Using the PSL Ontology

Michael Grüninger

Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, ON, Canada, gruninger@mie.utoronto.ca

1 Introduction

Representing activities and the constraints on their occurrences is an integral aspect of commonsense reasoning, particularly in manufacturing, enterprise modelling, and autonomous agents or robots. In addition to the traditional concerns of knowledge representation and reasoning, the need to integrate software applications in these areas has become increasingly important. However, interoperability is hindered because the applications use different terminology and representations of the domain. These problems arise most acutely for systems that must manage the heterogeneity inherent in various domains and integrate models of different domains into coherent frameworks. For example, such integration occurs in business process reengineering, where enterprise models integrate processes, organizations, goals and customers. Even when applications use the same terminology, they often associate different semantics with the terms. This clash over the meaning of the terms prevents the seamless exchange of information among the applications. translators between every pair of applications that must cooperate. What is needed is some way of explicitly specifying the terminology of the applications in an unambiguous fashion.

The Process Specification Language (PSL) ([6, 9]) has been designed to facilitate correct and complete exchange of process information among manufacturing systems.[1] Included in these applications are scheduling, process modeling, process planning, production planning, simulation, project management, workflow, and business process reengineering. This chapter will give an overview of the PSL Ontology, including its formal characterization as a set of theories in first-order logic and the range of concepts that are axiomatized in these theories.

[1] PSL has been published as the International Standard ISO 18629 by the International Organisation of Standardisation.

S. Staab and R. Studer (eds.), *Handbook on Ontologies,* International Handbooks on Information Systems, DOI 10.1007/978-3-540-92673-3,

2 How are Ontologies Used?

Applications of ontologies focus on their role as sharable and reusable representations of knowledge (Chapters "What is an *Ontology*?", "Ontology-Based Recommender Systems". Semantic heterogeneity is particularly acute problem for tasks that require correct and meaningful communication and integration among software systems, since different systems may ascribe disparate meanings to the same terms or use distinct terms to convey the same meaning. Ontologies support semantic integration through a shared understanding of the intended semantics of the terminology used by the software systems.

The reusability of an ontology is determined relative to the genericity of its axiomatization. In one sense, the axioms of the ontology can be instantiated within different domains; this leads to the notion of domain theories that capture the knowledge for particular problems. In another sense, the axioms of the ontology capture those properties of the world that are valid across multiple domains; new ontologies can then be constructed as more domain-specific extensions of the generic ontologies.

2.1 Specifying Domain Theories

Within the context of a process ontology , domain theories take the form of descriptions of processes as repeatable patterns of behaviour. The various forms of process representations are ubiquitous in industry: there is a plethora of business and engineering software applications – workflow, scheduling, discrete event simulation, process planning, business process modeling, and others – that are designed explicitly for the construction of process models of various sorts [6]. In addition, there are many concrete domains for process representations, including manufacturing, web services, and business processes.

A process ontology provides the underlying semantics for the process terminology that is common to the many disparate domains and software applications. This allows us to evaluate the consistency of process descriptions. In this way, ontologies can be used to support automated reasoning (such as theorem proving and constraint satisfaction) with the axioms of the ontology and domain theories alone.

Ontologies also provide guidance in the specification of domain theories. For example, each class of activities in the PSL Ontology is associated with a specific class of sentences that are the correct process descriptions for that class. The primary focus of this chapter will be a survey of the various classes of activities in the ontology together with examples of the corresponding process descriptions.

2.2 Semantic Integration

A semantics-preserving exchange of information between two software applications requires mappings between logically equivalent concepts in the ontology

of each application. The challenge of semantic integration is therefore equivalent to the problem of generating such mappings (Chapter "Why Is Ontology Mapping Difficult?"), determining that they are correct, and providing a vehicle for executing the mappings, thus translating terms from one ontology into another.

The Twenty Questions approach ([3]) describes a technique for the semi-automatic generation of semantic mappings from application ontologies to the PSL Ontology, which can then be used to automatically derive direct mappings between application ontologies.

The work in [7] describes an example of using PSL as a common ontology to facilitate manufacturing process information exchange between two different software applications, ProCAP – a process modelling tool based upon the IDEF3 method of systems modelling and ILOG – a C++ library for constraint-based scheduling. In a typical scenario, a user of ProCAP describes the types or processes that are necessary to produce some product, specifies the order in which these processes must occur, and describes what types of resources are necessary for the creation of the product. Semantic mappings between the PSL Ontology and the terminology used in IDEF3 and ILOG process descriptions form the basis for translators between the software applications.

2.3 Building New Ontologies

An ontology with a consistent and complete axiomatization of its intended semantics can be used as a semantic foundation for either building a new ontology or for augmenting an ontology that has an incomplete axiomatization (Chapter "Foundational Choices in DOLCE"). For example, the process model of the semantic web services ontology OWL-S (Chapter "Semantic Web Services", [5]) contains a taxonomy of control constructs for specifying composite web services; however, the intended semantics of these constructs is expressed in natural language, since it cannot be axiomatized in OWL. The work in [2] provides a first-order axiomatization of these constructs using the PSL Ontology.

The Semantic Web Services Ontology (SWSO) ([11]) is an extension of the PSL Ontology with Web service-specific concepts which enables reasoning about the semantics underlying Web services and along with their interactions with each other and with the "real world". Because SWSO is an extension of the PSL Ontology, it also provides a first-order axiomatization of the intended semantics of the process model of OWL-S. This supports reasoning with the axioms of the ontology alone, rather than use extra-logical algorithms to guarantee that queries are entailed by the web service specifications.

3 Basic Ontological Distinctions

The PSL Ontology consists of a set of first-order logic theories within which there is a distinction between core theories and definitional extensions.[2] Core theories introduce new primitive concepts, while all terms introduced in a definitional extension that are conservatively defined using the terminology of the core theories.

All core theories within the ontology are consistent extensions of PSL-Core (T_{psl_core}), although not all extensions need be mutually consistent. Table 1 is a summary of the key terms in the lexicon of the eight core theories which will be used in this chapter.

3.1 Activity and Activity Occurrence

In general, business and engineering processes are described at the type level – a process specification characterizes a certain general pattern that might admit of many instantiations which might differ considerably from one another. For example, the specification of the manufacturing process for making a car will describe different sequences of tasks for building the components of the car and may even describe alternative ways of producing subassemblies. A robust foundation for process modelling, therefore, should be able to characterize both the general process pattern described by a specification as well as the class of possible instantiations of that pattern. Moreover, such a foundation must be able to clearly represent the constraints that a process specification places on something's counting as an instantiations of the process, that is, the constraints on process execution.

Within the PSL Ontology, an *activity* is a repeatable pattern of behaviour, while an *activity occurrence* corresponds to a concrete instantiation of this pattern. The relationship between activities and activity occurrences is represented by the $occurrence\_of(o, a)$ relation. Activities may have multiple occurrences, or there may exist activities which never occur. Any activity occurrence corresponds to a unique activity.

In contrast to many object-oriented approaches, activity occurrences are not considered to be instances of activities, since activities are not classes within the PSL Ontology. One can of course specify classes of activities in a process description. For example the term $pickup(x, y)$ can denote the class of activities for picking up some object x with manipulator y, and the term $move(x, y, z)$ can denote the class of activities for moving object x from location y to location z. Ground terms such as $pickup(Block1, LeftHand)$ and

[2] The complete set of axioms for the PSL Ontology can be found at `http://www.mel.nist.gov/psl/psl-ontology/`. Core theories are indicated by a .th suffix and definitional extensions are indicated by a .def suffix.

All axioms and definitions in the PSL Ontology are written in CLIF (Common Logic Interchange Format).

Table 1. Lexicon for core theories in the PSL Ontology

T_{psl_core}	$activity(a)$	a is an activity
	$activity\_occurrence(o)$	o is an activity occurrence
	$timepoint(t)$	t is a timepoint
	$object(x)$	x is an object
	$occurrence\_of(o, a)$	o is an occurrence of a
	$beginof(o)$	the beginning timepoint of o
	$endof(o)$	the ending timepoint of o
	$before(t_1, t_2)$	timepoint t_1 precedes timepoint t_2 on the timeline
$T_{subactivity}$	$subactivity(a_1, a_2)$	a_1 is a subactivity of a_2
	$primitive(a)$	a is a minimal element of the *subactivity* ordering
T_{atomic}	$atomic(a)$	a is either primitive or a concurrent activity
	$conc(a_1, a_2)$	the activity that is the concurrent composition of a_1 and a_2
$T_{occtree}$	$legal(s)$	s is an element of a legal occurrence tree
	$earlier(s_1, s_2)$	s_1 precedes s_2 in an occurrence tree
T_{disc_state}	$holds(f, s)$	the fluent f is true immediately after the activity occurrence s
	$prior(f, s)$	the fluent f is true immediately before the activity occurrence s
$T_{complex}$	$min\_precedes(s_1, s_2, a)$	the atomic subactivity occurrence s_1 precedes the atomic subactivity occurrence s_2 in an activity tree for a
	$root(s, a)$	the atomic subactivity occurrence s is the root of an activity tree for a
	$next\_subocc(s_1, s_2, a)$	the atomic subactivity occurrence s_1 is by the atomic subactivity occurrence s_2 in an activity tree for a
T_{actocc}	$subactivity\_occurrence(o_1, o_2)$	o_1 is a subactivity occurrence of o_2
	$root\_occ(o)$	the initial atomic subactivity occurrence of o
	$leaf\_occ(s, o)$	s is the final atomic subactivity occurrence of o
$T_{duration}$	$timeduration(d)$	d is a timeduration
	$duration(t_1, t_2)$	the timeduration whose value is the "distance" from timepoint t_1 to timepoint t_2

$move(Shipment1, Seattle, Chicago)$ are instances of these classes of activities, and each instance can have different occurrences. Furthermore, there may be classes of activity occurrences that do not correspond to activities, e.g. that class of activity occurrences that finish by Friday.

3.2 Time

Underlying the intuition that activity occurrences are the instantiations of activities is the notion that each activity occurrence is associated with unique timepoints that mark the begin and end of the occurrence. The PSL Ontology introduces two functions $beginof$ and $endof$ for this purpose.

The set of timepoints is linearly ordered, forwards into the future, and backwards into the past. Within the PSL Ontology, the extension of the $before$ relation captures this linear ordering. There are also different ontological commitments about time that are not made within the PSL Ontology, such as the denseness of the timeline; any such commitments must be axiomatized within a theory that extends the PSL Ontology.

There are some approaches (e.g. [4]) that do not distinguish between timepoints and activity occurrences, so that activity occurrences form a subclass of timepoints. However, activity occurrences have preconditions and effects, whereas timepoints do not. Other approaches hold that timepoints are primitives but activity occurrences are not; for example, approaches such as [10] claim that one can derive timepoints as "ticks" of a clock activity; however, such an approach ties the temporal ontology too closely to the process ontology.

The core theory $T_{duration}$ for duration adds a metric to the timeline by mapping every pair of timepoints to a new sort called *timeduration* that satisfies the axioms of algebraic fields. Of course, the duration of an activity occurrence is of most interest, and is equal to the duration between the endof and beginof timepoints of the activity occurrence.

3.3 Objects

Many debates have erupted within philosophy over the distinction between objects that are *continuants* (that is, they exist whole and entire at different times) and objects that are *occurrents* (that is, they have different parts existing at different times).[3] Although the PSL Ontology tries to avoid making any commitments that would preclude one position or another in this debate, activity occurrences can be considered to be occurrents, while continuants are represented by *objects*. The ternary relation $participates\_in(x, o, t)$ is used to tie the two approaches together by specifying that object x participates in activity occurrence o at timepoint t.

[3] This terminology is used in [1]. The treatment of objects as continuants is also known as endurantism or 3D-ontology, while the treatment of objects as occurrents is also known as perdurantism or 4D-ontology.

3.4 Composition

A ubiquitous feature of process formalisms is the ability to compose simpler activities to form new complex activities (or conversely, to decompose any complex activity into a set of subactivities). The PSL Ontology incorporates this idea while making several distinctions between different kinds of composition that arise from the relationship between composition of activities and composition of activity occurrences.

Subactivities

The PSL Ontology uses the *subactivity* relation to capture the basic intuitions for the composition of activities. The core theory $T_{subactivity}$ axiomatizes this relation as a discrete partial ordering (such as Fig. 1), in which primitive activities are the minimal elements.

$T_{subactivity}$ alone does not specify any relationship between the occurrence of an activity and occurrences of its subactivities. For example, we can compose the primitive activities *press* and *punch* in Fig. 1 to make the complex activity $surfacing$ and we can also compose them to make a different complex activity *shaping*. However, this specification of subactivities alone does not allow us to say that $surfacing$ is a deterministic activity, or that *shaping* is a nondeterministic activity. The core theory $T_{complex}$ is therefore introduced to characterize the relationship between the occurrence of a complex activity and occurrences of its subactivities.

Concurrency

Concurrency involves more than the fact that two activities occur at the same time, since concurrent activities may have different preconditions and effects than any of the activities if they occur alone. In particular, the activities may have interfering preconditions, so that even if two activities can possibly occur separately, they cannot occur concurrently (e.g. the oven cannot be used to bake a cake and a turkey at the same time) or the effects of two activities may clobber each other, so that the effects of the concurrent activity are different

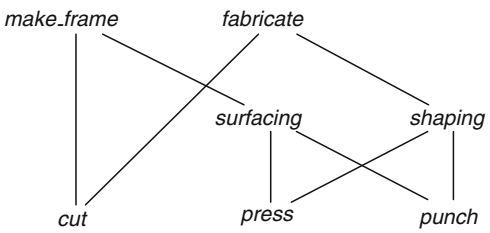

Fig. 1. Example of subactivities and their composition into different complex activities

than the effects of the two activities if they occur separately ([8]); for example, the effect of lifting only the right side or only the left side of a table has the effect that the table is touching the floor. Lifting both the right and left sides concurrently has the effect of lifting the entire table.

This observation leads to a notion of *atomic* activity which corresponds to some set of primitive activities. Concurrency is represented by the occurrence of concurrent activities rather than concurrent activity occurrences. The core theory T_{atomic} axiomatizes the *conc* function that specifies the aggregation of sets of primitive activities into concurrent activities.

Subactivity Occurrences

The core theory T_{actocc} axiomatizes the *subactivity_occurrence* relation, which is the composition relation over activity occurrences corresponding to the composition relation over activities. Occurrences of atomic activities are the minimal elements in this composition ordering – they do not have any nontrivial subactivity occurrences.

Following the intuition that activity occurrences are occurrents rather than continuants, one can consider the subactivity occurrence to be a temporal part of the complex activity occurrence. The axioms of T_{actocc} guarantee that any subactivity occurrence is "during" an occurrence of the complex activity.

3.5 State and Change

Many applications of process ontologies are used to represent dynamic behaviour in the world so that software systems may make predictions about the future and explanations about the past. In particular, these predictions and explanations are often concerned with the state of the world and how that state changes. The PSL core theory T_{disc_state} is intended to capture the basic intuitions about state and its relationship to activities.

Properties in the domain that can change are called *fluents* . Similar to the representation of activities, fluents can also be denoted by terms within the language. For example, $in\_stock(Gadget1, Cambridge)$ denotes the fluent that represents the property that the object $Gadget1$ is available in stock at the Cambridge warehouse.

Intuitively, a change in state is captured by the set of fluents that are either achieved or falsified by an activity occurrence. The $prior(f, o)$ relation specifies that a fluent f is intuitively true prior to an activity occurrence o and the $holds(f, o)$ relation specifies that a fluent f is intuitively true after an activity occurrence o. For example, before a delivery, $Gadget1$ is not in the Cambridge warehouse, but after delivery occurs, it is in stock:

$$\begin{aligned} &occurrence\_of(o, delivery(Gadget1, Cambridge)) \supset \\ &\quad \neg prior(in\_stock(Gadget1, Cambridge), o) \\ &\quad \wedge holds(in\_stock(Gadget1, Cambridge), o) \end{aligned}$$

A fluent is changed by the occurrence of activities, and a fluent can only be changed by the occurrence of activities. Thus, if some fluent holds after an activity occurrence, but after an activity occurrence later along the branch it is false, then an activity must occur at some point between that changes the fluent. This also leads to the requirement that the fluent holding after an activity occurrence will be the same fluent holding prior to any immediately succeeding occurrence, since there cannot be an activity occurring between the two by definition.

State does not change during the occurrence of an atomic activity. Consequently, the PSL Ontology cannot represent phenomena in which some feature of the world is changing as some continuous function of time (hence the name "Discrete State" for the extension). If state changes during an activity occurrence, then it must be an occurrence of a complex activity.

4 Process Descriptions for Atomic Activities

Within the taxonomy of the PSL Ontology, activities are classified according to the kinds of constraints that their occurrences satisfy. A process description for an activity in some class imposes constraints on activity occurrences corresponding to the definition of the class. Classes of atomic activities are defined with respect to constraints that arise from the following two questions:

- Under what conditions does an atomic activity occur?
- How do occurrences of atomic activities change fluents?

A detailed exposition of these constraints requires a closer look at the model theory of the core theory $T_{occtree}$, in particular, the notion of occurrence trees.

4.1 Occurrence Trees

An occurrence tree is a partially ordered set of atomic activity occurrences, such that for a given set of activities, all discrete sequences of their occurrences are branches of the tree. It is important to note that an occurrence tree contains all occurrences of *all* atomic activities; it is not simply the set of occurrences of a particular (possibly complex) activity. Because the tree is discrete, each activity occurrence in the tree has a unique successor occurrence of each activity.

Although occurrence trees characterize all sequences of activity occurrences, not all of these sequences will intuitively be physically possible within the domain. This leads to the notion of the legal occurrence tree, which is the subtree of the occurrence tree that consists only of *possible* sequences of activity occurrences; The $legal(o)$ relation specifies that the atomic activity occurrence o is an element of the legal occurrence tree.

4.2 Constraints on Legal Occurrence

The process descriptions for atomic activities constrain the legal occurrence tree. The general form of such a process description is:

$$(\forall o)\, occurrence\_of(o, a) \wedge legal(o) \supset \Phi(o) \tag{1}$$

where $\Phi(o)$ is a formula that specifies the constraint on the legal activity occurrence. Within the PSL Ontology, different classes of atomic activities correspond to different classes of formulae that are used to instantiate $\Phi(o)$ in the general process description. In particular, we consider cases in which the preconditions are based on state, time, or the occurrence of other activities.

State-Based Preconditions

The most prevalent kind of precondition are markovian preconditions, in which the possibility of occurrence depends only on the state that holds prior to an activity occurrence, e.g.

Mixing is not performed unless the moulding machine is clean.

In this case, the cleanliness of the machine is the state, and the occurrence of the mixing activity depends on whether or not this state holds:

$$(\forall o, x)\, occurrence\_of(o, mixing(x)) \wedge legal(o) \supset prior(clean(x), o) \tag{2}$$

Note that for this particular class of activities, the consequent of the sentence is a formula that contains only *prior* literals.

Time-Based Preconditions

In more general scenarios, there may be temporal preconditions that depend only on the time at which the activity is to occur, such as

The pre-heating operation can only be performed on Tuesday or Thursday.

which is axiomatized as

$$(\forall o, x)\, occurrence\_of(o, preheat(x)) \wedge legal(o) \supset$$
$$(beginof(o) = Tuesday) \vee (beginof(o) = Thursday) \tag{3}$$

The consequent of this process description is a formula that contains only *beginof* literals.

Occurrence Constraints

The possibility of an activity occurrence may depend on the occurrence of other activities. Consider the example:

If we do not fold the metal after fabrication, we need to reheat it

which is axiomatized as

$$(\forall o_1, x)\, occurrence\_of(o_1, reheat(x)) \wedge legal(o_1) \supset$$
$$\neg(\exists o_2)\, occurrence\_of(o_2, fold(x)) \wedge earlier(o_2, o_1) \wedge legal(o_2) \quad (4)$$

In this case, an occurrence of the reheating activity will depend on the condition that there is no earlier legal occurrence of the folding activity.

Time-Based Occurrence Constraints

Preconditions may also take the form of periodic occurrences, e.g.

Drill bits are replaced every 10 min.

$$(\forall o_1, x_1)\, occurrence\_of(o_1, replace(x_1)) \wedge legal(o_1) \supset$$
$$(\exists o_2, x_2)\, occurrence\_of(o_2, replace(x_2)) \wedge earlier(o_2, o_1)$$
$$\wedge legal(o_2) \wedge (duration(beginof(o_2), beginof(o_1)) = 10) \quad (5)$$

In this example, occurrences of the replacement activity depend not only on the occurrence of an earlier replacement activity but also on the time at which that activity occurred.

4.3 Effects

Effects characterize the ways in which activity occurrences change the state of the world. Such effects may be context-free, so that all occurrences of the activity change the same states, or they may be constrained by other conditions. The general form of such a process description is:

$$(\forall o)\, occurrence\_of(o, a) \wedge \Phi(o) \supset holds(f, o) \quad (6)$$

where $\Phi(o)$ is a formula that specifies the constraint on the effects of the activity occurrence.

State-Based Effects

The most common constraint is state-based effects that depend on some context:

If the object is fragile, then it will break when dropped; if the object is elastic, then it will bounce when dropped.

$$(\forall o, x)\, occurrence\_of(o, drop(x)) \wedge prior(fragile(x), o)$$
$$\supset holds(broken(x), o) \quad (7)$$

Time-Based Effects

Although process descriptions for the effects of atomic activities are most often specifying state-based effects, other kinds of constraints also arise in practice, such as time-based effects:

If the rental car is returned after the due date, then the cost includes a late fee

which is axiomatized by

$$(\forall o, x)\ occurrence\_of(o, rental(x)) \wedge before(DueDate, endof(o)) \supset holds(late\_fee(x), o) \quad (8)$$

The effects of the activity occurrence depend only on timepoints – the time at which the activity occurrence ends and the timepoint that is the due date of the rental.

Occurrence-Based Effects

In some cases, the effects depend not only on when the activity occurs, but also on the timepoints at which other activity occurrences begin or end. For example,

If we remove the coffee pot before the brewing activity completes, then the burner will be wet

is axiomatized by

$$(\forall o_1, o_2, x, y)\ occurrence\_of(o_1, brew(x, y)) \wedge occurrence\_of(o_2, remove(x, y)) \wedge before(beginof(o_2), beginof(o_1)) \supset holds(wet(y), o_1) \quad (9)$$

and in this case, the formula in the process description contains multiple variables denoting different activity occurrences, as well as *before* literals.

Duration-Based Effects

For some classes of atomic activities, the effects are dependent on the duration of the activity occurrences. For example,

The time on the clock display will change after holding the button for 3 s

is axiomatized by

$$(\forall o, x)\ occurrence\_of(o, press(x)) \wedge duration(endof(o), beginof(o)) = 3 \supset holds(display(x), o) \quad (10)$$

The effects do not depend on the time at which the activity occurs, so that the formula does not contain any *before* literals.

5 Process Descriptions for Complex Activities

Classes of complex activities are defined with respect to the following two questions:

- What is the relationship between the occurrence of the complex activity and occurrences of its subactivities?
- Under what conditions does a complex activity occur?

An activity may have subactivities that do not occur; the only constraint is that any subactivity occurrence must correspond to a subtree of the activity tree that characterizes the occurrence of the activity.

5.1 Activity Trees

The basic structure that characterizes occurrences of complex activities is the *activity tree*, which is a subtree of the legal occurrence tree that consists of all possible sequences of atomic subactivity occurrences beginning from a root subactivity occurrence. Each branch of an activity tree corresponds to a possible sequence of occurrences of subactivities of the complex activity.

In a sense, an activity tree is a microcosm of the occurrence tree, in which we consider all of the ways in which the world unfolds *in the context of an occurrence of the complex activity.* For example, consider the occurrence tree in Fig. 2, and suppose that an occurrence of the complex activity $make\_frame$

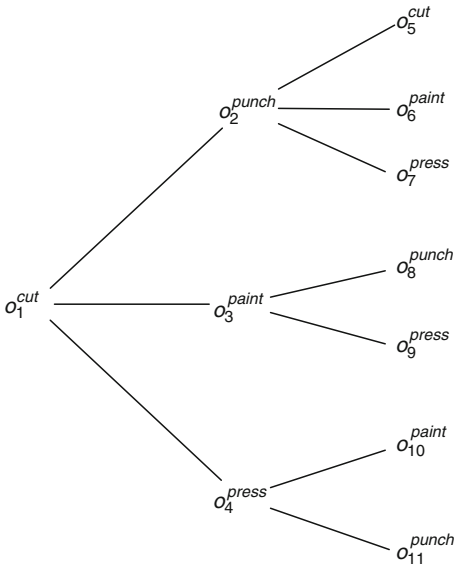

Fig. 2. Example of occurrence tree and activity trees

consists of an occurrence of *cut* followed by occurrences of *punch* and *press*. The subtree consisting of

$$\{o_1^{cut}, o_2^{punch}, o_7^{press}, o_4^{press}, o_{11}^{punch}\}$$

is a possible activity tree for $make\_frame$.

The models of any process description for a complex activity consists of a set of activity trees within an occurrence tree. Each branch of an activity tree is a sequence of atomic subactivity occurrences that satisfies the process description.

Three relations in particular are used in process descriptions for complex activities. The $root(o, a)$ relation specifies that the atomic subactivity occurrence o is the root of the activity tree. The $min\_precedes$ relation is the ordering relation over the atomic subactivity occurrences in the activity tree. In Fig. 2, the activity tree for $make\_frame$ satisfies the process description

$$\begin{gathered}(\forall o)\, occurrence\_of(o, make\_frame) \supset (\exists o_1, o_2, o_3)\, occurrence\_of(o_1, cut) \\ \wedge occurrence\_of(o_2, punch) \wedge occurrence\_of(o_3, press) \\ \wedge root(o_1, make\_frame) \\ \wedge min\_precedes(o_1, o_2, make\_frame) \wedge min\_precedes(o_1, o_3, make\_frame)\end{gathered}$$

The axioms of T_{actocc} guarantees that there is a one-to-one correspondence between branches of activity trees and complex activity occurrences. The axioms for *subactivity_occurrence* relation guarantee that the branches of the activity trees for a subactivity are contained in the branches of the activity tree for the complex activity. In Fig. 2, the branch $\{o_1^{cut}, o_2^{punch}, o_7^{press}\}$ of the activity tree corresponds to an occurrence $o_{12}^{make\_frame}$ of $make\_frame$, and each element of the branch is a subactivity occurrence of $o_{12}^{make\_frame}$.

5.2 Branch Structure

Different subactivities may occur on different branches of the activity tree – different occurrences of an activity may have different subactivity occurrences or different orderings on the same subactivity occurrences.

In this sense, branches of the activity tree characterize the nondeterminism that arises from different ordering constraints or iteration. For example, the $surfacing$ activity is intuitively nondeterministic; the activity trees for $surfacing$ contain two branches, one branch consisting of an occurrence of *polish* and one branch consisting of an occurrence of *paint*.

Complex activities can be classified with respect to symmetries of its activity trees. Concretely, these are axiomatized by relationships between the different branches of an activity tree. We will now take a closer look at the process descriptions for activities in these classes.

Permuted Activities

For permuted activities, each branch of the activity tree is a different permutation of the same set of subactivity occurrences. For example, the informal process description

Making the frame consists of cutting, punching, and pressing. can be formally written as

$$\begin{gathered}(\forall o, x)\, occurrence\_of(o, make\_frame(x)) \\ \supset (\exists o_1, o_2, o_3)\, occurrence\_of(o_1, cut(x)) \\ \wedge occurrence\_of(o_2, punch(x)) \wedge occurrence\_of(o_3, press(x))\end{gathered} \tag{11}$$

If we consider the activity trees that satisfy this sentence (Fig. 3), we can see that each branch contains an occurrence of each subactivity.

Activities may also be nondeterministic; for example, there could be alternative process plans to produce the same product depending on the customer, such as the constraint

Fabrication consists of cutting the metal together with either pressing or punching. which is formally written as

$$\begin{gathered}(\forall o, x)\, occurrence\_of(o, fabricate(x)) \\ \supset (\exists o_1, o_2)\, subactivity\_occurrence(o_1, o) \wedge subactivity\_occurrence(o_2, o) \\ \wedge occurrence\_of(o_1, cut(x)) \\ \wedge (occurrence\_of(o_2, press(x)) \vee occurrence\_of(o_2, punch(x)))\end{gathered} \tag{12}$$

The activity tree in Fig. 4 that satisfies this sentence has branches that contain occurrences of different subactivities.

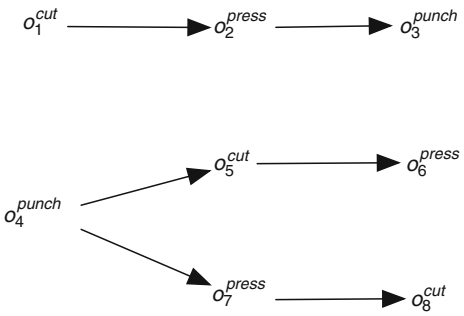

Fig. 3. Activity trees for permuted activities

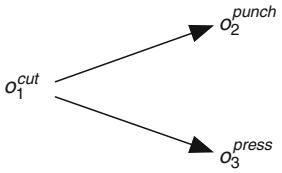

Fig. 4. Activity tree for a nondeterministic activity

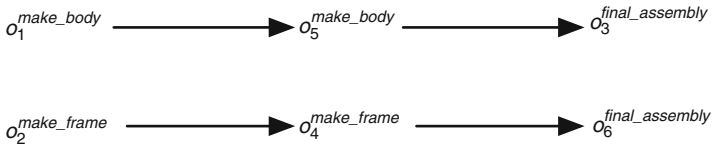

Fig. 5. Activity trees for permuted activities

Ordering Constraints

One of the most common intuitions about processes is the notion of process flow, or the specification of some ordering over the subactivities of an activity, such as

Making the car chassis involves making the body and making the frame in parallel, followed by final assembly.

which is axiomatized by the process description

$$\begin{aligned}(\forall o, o_1, o_2, o_3, x, y)\, occurrence\_of(o, make\_chassis(x, y)) \\ \wedge occurrence\_of(o_1, make\_body(y)) \wedge occurrence\_of(o_2, make\_frame(x)) \\ \wedge occurrence\_of(o_3, final\_assembly(x, y)) \\ \supset min\_precedes(o_1, o_3, make\_chassis(x, y)) \\ \wedge min\_precedes(o_2, o_3, make\_chassis(x, y))\end{aligned} \tag{13}$$

In Fig. 5, we can see that each branch of the activity tree for this activity satisfies the same set of ordering constraints on subactivity occurrences.

Iteration

Iteration is captured by the class of repetitive activities, in which the activity tree can be decomposed into copies of some subtree (which intuitively corresponds to the activity tree of the subactivity that is being iterated).

Nondeterministic iteration, such as

Occurrences of painting consist of multiple occurrences of coating

is axiomatized by a process description of the form

$$(\forall o_1)\, occurrence\_of(o_1, painting) \supset$$
$$((\forall o_2, s_1)\, occurrence\_of(o_2, coating) \wedge subactivity\_occurrence(o_2, o_1)$$
$$\wedge leaf\_occ(s_1, o_2) \supset (leaf\_occ(s_1, o_1) \vee (\exists o_3, s_2)\, occurrence\_of(o_3, coating)$$
$$\wedge (s_2 = root\_occ(o_3)) \wedge next\_subocc(s_1, s_2, painting)) \quad (14)$$

This process description says that for every occurrence of *coating* in an activity tree for *painting*, either there exists a next occurrence of *coating* or the leaf subactivity occurrence of the occurrence of *coating* is also the leaf occurrence of the occurrence of *painting*.

Complex activities in which the number of iterations depends on achieving some state (analogous to **while** loops) is a property of a set of activity trees, as we shall see in the next section.

5.3 Spectrum and Variation

A complex activity will in general have multiple activity trees within an occurrence tree, and not all activity trees for an activity need be isomorphic to each other. This property leads to the notion of the spectrum of an activity, which is the set of equivalence classes of isomorphic activity trees. While the former classes of activities compared branches within the same activity tree, we can also define classes with respect to the spectrum of the activity.

The notion of variation within the PSL Ontology characterizes the conditions under which activity trees for a complex activity are isomorphic to each other. Different activity trees for the same activity can have different subactivity occurrences, or the activity trees may differ on the ordering of the subactivity occurrences.

For conditional activities, the fluents that hold prior to the activity occurrence determine which subactivities occur, as in the constraint

Within the painting activity, if the surface of the product is rough, then sand the product:

which is written as

$$(\forall s, o_1, x)\, occurrence\_of(o_1, paint(x)) \wedge root\_occ(o_1) = s \wedge (prior(rough(x), s)$$
$$\supset (\exists o_2)\, occurrence\_of(o_2, sand(x)) \wedge subactivity\_occurrence(o_2, o_1)$$
$$\wedge (root\_occ(o_2) = s) \quad (15)$$

Alternatively, the ordering over subactivity occurrences of an activity may depend on state, as in the constraint

If the machine is not ready, then perform the painting before final assembly

which can be written as

$$\begin{gathered}(\forall o, o_1, o_2, x, y)\ occurrence\_of(o, assembly(x, y)) \\ \wedge occurrence\_of(o_1, paint(x)) \wedge occurrence\_of(o_2, final(x)) \\ \wedge \neg prior(ready(y), root\_occ(o)) \\ \supset min\_precedes(root\_occ(o_1), root\_occ(o_2), assembly(x)) \end{gathered} \tag{16}$$

Notice how this is distinct from conditional activities, since both painting and final assembly will occur; the different activity trees in this case arise from the ordering of the occurrences of these activities.

5.4 Distribution

The preceding two sections have presented some of the classes in the ontology that are defined with respect to the relationship between occurrences of complex activities and occurrences of their atomic subactivities. We now turn to the classes of complex activities that arise from constraints under which complex activities themselves occur.

There may be branches of a subtree of the occurrence tree that are isomorphic to branches of an activity tree, yet they do not correspond to occurrences of the activity. For example, in Fig. 2, $\{o_1^{cut}, o_8^{punch}\}$ need not be an activity tree for $make\_frame$, even though it is isomorphic to a branch of an activity tree.

The general form for process descriptions related to distribution is:

$$(\forall s)\ \Phi(s) \supset (\exists o)\ occurrence\_of(o, a) \wedge s = root\_occ(o) \tag{17}$$

For triggered activities such as

Deliver the product when we have received three orders.

State determines when an activity must occur, so that the process description is written as

$$\begin{gathered}(\forall s, x)\ prior(order\_quantity(x, 3), s) \supset \\ (\exists o)\ occurrence\_of(o, deliver(x)) \wedge s = root\_occ(o) \end{gathered} \tag{18}$$

For launched activities such as

Deliver the product at 1,000.

Time determines when an activity must occur, leading to the process description

$$\begin{gathered}(\forall s)\ (beginof(s) = 1000) \supset \\ (\exists o, x)\ occurrence\_of(o, deliver(x)) \wedge s = root\_occ(o) \end{gathered} \tag{19}$$

In either case, models of the process description specify the distribution of activity trees within the occurrence tree.

5.5 Embedding Constraints

The PSL Ontology does not force the existence of complex activities; there may be subtrees of the occurrence tree that contain occurrences of subactivities, yet not be activity trees. We can exploit this property to represent the existence of activity attempts, intended effects, and temporal constraints; subtrees that do not satisfy the desired constraints will simply not correspond to activity trees for the activity.

External Activity Occurrences

For a given complex activity, there may be external activities (that is, activities that are not subactivities) whose occurrence either interfere with the complex activity or which are necessary for the activity to occur. Examples of such necessary activities include either activities performed by external agents (such as a courier delivery or pickup) or it may be an activity such as setup. In the constraint *To produce the chassis, first drill the series of 1 cm holes, followed by drilling the series of 2 cm holes*, the activity that changes the drill bit fixture is not a subactivity of the process plan for producing the chassis, but is a setup activity that must occur between drilling the two sets of holes.

Interruptability

Closely related to external activity occurrences is the notion of interruptability and activity attempts. With an interruptable activity, an external activity may occur without interfering with the original activity. For example, interruptable activities may be preempted or suspended:

The assembly of computers for one customer can be halted to work on a rush order for another customer

$$\begin{aligned}(\forall s_1, x_1, x_2)\, root(s_1, assemble(x_1)) \wedge occurrence\_of(s_3, assemble(x_2)) \\ \wedge legal(s_3) \wedge earlier(s_1, s_3) \\ \supset (\exists s_2)\, leaf(s_2, assemble(x_1)) \wedge min\_precedes(s_1, s_2, assemble(x_1))\end{aligned} \quad (20)$$

while noninterruptable activities may not:

Pouring of metal from the furnace cannot be stopped once initiated.

$$\begin{aligned}(\forall s_1, s_2)\, root(s_1, pour\_metal) \wedge leaf(s_2, pour\_metal) \\ \wedge min\_precedes(s_1, s_2, pour\_metal) \\ \supset \neg(\exists s_3)\, occurrence\_of(s_3, stop) \wedge earlier(s_1, s_3) \wedge earlier(s_3, s_2)\end{aligned} \quad (21)$$

In this latter example, if for some reason the metal pouring does stop, then we would intuitively consider this to be an activity attempt, rather than an occurrence of the activity.

Intended Effects

There are many circumstances in which we want to make a distinction between the intended effects of an activity and the actual effects of the activity. For example, the manufacturing process plan for making some product in a steel company is defined with respect to the properties specified by customer and quality requirements (such as grade, surface properties, width, and thickness), but due to external nondeterministic factors, not every occurrence of the process will provide products that satisfy these requirements. Quality problems arise from this divergence of actual effects from intended effects.

For example, informal process descriptions such as *Bake the soup until it is opaque* or *Heat the solution until reaches 50 C* can be formalized by sentences of the form

$$(\forall s)\, leaf(s, a) \supset holds(f, s) \tag{22}$$

In both of these examples, it is possible to terminate the activity occurrence before the intended state is achieved, but in the context of the intended effects, the activity occurrence will terminate only when the state is achieved.

Temporal Constraints

With temporal constraints, subactivities are not allowed to occur at arbitrary times during occurrences of the activity. Examples of such constraints include schedules, which specify the possible times at which the subactivities may occur:

The part will arrive 10 days after placing the order request

$$(\forall o, s_1, s_2)\, min\_precedes(s_1, s_2, a) \wedge occurrence\_of(s_1, a_1) \wedge occurrence\_of(s_2, a_2)$$
$$\supset duration(endof(s_2), endof(s_1)) = 10 \tag{23}$$

In this example, the possible occurrences of the activity are restricted to those whose subactivities satisfy the temporal constraints.

6 Summary

Within the increasingly complex environments of enterprise integration, electronic commerce, and the Semantic Web, where process models are maintained in different software applications, standards for the exchange of this information must address not only the syntax but also the semantics of process concepts. PSL draws upon well-known mathematical tools and techniques to provide a robust semantic foundation for the representation of process information. This foundation includes first-order theories for concepts together with complete characterizations of the soundness and completeness of these theories. In this chapter, we have seen how the PSL Ontology can be used to specify process descriptions for a broad range of problems and provide the semantic foundations for new ontologies.

References

1. Grenon, P. and Smith, B. (2004) SNAP and SPAN: Towards dynamic spatial ontology. *Spatial Cognition and Computation*, 4(1):69-104, 2004.
2. Gruninger, M. (2003) Applications of PSL to Semantic Web Services, *Workshop on Semantic Web and Databases*. Very Large Databases Conference, Berlin.
3. Gruninger, M. and Kopena, J. (2004) Semantic Integration through Invariants, *AI Magazine*, 26:11-20, 2004.
4. Hayes, P. (1996) *A Catalog of Temporal Theories*. Artificial Intelligence Technical Report UIUC-BI-AI-96-01, University of Illinois at Urbana-Champaign.
5. McIlraith, S., Son, T.C. and Zeng, H. (2001) Semantic Web Services, *IEEE Intelligent Systems*, Special Issue on the Semantic Web. 16:46–53, March/April, 2001.
6. Menzel, C. and Gruninger, M. (2001) A formal foundation for process modeling, *Second International Conference on Formal Ontologies in Information Systems*, Welty and Smith (eds), 256-269.
7. Ciocoiu, M., Gruninger M., and Nau, D. (2001) Ontologies for integrating engineering applications, *Journal of Computing and Information Science in Engineering*, 1:45-60.
8. Pinto, J. and Reiter, R. (1993) Temporal reasoning in logic programming: A case for the situation calculus. *Proceedings of the 10th International Conference on Logic Programming*, Budapest, Hungary, June 1993.
9. Schlenoff, C., Gruninger, M., Ciocoiu, M., (1999) The Essence of the Process Specification Language, *Transactions of the Society for Computer Simulation* vol.16 no.4 (December 1999) pages 204-216.
10. Sowa, J. (2000) *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks/Cole Publishing.
11. Semantic Web Services Framework (SWSF) Overview W3C Member Submission 9 September 2005.