a Handbook of Organizational Processes," *Management Science* (45:3) 1999, pp 425-443.

Malone, T.W., Crowston, K., and Herman, G., eds., "Organizing Business Knowledge: The MIT Process Handbook", MIT Press, Sept. 2003.

Schlenoff, C., M., G., Tissot, F., Valois, J., Lubell, J., and Lee, J. "The Process Specification Language (PSL): Overview and Version 1.0 Specification," NISTIR 6459, National Institute of Standards and Technology, Gaithersburg, MD, p. 73, 2000.

The DAML Services Coalition, Ankolekar, A., Burstein, M., Hobbs, J.R., Lassila, O., Martin, D.L., McIlraith, S.A., Narayanan, S., Paolucci, M., and Payne, T. "DAML-S: Semantic Markup For Web Services," The First Semantic Web Working Symposium, Stanford University, California, USA, 2001, pp. 411-430.

Van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D., Patel-Schneider, P., and Stein, L.A. "OWL Web Ontology Language Reference," WD-owl-ref-20030331, W3C Working Draft, World Wide Web Consortium, Cambridge, MA, 2003.