Schlenoff, C., Gruninger, M., Lee, J., (2000), "The Essence of the Process Specification Language", Transaction of the Society for Computer Simulation International, Vol. 16, No. 4.

The Essence of the Process Specification Language

Craig Schlenoff
National Institute of Standards and Technology
Bldg. 220, Rm. A127
Gaithersburg, MD 20899

Michael Gruninger
Department of Mechanical and Industrial Engineering
4 Taddle Creek Road
University of Toronto
Toronto, Ontario M5S 3G9

Mihai Ciocoiu
Institute of Systems Research
University of Maryland
College Park, MD 20742

Jintae Lee
Information Systems
Campus Box 419
University of Colorado
Boulder, Colorado 80309-0419

Keywords: manufacturing process specification, PSL, interoperability, ontology

In all types of communication, the ability to share information is often hindered because the meaning of that information can be affected drastically by the context in which it is viewed and interpreted. This is especially true among manufacturing simulation systems because of the growing complexity of manufacturing information and the increasing need to exchange this information not only among different simulation systems but also between simulation systems and systems that perform different functions (e.g., process planning, scheduling, etc.). Different manufacturing functions may use different terms to mean the exact same concept or use the exact same term to mean very different concepts. Often, the loosely defined natural language definitions associated with the terms contain much ambiguity that doesn't make these differences evident and/or do not provide enough information to resolve the differences.

A solution to this problem is the development of a taxonomy, or ontology, of manufacturing concepts and terms along with their respective formal and unambiguous definitions. The Process Specification Language (PSL) (Version 1.0) developed at the National Institute of Standards and Technology identifies, formally defines, and structures the semantic concepts intrinsic to the capture and exchange of discrete manufacturing process information.

1.0. Background and Overview

1.1. Purpose

As the use of information technology in manufacturing operations has matured, the capability of software applications to interoperate has become increasingly important. Initially, translation programs were written to enable communication from one specific application to another, although not necessarily both ways. As the number of applications has increased and the information has become more complex, it has become much more difficult for software developers to provide translators between every pair of applications that need to exchange information. Standards-based translation mechanisms have simplified integration for some manufacturing software developers by requiring only a single translator to be developed between their respective software product and the interchange standard. By only developing this single translator, the application can interoperate with a wide variety of other applications that have a similar translator between that standard and their application.

This challenge of interoperability is especially apparent with respect to manufacturing process information. Many manufacturing engineering and business software applications use process information, including manufacturing simulation, production scheduling, manufacturing process planning, workflow, business process reengineering, product realization process modeling, and project management. Each of these applications utilizes process information in a different way, so it is not surprising that these applications' representations of process information are different as well. Traditional approaches to integrating these packages rely on point-to-point translators,

which force the existence of n^2 translators for every n applications to be integrated. A standard process exchange language can reduce this n^2 number to 2n. The primary difficulty with developing a standard to exchange process information is that these applications sometimes associate different meanings with the terms representing the information that they are exchanging. For example, in the case of a workflow system, a resource is primarily thought of as the information that is used to make necessary decisions. In a simulation system, a resource is primarily thought of as a person or machine that will perform a given task. If one were to integrate a process model from a workflow with a simulation application, one's first inclination would most likely be to map one resource concept to the other. This mapping would undoubtedly cause confusion. Therefore, both the semantics and the syntax of these applications need to be considered when translating to a neutral standard. In this case, the standard must be able to capture all of the potential meanings behind the information being exchanged.

The Process Specification Language (PSL) project at the National Institute of Standards and Technology (NIST) is addressing this issue by creating a neutral, standard language for process specification to serve as an interlingua to integrate multiple process-related applications throughout the manufacturing life cycle. This interchange language is unique due to the formal semantic definitions (the ontology) that underlie the language. Because of these explicit and unambiguous definitions, information exchange can be achieved without relying on hidden assumptions or subjective mappings.

1.2. Approach

The plan for the PSL project has five phases: requirements gathering, existing process representation analysis, language creation, pilot implementation and validation, and

submission as a candidate standard. The completion of the first phase resulted in a comprehensive set of requirements for specifying manufacturing processes [1]. In the second phase, twenty-six process representations were identified as candidates for analysis by the PSL team and analyzed with respect to the phase one requirements [2]. Nearly all of the representations studied focused on the syntax of process specification rather than the meaning of terms, the semantics. While this is sufficient for exchanging information between applications of the same type, such as process planning, different types of applications associate different meanings with similar or identical terms. As a result of this, a large focus of the third phase involved the development of a formal semantic layer (an ontology) for PSL based on the Knowledge Interchange Format (KIF) specification [3]. By using this ontology to define explicitly and clearly the concepts intrinsic to manufacturing process information, PSL was used to integrate multiple manufacturing process applications in the fourth phase of the project.

1.3. Scope

To keep this work feasible, the scope of study is limited to the realm of discrete processes related to manufacturing, including all processes in the design/manufacturing life cycle. Business processes and manufacturing engineering processes are included in this work both to ascertain common aspects for process specification and to acknowledge the current and future integration of business and engineering functions.

In addition, the goal of this project is to create a "process specification language", not a "process characterization model". Our definition of a *process specification language* is a language with which to specify a process or a flow of processes, including supporting parameters and settings and establishing varying contexts. This may be done for

prescriptive or descriptive purposes and is composed of an ontology and one or more presentations. This is different from *a process characterization model*, which we define as a model describing the behaviors and capabilities of a process independent of any specific application. For example, the dynamic or kinematic properties of a process (e.g., tool chatter, a numerical model capturing the dynamic behavior of a process or limits on the process' performance or applicability) independent of a specific process would be included in this characterization model.

2.0. Related Work

The PSL project at NIST is creating a neutral, standard language for process specification to serve as an neutral interchange language to integrate multiple process-related applications throughout the manufacturing life cycle. This project is related to, and in many cases working closely with, many other efforts. These include individual efforts (those involving only a single company or academic institution) such as A Language for Process Specification (ALPS) Project [4], the Toronto Virtual Enterprise (TOVE) Project [5], the Enterprise Ontology Project [6], and the Core Plan Representation (CPR) Project [7]. In addition, the PSL project is in close collaboration with various projects (those which involve numerous companies or academic institutions) such as Shared Planning and Activity Representation (SPAR) Project [8], the Process Interchange Format (PIF) Project [9], and the WorkFlow Management Coalition (WfMC) [10].

ALPS was a NIST research project whose goal was to identify information models to facilitate process specification and to transfer this information to process control. The PSL project, which could be viewed as a spin-off of the ALPS project, has a goal to take

a much deeper look into the issues of process specification and to explore these issues in a much broader set of manufacturing domains.

The TOVE project provides a generic, reusable data model that provides a shared terminology for the business and enterprise. The Enterprise Ontology project's goal is to provide "a collection of terms and definitions relevant to business enterprises to enable coping with a fast changing environment through improved business planning, greater flexibility, more effective communication and integration". While both TOVE and the Enterprise Ontology focus on business processes, there are common semantic concepts in both these projects and the manufacturing process-focused PSL.

The CPR project is attempting to develop a model that supports the representation needs of many different military planning systems. The SPAR project is an ARPI (ARPA (Advanced Research Projects Agency)/Rome Laboratory Planning Initiative) funded project whose goals are similar to CPR. Both of these projects are similar to PSL in the sense that they are attempting to create a shared model of what constitutes a plan, process, or activity. However, both SPAR and CPR are focusing more on military types of plans and processes.

PIF is an interchange format based upon formally defined semantic concepts, like PSL. However, unlike PSL, PIF is focused on modeling business processes and offers a single, syntactical presentation, the BNF (Backus-Naur Format) specification of the Ontolingua[1] Frame syntax.

The Workflow Management Coalition has developed a Workflow Reference Model whose purpose is to identify the characteristics, terminology, and components to enable the development and interoperability of workflow specifications. Although the area of

workflow is within the scope of the PSL project, it is only one small component. The Workflow Reference Model has and will be used by the PSL project to ensure consistency.

In addition to the existing projects described above, there have been countless, previous efforts to create process representations focusing specifically on various representational areas or on different functionality. For example, representational areas such as workflow, process planning, and business process re-engineering have had representations developed focusing solely on their respective areas. Equally important to the representational area in which the representations are being developed is the role (functionality) that the representation will play. There have been process representations developed which have focused on simply graphically documenting a process, to those which are used as internal representations for software packages, to those which are used as a neutral representation to enable integration. The process representations that resulted from many of these efforts were analyzed in the second phase of the PSL project (described above). A sampling of some of these existing process representations is shown in Figure 1. For more information about the representations listed in the figure, please see [2].

3.0. The Process Specification Language

3.1. The Need for Semantics

Existing approaches to process modelling lack an adequate specification of the semantics of the process terminology, which leads to inconsistent interpretations and uses of the information. Analysis is hindered because models tend to be unique to their applications

and are rarely reused. Obstacles to interoperability arise from the fact that the legacy systems that support the functions in many enterprises were created independently, and do not share the same semantics for the terminology of their process models.

For example, consider Figure 2 in which two existing process planning applications are attempting to exchange data. Intuitively the applications can share concepts; for example, both *material* in Application A and *workpiece* in Application B correspond to a common concept of *work-in-progress*. However, without explicit definitions for the terms, it is difficult to see how concepts in each application correspond to each other. Both Application A and B have the term *resource*, but in each application this term has a different meaning. Simply sharing terminology is insufficient to support interoperability -- the applications must share their semantics.

A rigorous foundation for process design, analysis, and execution therefore requires a formal specification of the semantics of process models. One approach to generating this specification is through the use of ontologies. An ontology is a formal description of entities and their properties, relationships, constraints, and behaviors [11]. It provides a common terminology that captures key distinctions and is generic across many domains, facilitating translation of concepts among these domains.

Translation itself places constraints on the specification of process model semantics. Applications interoperate by translating between their native format and PSL. One approach is to design unique translators that must be written for every two-party exchange; however, this would require $O(n^2)$ translators for n different ontologies. A major goal of PSL is to reduce the number of translators to $O(n)$ for n different ontologies, since it would only require translators from a native ontology into the

interchange ontology. The other feature of this approach is that the applications primarily interact through the exchange of files that contain process information. This requires the declarative specification of semantics -- there can be no procedural interpretation of the application constructs, since all we have is the input file. Similarly, all assumptions made by the application must be made explicit since translation must be done using the input file alone.

3.2. What is PSL?

Within our work, the term "ontology" refers to a set of sentences in first-order logic, comprising of a set of foundational theories and sets of definitions written using the foundational theories. In providing such an ontology, we must specify three notions:

- language
- model theory
- proof theory (axioms and definitions)

A language is a set of symbols (lexicon) and a specification of how these symbols can be combined to make well-formed formulae (grammar/syntax). The lexicon consists of logical symbols (such as connectives, variables, and quantifiers) and nonlogical symbols. For PSL, the nonlogical part of the lexicon consists of expressions (constants, function symbols, and predicates) that refer to everything needed to describe processes.

The underlying language used for PSL is KIF (Knowledge Interchange Format). Briefly stated, KIF is a formal language developed for the exchange of knowledge among disparate computer programs. KIF provides the level of rigor necessary to define concepts in the ontology unambiguously, a necessary characteristic to exchange manufacturing process information using the PSL Ontology.

The primary component of PSL is its terminology for classes of processes and relations for processes and resources, along with definitions of these classes and relations. Such a lexicon of terminology along with some specification of the meaning of terms in the lexicon constitutes what is known as an ontology. In our case, this will be the PSL ontology for processes.

The model theory of PSL provides a rigorous mathematical characterization of the semantics of the terminology of PSL. The objective is to identify each term with an element of some mathematical structure, such as a set or a set with additional structure (e.g. a complete partial order); the underlying theory of the mathematical structure then becomes available as a basis for reasoning about the terms of the language and their relationships.

The proof theory of PSL provides axioms for the interpretation of terms in the ontology. It is useful to distinguish two types of sentences in this set of axioms: core theories and definitions. A core theory is a set of distinguished predicates, function symbols, and individual constants, together with some axiomatization. Distinguished predicates are those for which there are no definitions; the intended interpretations of these predicates are defined using the axioms in the core theories. For these terms, we need to describe the set of models corresponding to the intuitions that we have for them. We then write axioms that are sound and complete with respect to the set of models. That is, every interpretation that is consistent with the axioms is a model in the set, and any model in the set is an interpretation consistent with the axioms. These axioms constitute the foundational theories of the ontology. The set of models form the semantics (or model theory) of the ontology.

All other terms in the ontology are given definitions using the set of primitive terms. These definitions are known as conservative definitions since they do not add to the expressive power of the core theories, that is, anything that we can deduce with the definitions, we can deduce using the core theories alone. All definitions in an ontology are specified using the core theories; any terminology that does not have a definition is axiomatized in some core theory. Since all other terms are defined using these primitives, the set of models for them can be defined using the models of the core theories for the primitives. We can therefore give a semantics to the definitions using the classes of models that have already been specified for the core theories.

The challenge is that we need some framework for making explicit the meaning of the terminology for many ontologies that reside only in people's heads. Any ideas that are implicit are a possible source of ambiguity and confusion. For PSL, the model theory provides a rigorous mathematical characterization of process information and the axioms give precise expression to the basic logical properties of that information in the PSL language. So when we speak about a semantics for PSL, we are referring to the axiomatization of core theories and definitions for the PSL terminology.

The focus of the ontology is not only on the terms, but also on their definitions. We can include an infinite set of terms in our ontology, but they can only be shared if we agree on their definitions. It is the definitions that are being shared, *not simply* the terms.

3.3. Semantic Architecture

The PSL Ontology consists of axioms and definitions for the lexicon of PSL. However, this is not simply an amorphous set of sentences. Figure 3 gives an overview of the semantic architecture of the PSL Ontology. There are three major components of the

architecture -- the axioms of PSL-Core, core theories, and definitions that are organized as sets of extensions to PSL-Core.

PSL-Core is used to specify the semantics of the primitives in the PSL Ontology corresponding to the fundamental intuitions about activities. Recall that primitives are those terms for which we do not give definitions; rather, we specify sentences which constrain the interpretation of the terms.

The terms that have definitions can be grouped into modules, each of which is an extension of PSL-Core. The modules are organized by logical dependencies -- one module depends on another if the definitions of the terminology of the first module require the lexicon of the second module. PSL-Core is therefore intended to be used as the basis for defining terminology of the extensions in the PSL Ontology.

The first version ofPSL illustrates some of this organization. Figure 4 illustrates the modules required to define the terminology of the simplified process planning domain. There are four extensions to PSL-Core: Ordering Relations, Resource Roles, Processor Actions, and Resource Paths. The arcs in the diagram illustrate direct logical dependencies -- if there is an arc from one module to another, then there exists a term in the second module which uses a term defined in the first module. Thus, the definition of the ordering relations and resource roles depends only on PSL-Core. The definition of processor actions uses resource roles, but no ordering relations. Finally, resource paths are partially ordered sets of processor actions through which material resources flow; hence, the definition depends on both ordering relations and processor actions.

In addition to PSL-Core and its extensions, other sets of axioms may be required to introduce new primitive concepts; these axioms are grouped into core theories. The

extensions that introduce new primitive concepts do so because the concepts that are introduced in the PSL Core are not sufficient for defining the terms introduced in the extensions. Therefore, new primitive concepts are introduced within the extensions to ensure that all other terms within the extension can be defined completely.

One of the most important such core theories within PSL 1.0 is situation calculus [12]. This theory is powerful enough to prove theorems about PSL-Core and its extensions, such as theorems to characterize the completeness of the set of resource roles, and similar theorems to characterize the structure of partially ordered actions. It is also strong enough for building formal semantic models and proving the soundness of proposed semantic translation schemes. Finally, and perhaps most important, in situation calculus, one is able to give precise definitions and axiomatizations of many notions that are usually left as primitives or are at best inadequately axiomatized, thus greatly enhancing the precision of semantic translations between different schemes.

Other core theories include Kripke-Platek Set Theory [13], which provides a simple set of axioms for reasoning about sets, Resource Requirements Theory [13], which provides axioms for the relation between activities and resources, and Duration Theory [13], which provides axioms for the concept of duration, clocks, and their relationship to timepoints.

3.4. PSL-Core

PSL-Core is based upon a precise, mathematical, first-order theory, i.e., a formal language, a precise mathematical semantics for the language, and a set of axioms that express the semantics in the language. Here we will provide a brief informal sketch of the semantics and give the basic axioms for that semantics. There are four basic classes

and four basic relations in the ontology of PSL-Core. The classes are Object, Activity, Activity Occurrence, and Timepoint. The four relations are Participates-in, Before, BeginOf, and Endof. Activities, Activity Occurrences, Timepoints (or "points", for short), and objects are known collectively as entities, or things. These entities are all pairwise disjoint.

Intuitively, an Object is a concrete or abstract thing that can participate in an Activity. The most typical examples of objects are ordinary tangible things - like people, chairs, car bodies, NC-machines, and the like - though abstract objects like numbers are not excluded. Objects can come into existence (e.g., be created) and go out of existence (e.g., be "used up" as a resource) at certain points in time. In such cases, an Object has a begin and/or end point. Some Objects, e.g., numbers, do not have finite begin and end points. In some contexts, it may be useful to model certain ordinary objects as having no such points either.

An Activity Occurrence is a limited, temporally extended piece of the world, such as the first mountain stage of the 1997 Tour de France or the eruption of Mt. St. Helen. Any Activity Occurrence is simply taken to be characterized chiefly by two things: its temporal extent, as determined by it's begin and end points (possibly at infinity), and the set of Objects that participate in that activity at some point between its begin and end points.

Timepoints are assumed to be ordered by the Before relation. This relation is taken to be a transitive, irreflexive, total ordering. It is not assumed in PSL-Core that time is dense (i.e., between any two distinct timepoints there is a third), though it is assumed that time is infinite. Points at infinity are assumed for convenience. (Denseness, of course,

could easily be added by a user as an additional postulate.) Time intervals are not included among the primitives of PSL-Core, because they can be defined with respect to Timepoints and Activities. Time durations are included in an extension of the PSL-Core that builds upon [14].

The basic notions of the PSL-Core are axiomatized formally as a first-order theory. These axioms simply capture, in a precise way, the basic properties of the PSL ontology. The basic axioms for Activities, Objects, and Timepoints are listed below. The following definitions simplify the axioms.

3.4.1. Supporting Definitions for PSL-Core

Definition 1. Timepoint q is between timepoints p and r if and only if p is before q and q is before r.

```
(defrelation between (?p ?q ?r) :=
        (and (before ?p ?q) (before ?q ?r)))
```

Definition 2. Timepoint p is beforeEq timepoint q if and only if p is before or equal to q.

```
(defrelation beforeEq (?p ?q) :=
        (and (timepoint ?p)  (timepoint ?q)
                (or  (before ?p ?q)
                     (= ?p ?q))))
```

Definition 3. Timepoint q is betweenEq timepoints p and r if and only if p is before or equal to q, and q is before or equal to r.

```
(defrelation betweenEq (?p ?q ?r) :=
        (and  (beforeEq ?p ?q)
              (beforeEq ?q ?r)))
```

Definition 4. An object exists-at a timepoint p if and only if p is betweenEq its begin and end points.

```
(defrelation exists-at (?x ?p) :=
        (and      (object ?x)
                  (betweenEq (beginof ?x) ?p (endof ?x))))
```

Definition 5. An activity is-occurring-at a timepoint p if and only if p is betweenEq the activity's begin and end points.

```
(defrelation is-occurring-at (?a ?p) :=
```

```
(exists (?occ)
(and    (occurrence ?occ ?a)
        (betweenEq (beginof ?occ) ?p (endof ?occ)))))
```

3.4.2. PSL-Core Axioms

Axiom 1. The before relation only holds between timepoints.

```
(forall (?p ?q)
        (=>     (before ?p ?q)
                (and    (timepoint ?p)
                        (timepoint ?q))))
```

Axiom 2. The before relation is a total ordering.

```
(forall (?p ?q)
        (=>     (and    (timepoint ?p)
                        (timepoint ?q))
        (or     (= ?p ?q)
                        (before ?p ?q)
                        (before ?q ?p))))
```

Axiom 3. The before relation is irreflexive.

```
(forall (?p)
        (not (before ?p ?p)))
```

Axiom 4. The before relation is transitive.

```
(forall (?p ?q ?r)
        (=>     (and    (before ?p ?q)
                        (before ?q ?r))
                (before ?p ?r)))
```

Axiom 5: The timepoint inf- is before all other timepoints.

```
(forall (?t)
        (=>     (and    (timepoint ?t)
                        (not (= ?t inf-)))
                (before inf- ?t))
```

Axiom 6. Every other timepoint is before inf+.

```
(forall (?t)
        (=>     (and     (timepoint ?t)
                        (not (= ?t inf+)))
                 (before ?t inf+))
```

Axiom 7. Given any timepoint t other than inf-, there is a timepoint between inf- and t.

```
(forall (?t)
        (=>      (and      (timepoint ?t)
                           (not (= ?t inf-)))
                 (exists (?u)
                           (between inf- ?u ?t))))
```

Axiom 8. Given any timepoint t other than inf+, there is a timepoint between t and inf+.

```
(forall (?t)
        (=>      (and      (timepoint ?t)
                           (not (= ?t inf+)))
                 (exists (?u)
                           (between ?t ?u inf+))))
```

Axiom 9. Everything is either an activity, object, or timepoint.

```
(forall (?x)
        (or      (activity ?x)
                 (activity-occurrence ?x)
                 (object ?x)
                 (timepoint ?x)))
```

Axiom 10. Objects, activities, activity occurrences, and timepoints are all distinct kinds of things.

```
(forall (?x)
        (and     (=>       (activity ?x)
                 (not (or  (activity-occurrence ?x)
                           (object ?x)
                           (timepoint ?x))))
                 (=>       (activity-occurrence ?x)
                           (not (or  (object ?x)
                           (timepoint ?x))))
                 (=>       (object ?x)
                           (not (timepoint ?x)))))
```

Axiom 11. The occurrence relation only holds between activities and activity occurrences.

```
(forall (?a ?occ)
        (=>      (occurrence ?occ ?a)
                 (and      (activity ?a)
                           (activity-occurrence ?occ))))
```

Axiom 12. An activity occurrence is associated with a unique activity.

```
(forall (?occ ?a1 ?a2)
```

```
(=>    (and   (occurrence ?occ ?a1)
              (occurrence ?occ ?a2))
       (= ?a1 ?a2))))
```

Axiom 13. The begin and end of an activity occurrence or object are timepoints.

```
(forall (?a ?x)
        (=>      (or      (occurrence ?x ?a)
                          (object ?x))
                 (and     (timepoint (beginof ?x))
                          (timepoint (endof ?x)))))
```

Axiom 14. The begin point of every activity occurrence or object is before or equal to its end point.

```
(forall (?a ?x)
        (=>      (or      (occurrence ?x ?a)
                          (object ?x))
                 (beforeEq (beginof ?x) (endof ?x))))
```

Axiom 15. The participates-in relation only holds between objects, activities, and timepoints, respectively.

```
(forall (?x ?a ?t)
        (=>      (participates-in ?x ?a ?t)
                 (and     (object ?x)
                          (activity ?a)
                          (timepoint ?t)))))
```

Axiom 16. An object can participate in an activity only at those timepoints at which both the object exists and the activity is occurring.

```
(forall (?x ?a ?t)
        (=>      (participates-in ?x ?a ?t)
                 (and     (exists-at ?x ?t)
                          (is-occurring-at ?a ?t))))
```

3.5. Extensions in PSL 1.0

The set of extensions in PSL 1.0 fall roughly into three "families":

- Generic Activities and Ordering Relations
- Process Planning
- Resources and Schedules

3.5.1. PSL Extensions for Generic Activities and Ordering Relations

Figure 5 illustrates the modules in PSL that are required to define the terminology for generic classes of activities and their ordering relations. There are ten relevant extensions to PSL-Core, five dealing with generic process modeling concepts and five dealing with resources and schedules. The five focusing on resources and schedules will be discussed in Section 3.5.3. The five dealing with generic process modeling concepts are:

- Ordering Relations
- Partially Ordered Activities
- Nondeterministic Activities
- Complex Sequence Ordering Relations
- Junctions

The first two of these extensions characterize deterministic activities -- all subactivities occur, and the (partial) ordering relations are defined over these occurrences. The final three extensions characterize nondeterministic activities in which not every subactivity occurs when the activity occurs; for example, to fabricate an engine block, one may either use the casting machine or modify an existing engine block. Junctions are a particular class of nondeterministic activities used to define notions such as splits and joins. Within a split, one of several activities may possibly occur next, whereas within a join, one of several activities must occur before the next activity occurs.

3.5.2. PSL Extensions for Process Planning

Figure 4 (in Section 3.3) illustrates the modules in PSL that are required to define the terminology of a simplified process planning domain. There are four relevant extensions:

- Ordering Relations

- Resource Roles
- Processor Actions
- Resource Paths

The arcs in Figure 5 illustrate direct logical dependencies -- if there is an arc from one module to another, then there exists a term in the second module that uses a term defined in the first module. Thus, the definition of the ordering relations and resource roles depends only on PSL-Core. The definition of processor actions uses resource roles, but not any ordering relations. Finally, resource paths are partially ordered sets of processor actions through which material resources flow; hence, the definition depends on both ordering relations and processor actions.

The Resource Roles extension axiomatizes the fundamental intuitions about resources. An Object is a resource only with respect to the role that it plays in some activity that requires the Object. We are therefore not axiomatizing any other properties of resources. Rather, resource roles are one way of formalizing the way in which an activity requires the resource.

In particular, resources are the means for reasoning about Activity interaction. This is a common intuition in the artificial intelligence (AI) planning community, where tasks are frequently described in terms of the resources that are required. This appears to be a useful concept for reasoning about how different activities interact. The first step in the axiomatization of resource roles is to define activity interactions with respect to the preconditions and effects of activities. In particular, the resource role definitions in PSL 1.0 apply to a class of activity interactions in which two activities are interfering with one another in that the occurrence of both activities is not possible.

The class of processor actions is defined with respect to the resource roles. These are activities that use or consume some set of resources, and produce a set of objects. Intuitively, reusable resources are machines and the consumable resources are materials.

Finally, the extension for resource paths defines partially ordered sets of processor actions, where the object produced by one activity is the input object for the next activity.

3.5.3. PSL Extensions for Resources and Schedules

These extensions were motivated by the applications in the PSL pilot implementation, in particular ILOG Scheduler 4.3. At the beginning of the pilot implementation of PSL, there were no extensions capable of defining concepts such as temporal or resource constraints completely. It was necessary therefore to design new extensions containing terminology whose definitions correctly and completely captured the intuitive meaning of the ILOG Schedule concepts.

Scheduling can be characterized intuitively as the assignment of resources to activities such that both resource constraints and temporal constraints are satisfied. Resource constraints include notions such as resource sharing (capacity) and the states of resources, such as availability. Temporal constraints include the duration of activities and the temporal ordering of activity occurrences. These intuitions lead to the introduction of five extensions within PSL 1.0, shown in Figure 5:

- Durations
- Activities and Duration
- Temporal Ordering Relations
- Reasoning about State

- Interval Activities

The biggest hurdle in the development of these extensions is the axiomatization of discrete capacity resources. The major problem in this case is that the discreteness of the resource arises from the fact that it is actually composed of a set of resources, and any activity requires or provides some subset of resources in this set. Within the PSL Ontology, this led to the introduction of the following extensions, presented in order of increasing specialization (shown in Figure 6):

- Kripke-Platek Set Theory, which defines the basic notion of a set of objects
- Resource Sets, which defines the class of sets of resources which themselves behave as resources
- Resource Set-based Activities, which defines classes of activities which use resource sets
- Substitutable Resources, which makes the distinction between sets of arbitrary resources and sets of resources that can be substituted for others in an activity (e.g. the set of carpenters in a house construction activity)
- Resource Pools, which are equivalent to discrete capacity resources within ILOG Schedule;
- Inventory Resource Sets, which are equivalent to reservoirs within ILOG Schedule.

3.6. Approach for Developing Extensions

From the above list of extensions, it is easy to see that certain representational areas within PSL have been thoroughly worked out and some have not been addressed yet. For example, the area relating to "ordering of activities" has been well addressed within the extensions of "Ordering Relations for Complex Sequence Actions", "Ordering Relations

over Activities", and "Temporal Ordering". However, other representational areas such as "Process Intent" have not yet been addressed.

The initial PSL ontology was developed using a single scenario, the EDAPS (Electromechanical Design and Planning System) scenario developed by Steve Smith at the University of Maryland [15]. The concepts introduced in that scenario were defined and modeled within PSL and later extended as other scenarios were explored. The PSL ontology was then further expanded to incorporate the concepts introduced in various manufacturing software applications when PSL was used to exchange process information among these packages. As more software applications become "PSL-compliant", PSL will be continuously expanded to ensure that ALL process-related concepts are capable of being represented within the language.

4.0. Translation Using PSL

4.1. Motivation

To guarantee correct and complete translation, translators must be based on the *formal* specifications of the representation's semantics. Translators written "by hand" provide no such guarantee, and proving that they actually perform the intended "correct" translation is so difficult that it is almost never done.

4.2. Overview of Semantic and Syntactic Translation

We consider translation to be a two-stage process -- syntactic translation and semantic translation. The syntactic translator is a parser between the PSL syntax (e.g. KIF) and the native syntax of one of the applications; this parser keeps the terminology of the

application intact. Figure 7 illustrates the translation transaction between two applications and the role played by the PSL Ontology.

Semantic translation substitutes terminology of one application with the definitions written using PSL terminology. These translation definitions between an application ontology and PSL are driven by the ontological definitions that were written using the same foundational theories. These are definitions for the terminology of the application ontology, using *only* the terminology from the PSL Ontology, as well as definitions for the terminology of the PSL Ontology using *only* the terminology of the application ontology.

This procedure is best shown by an example. The resource construct is highlighted during each stage of the example to show how it progresses through the translation process. We begin with a simple file written in Application A's syntax and using Application A's terminology:

```
{stock: wire (x)}
{stock: plug (x)}
{resource: inject_mold (x)}
{material: plug_head (x)}
{operation: fabricate_plug}
```

The syntactic translator takes this file and produces a corresponding file using PSL syntax, but still preserving Application A terminology.

```
(forall (?r)
        (=>     (wire ?r)
                (stock ?r)))

(forall (?r)
        (=>     (plug_head ?r)
                (material ?r)))
```

```
(forall (?r)
        (=>    (inject_mold ?r)
               (resource ?r)))

(forall (?r)
        (=>    (plug ?r)
               (stock ?r)))
```

The semantic translator takes this file and produces a file containing only PSL terminology by substituting the definitions of all Application A terms with their definitions in PSL.

```
(forall (?r)
        (=>    (wire ?r)
               (material ?r)))

(forall (?r)
        (=>    (plug_head ?r)
               (wip ?r)))

(forall (?r)
        (=>    (inject_mold ?r)
               (machine ?r)))

(forall (?r)
        (=>    (plug ?r)
               (material ?r)))
```

We now follow reversed steps to translate the file into Application B. Using the translation definitions for Application B, the PSL file is mapped to a file containing only Application B terminology.

```
forall (?r)
        (=>    (wire ?r)
               (resource ?r)))

(forall (?r)
        (=>    (plug_head ?r)
               (workpiece ?r)))
```

(forall (?r)
(=> (inject_mold ?r)
(machine-tool ?r)))

(forall (?r)
(=> (plug ?r)
(resource ?r)))

Finally, the syntactic translator for Application B maps the file back into Application B syntax.

(define-class wire
(Subclass-Of resource))

(define-class inject-mold
(Subclass-Of machine-tool))

(define-class plug
(Subclass-Of resource))

(define-class plug_head
(Subclass-Of workpiece))

(define-class fabricate_plug
(Subclass-Of task))

Note that this ontology-based approach to compliance is different than the traditional approach to standards compliance. Rather than forcing the adoption of exactly the same terminology, an application is PSL-compliant if there exist definitions for its terminology using either some foundational theory or other ontology. Given these definitions, translation definitions can be written between the application and PSL.

4.3. Semantic Translation: Practical Considerations

4.3.1. Problem Statement

Translating into PSL should be an easy task, due to the expressivity of KIF, the format on which PSL is based. Typically, it can be done by writing compilation rules that map each concept of a language with the corresponding KIF formula using PSL terminology. Writing such compilation rules can be seen as providing a declarative semantics for the language, and PSL can be thought of as a semantic description language. Once the semantic concepts of a source representation are clearly defined, a translator that mapped those concepts to the concepts within PSL could be written in less than one week.

The reverse process however, that is translating *from* PSL *to* a target language seems to be more difficult. We expect that the target languages will be much less expressive than KIF, and the same approach will not work, since there will be no way to write "definitions" in the target language for some of the constructs in PSL. Some form of re-construction will have to be done, that is, the construct in the target language will need to be built whenever its PSL semantic definition is satisfied. The problem is that there are infinitely many ways of expressing the same thing in KIF, and we want to build the target language construct whenever its KIF definition, or some logically equivalent form of it can be inferred. Therefore, a first-order theorem prover would need to be built in order to do the translation.

To better understand the difficulties involved, one can think of an analogy with translating programming languages. In this analogy, we want to build a translator between two high level (and thus less expressive) languages like C and Fortran. PSL is analogous to an assembly language. The problem of translating from a higher level (less

expressive) language into assembly language is solved relatively easy using compilers. The reverse problem however, namely, taking some assembly language and re-generating the high level code, is only solved when the compiler saved some information especially for this purpose. Generating high-level instructions of a *different* high level language has not been done.

There are two main challenges to be solved in order to translate out of PSL.

- one has to be able to write translation rules (or be able to infer them) for all concepts in PSL in order to build such a translator. (this task is made even harder by the extensibility requirement of PSL)
- a translator that does full power first-order theorem proving at translation time has to be *very* efficient.

4.3.2. An Approach for Semantic Translation

An approach to solving this problem is to develop a methodology for specifying compilation rules from arbitrary languages into PSL, that is, a method for writing semantic descriptions from a language constructs into PSL. This methodology should be both "user-friendly" (i.e. supporting a natural way of expressing a construct's semantics) and general (i.e. it should work for any language we might want to translate). In this vision, the semantics of an arbitrary language will be expressed by its semantic description, together with the (formally specified) semantics of PSL.

In order to exchange process information among arbitrary languages, all one needs to do is to provide the semantic description of the language.

This methodology will enable the automatic inference of translation rules among arbitrary languages based only on their semantic descriptions into PSL. Such a methodology would be useful since:

- It will solve the difficult problem of translating out of PSL.
- It will *guarantee* the correctness of the generated translators with respect to the semantic descriptions of the languages involved in the translation.
- It will facilitate the translation specification, since all that will need to be written for an application in order to be able to exchange information will be the semantic definitions.
- It will lead to very efficient translators, since the inference procedure will be run only once for any two languages and only on the semantic definitions.
- It will allow PSL to be extensible, without requiring one to rewrite the "out of PSL" translators each time a PSL extension is created.
- It will address the challenge of partial translation (a way of approximating concepts that are similar enough to be translated to one another but have slightly different semantic descriptions or are of a different granularity).

5.0. Conclusion

The purpose of the Process Specification Language is to provide a representation for manufacturing process information that will serve as an interlingua to facilitate the exchange of information among manufacturing software applications. This paper discusses the components of Version 1.0 of PSL.

Other efforts to develop mechanisms for the exchange of data, such as the Standard for the Exchange of Product model data (STEP) [16], have focused on syntactical

standards elements necessary for data exchange. This focus works well for exchanging information among similar domains where the terms used have the same meanings. However, within the increasingly complex manufacturing environment 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 meanings or semantics of terms and concepts used. PSL uniquely addresses this in its identification and development of semantics for specifying and exchanging process information. The identification of the necessary concepts was based on a thorough analysis of the requirements for specifying business and manufacturing engineering processes in the manufacturing domain and then analyzing a broad set of existing approaches to representing process models with respect to these requirements.

Version 1.0 of PSL represents the beginning point in the development of a robust and complete Process Specification Language. This initial version will be refined in an iterative fashion to continuously increase the robustness of the language. A series of pilot implementations, in which PSL will be used to exchange process information between existing manufacturing applications, will allow us to determine which representational areas need to be expanded upon to ensure that the PSL will be able to capture and exchange all current and future manufacturing process information. Early findings indicate that areas such as process intent and feature information are needed and currently lacking. These areas will be the immediate focus for the next release of PSL.

- ACT
- A Language for Process Specification (ALPS)
- AND/OR Graphs
- Behavior Diagrams
- Core Plan Representation (CPR)
- Data Flow Diagrams
- Directed Graphs
- Entity-Relationship (E-R) models
- EPFL Petri Net Representation
- Functional Flow Block Diagrams
- Gantt Charts
- Generalized Activity Networks (GAN)
- Hierarchical Task Networks (HTN)
- IDEF0
- IDEF3
- <I-N-OVA> Constraint Model
- Knowledge Interchange Format (KIF)
- O-Plan Formalism
- OZONE
- Parts and Action (Pact)
- PAR2
- STEP's Part 49
- PERT Networks
- Petri Nets
- Process Flow Representation (PFR)
- Process Interchange Format (PIF) Version 1.1
- Quirk Models
- State Transition Diagrams
- Tree Structures
- Visual Process Modeling Language (VPML)

Figure 1: A Sampling of Existing Process Representations

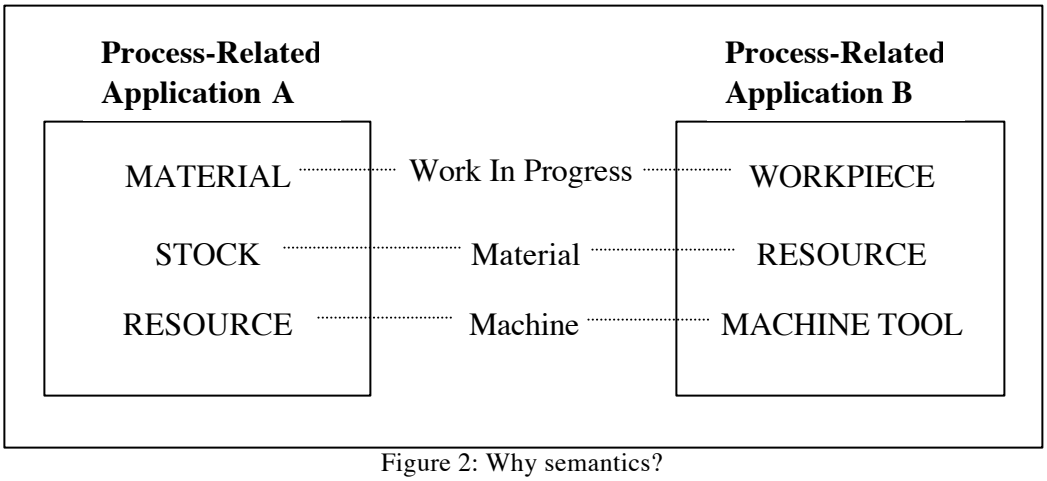

Figure 2: Why semantics?

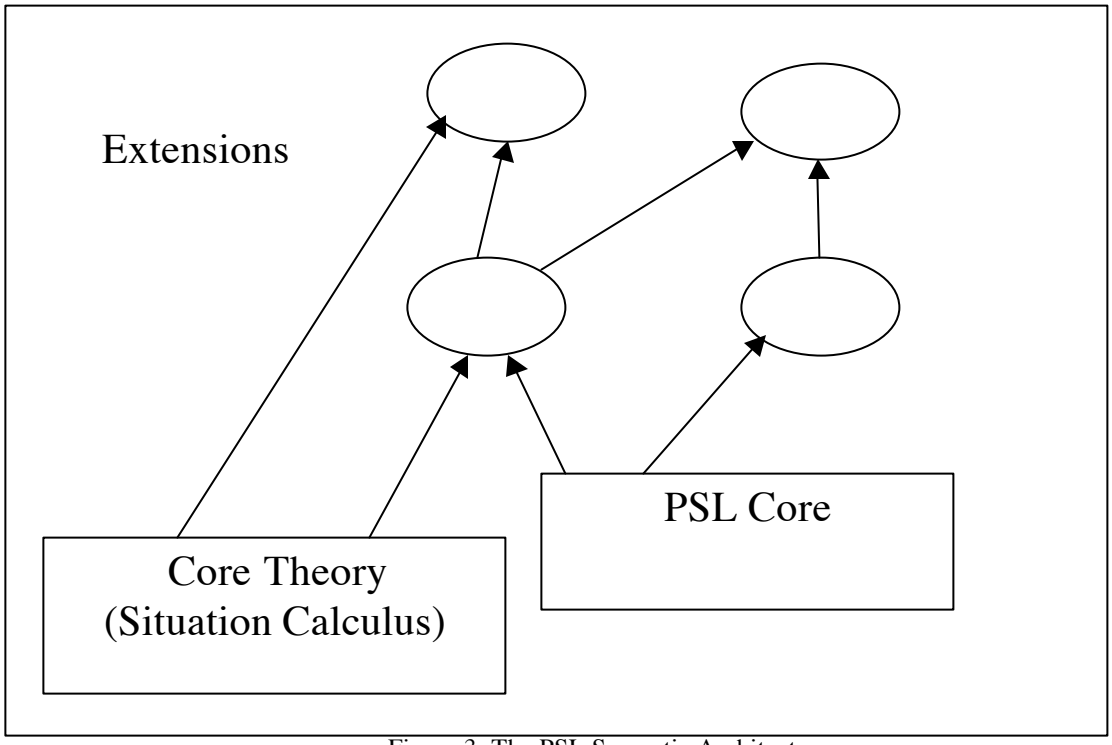

Figure 3: The PSL Semantic Architecture

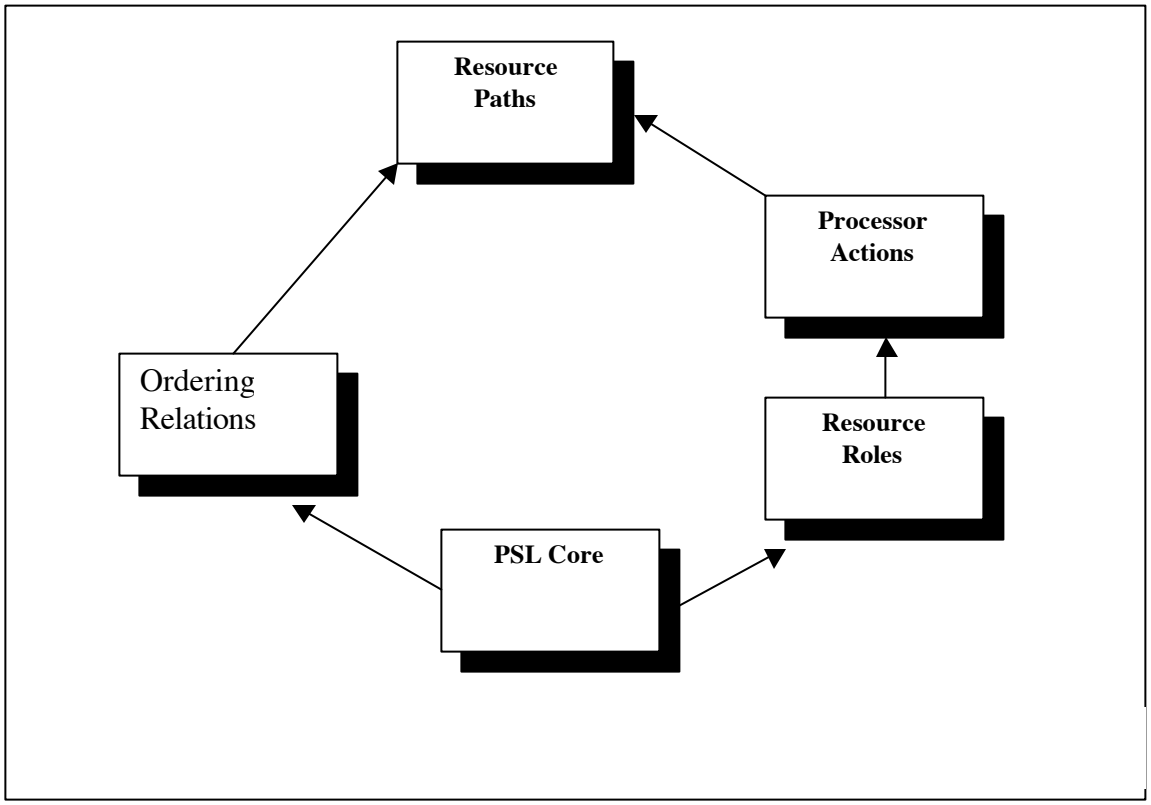

Figure 4: Modules Required for the Process Planning Domain

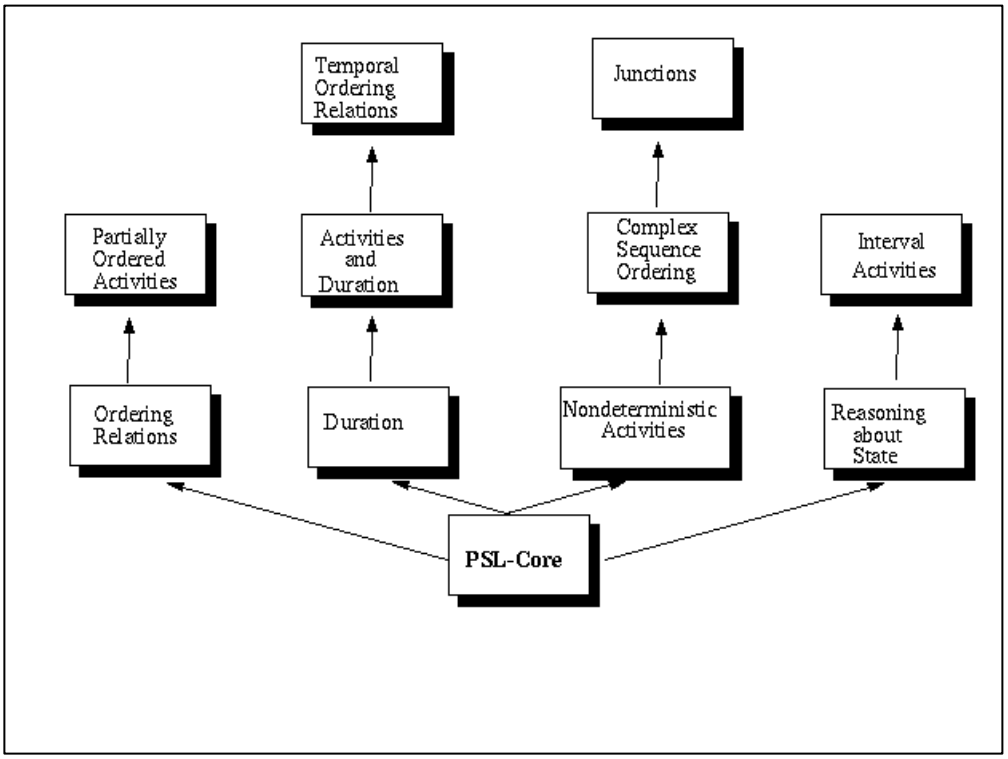

Figure 5: PSL modules for generic classes of activities and their ordering relations

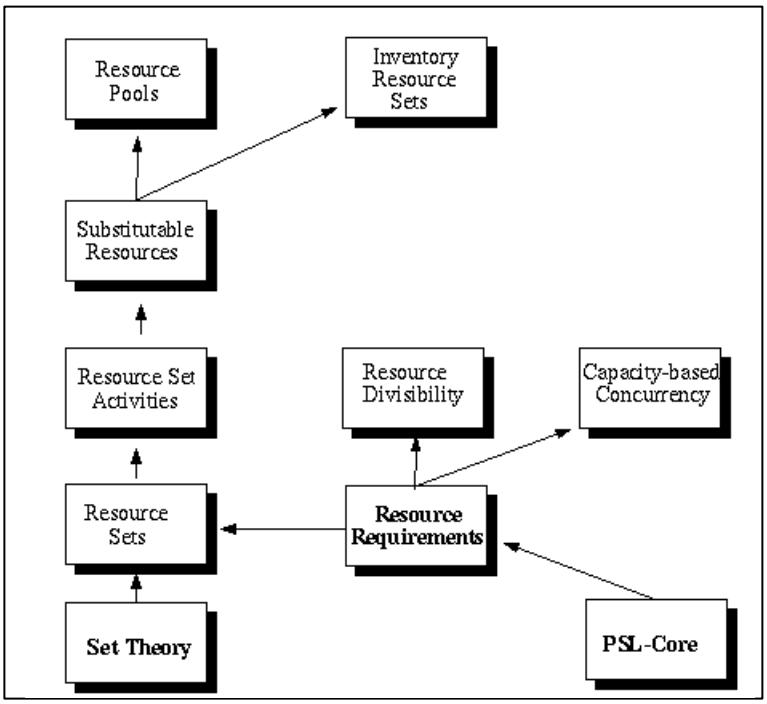

Figure 6: Resource-Related Extensions

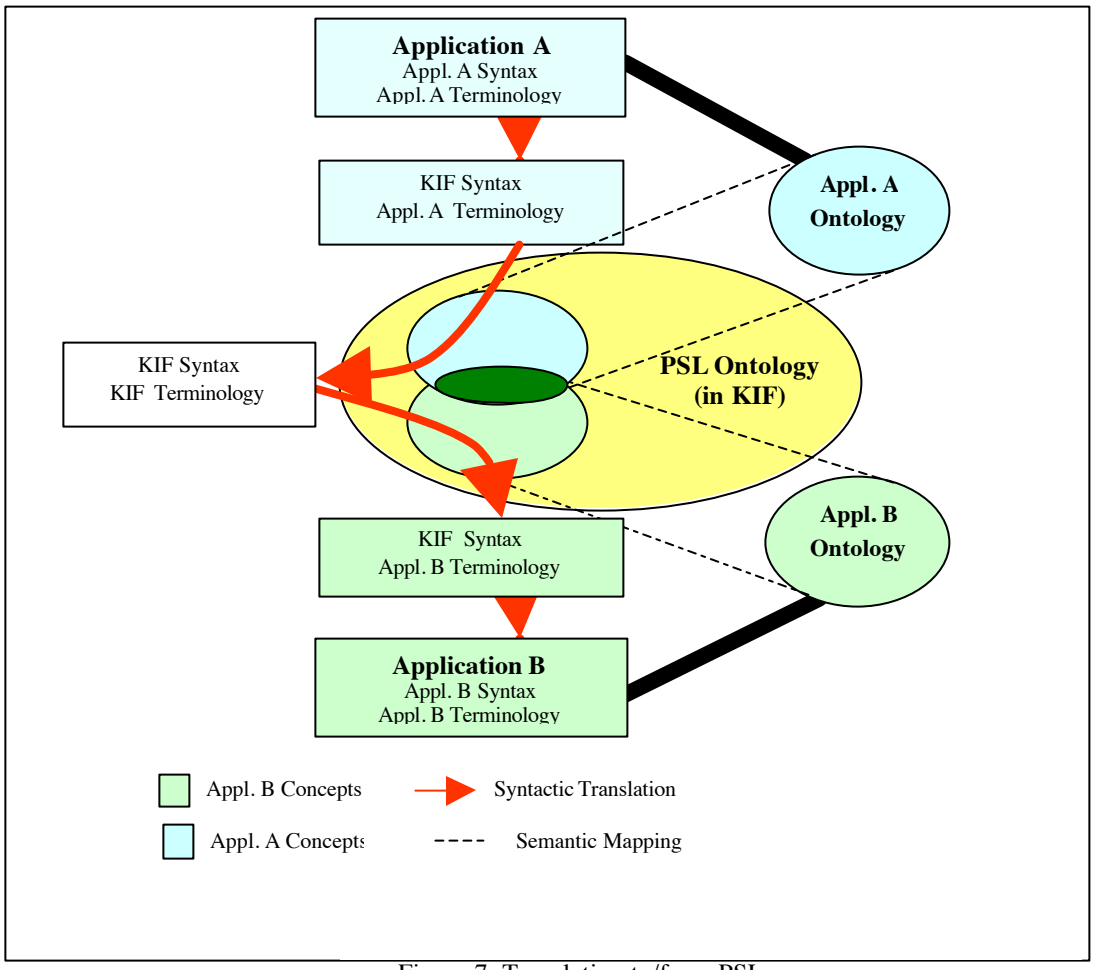

Figure 7: Translation to/from PSL

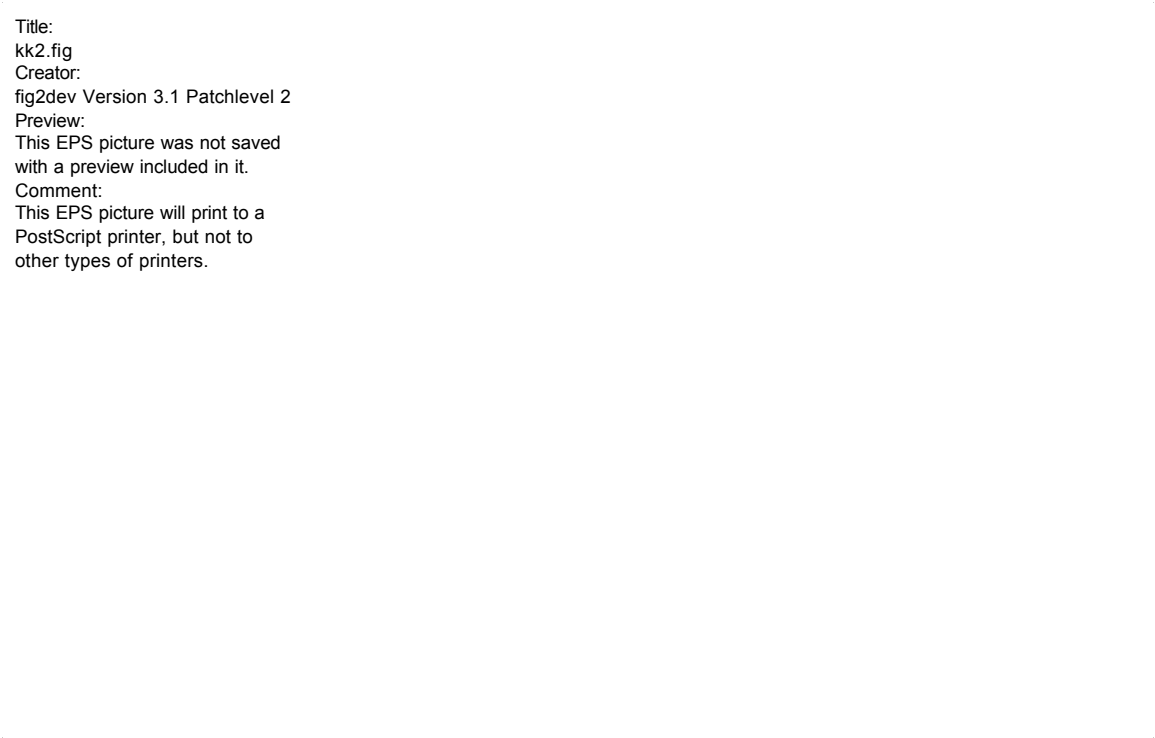

Figure 8: Translating In and Out of PSL

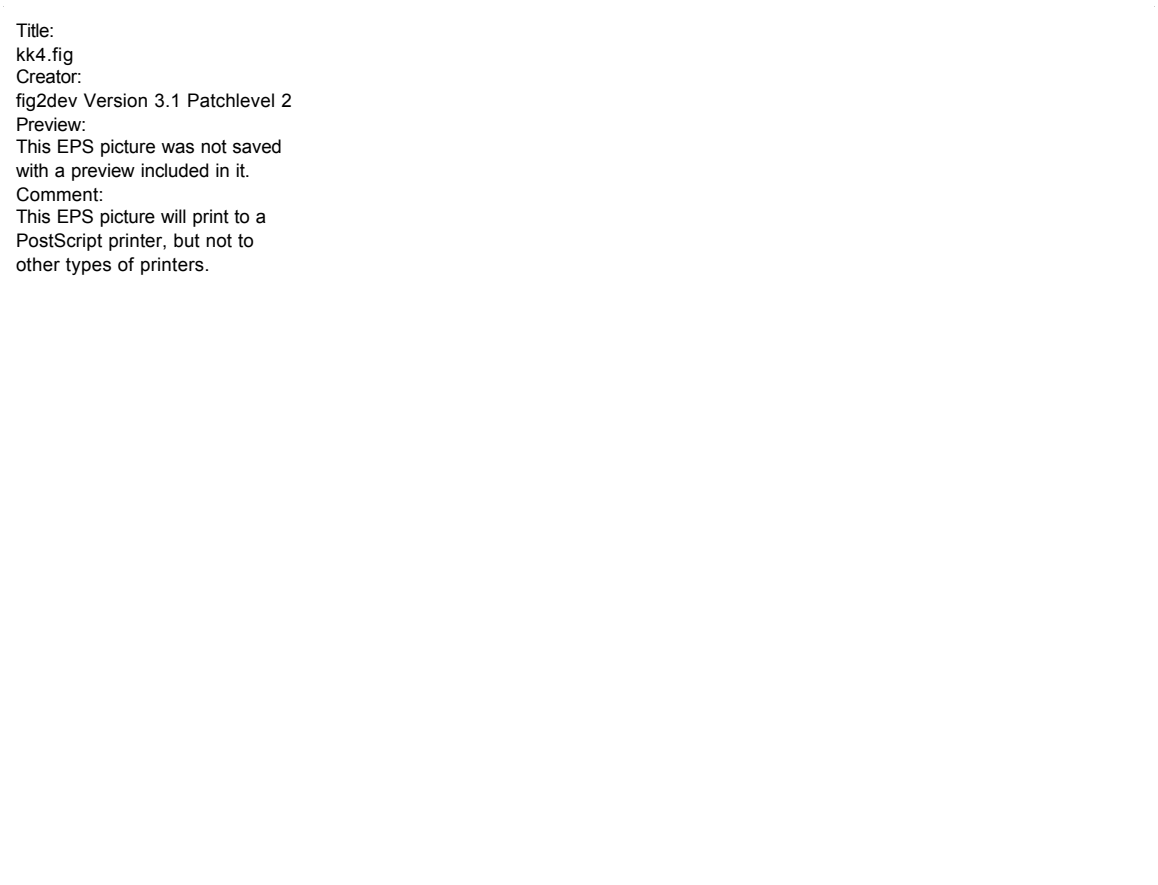

Figure 9: Challenges With Translating In and Out of an Additional Language

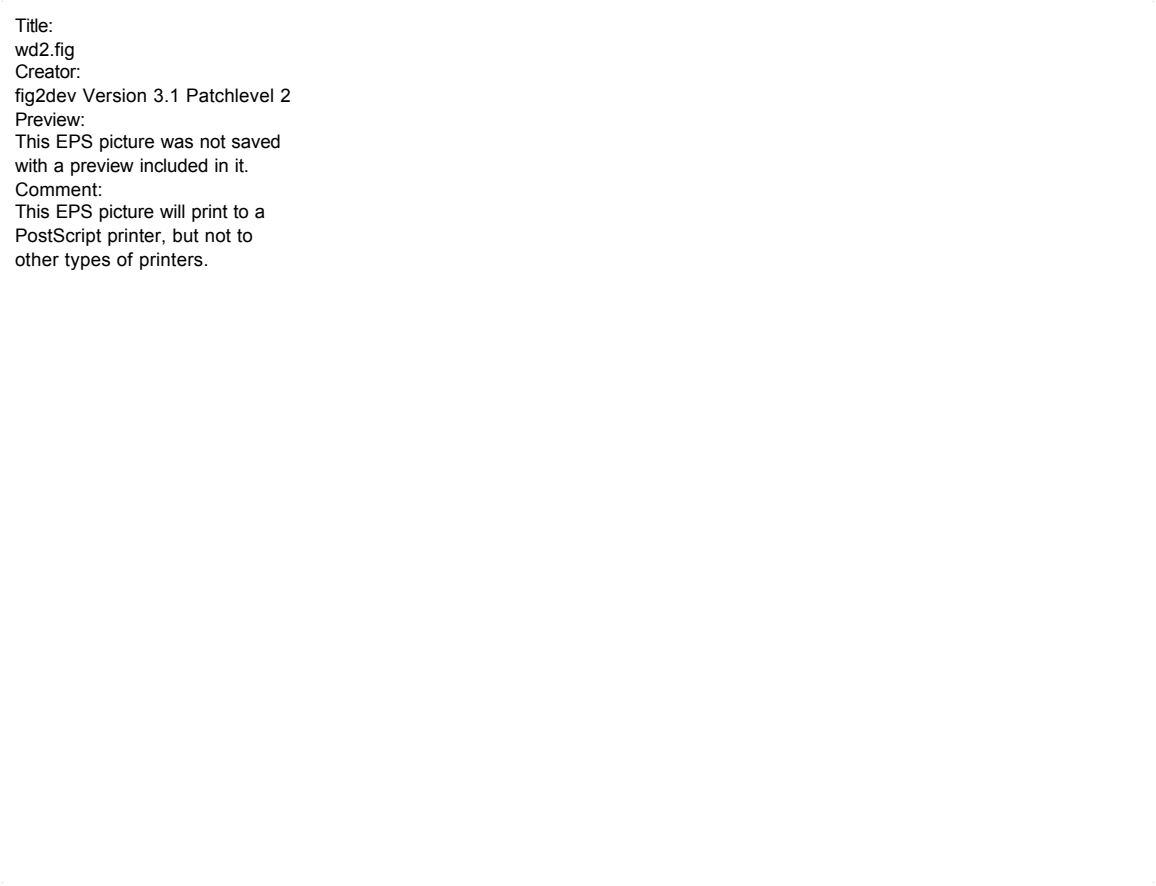

Figure 10: A Depiction of the Methodology when a New Language is Added

Appendix: An Example of the Challenges with Semantic Translation

This appendix illustrates the problems mentioned in the paper, and helps build one's intuitions about the semantic translation methodology.

(Partial) Translation

Let us consider the case of using PSL as an interlingua for translating from a language L1 to a language L2. Suppose we have some way of mapping a set of sentences (concepts) in L1 and L2 to PSL concepts. We say that a set S1 of L1-concepts is *partially translatable* into a set S2 of L2-concepts if the PSL concepts that are mapped to L1's concepts (i.e. Im(S1)) include the PSL concepts that are mapped to L2's concepts (i.e., Im(S2)).

We say that S1 is *translatable* to S2 if Im(S1) and Im(S2) are identical.

Translating In and Out of PSL

Figure 8 illustrates the process of using PSL as a language to translate process information.

Starting with some set S1 of L1-sentences, the translation definitions are applied to it, and the set Im(S1) is found.

The reverse translator has to now find a subset S2 of the PSL theory having Im(S1) as axioms (i.e. Cn(Im(S1))) and use the reverse translation rules to find the set InvIm(S2) whose image in PSL is S2. Finding such an S2 is difficult, since the set Cn(Im(S1)) is infinite. Also, writing the inverse translation rules from PSL to L2 is not as easy as writing direct translation rules from L1 to PSL. This is due firstly to the difference in language expressivity (i.e., PSL is usually more expressive than the languages that are

using it for translation). A second reason is language complexity and robustness: being an interlingua, PSL will be usually more robust and therefore larger than the languages being translated. An inverse translator must be able to translate all PSL sentences (S3 and S4 in Figure 8 are examples of such sets of PSL sentences) and thus specifying its rules will be a lot harder than specifying rules for translating into PSL.

Figure 9 illustrates the second challenge of translating in and out of PSL. That is, if PSL is extended after a reverse translator has been written, the reverse translator cannot handle sentences in the PSL extension, even though they might be translatable. (S5 is such a case, where it could be translated to InvIm(S5), but the reverse translator cannot handle that).

Of course if we would like to translate from another language L3 into L2 , and L3 has a set of sentences S6 for which the translator has been written in terms of the new extension, S6 cannot be translated by the combination of the L3-to-PSL translator plus L2 reverse translator, without first extending the reverse translator to handle the extension. This is not ideal, since in the worst case such an L2 translator might be rewritten for each new language. This would cause us to write $(N-1)^2$ translators, not an improvement over current practices. Another problem with this focuses on the person that has to do these translator rewritings. It can be the case that each organization maintaining a PSL compliant product dedicates a person to extending the PSL translator each time an extension is added to PSL, or that in the absence of such a person, the L3 expert has to do it herself, thus becoming an L2 expert in the process. Neither of these variants is very cost-effective.

A Methodology for Addressing this Challenge

Figure 10 illustrates a methodology for addresses these challenges. In order to be able to perform translation using PSL, a L1 language expert will write the semantic definitions of L1 concepts into PSL. In order to do this, he only needs to be familiar with the semantics of the language he is translating and that part of PSL he needs for expressing it. (as opposed to the necessity of being familiar with all current and future extensions of PSL for writing the reverse translator(s)).

Once the semantic definitions are written, that person's role is finished, and L1 is compliant with all future extensions of PSL.

Suppose now an L2 expert has similarly written L2's semantic definitions. To get a L2 to L3 translator, an automated procedure can be run that, based on L1 and L2's semantic definitions and on the (formal) PSL ontology, generates a direct translator from L1 to L2 (shown as the solid arrows in Figure 10).

When another language (L3), whose semantic definitions are written in terms of an extension that was not present at the time of writing L2's semantic definitions, is introduced, the L3 expert writes L3's semantic definitions as usual, and then runs the automated inference procedure. Even though the L2 definitions were written before the extension was added, they still work. This is because, at the time of running the inference procedure, the extensions' definitions are available for the inference mechanism to use (and figure out the set Im(S6)' axiomatizing the same PSL theory as Im(S6)).

References

[1] Schlenoff, C., Knutilla, A., Ray, S., Unified Process Specification Language: Requirements for Modeling Processes: NISTIR 5910, 1996, National Institute of Standards and Technology, Gaithersburg, MD.

[2] Knutilla, A., et al., Process Specification Language: An Analysis of Existing Representations, NISTIR 6133, 1998, National Institute of Standards and Technology, Gaithersburg, MD.

[3] Genesereth, M., Fikes, R.: Knowledge Interchange Format (Version 3.0) - Reference Manual, 1992, Computer Science Dept., Stanford University, Stanford, CA.

[4] Catron, B., Ray, S., ALPS: A Language for Process Specification, Int. J. Computer Integrated Manufacturing, 1991, Vol. 4, No. 2, 105-113.

[5] Fox, M., et al, An Organization Ontology for Enterprise Modeling: Preliminary Concepts", Computers in Industry, 1996, Vol. 19, pp. 123-134.

[6] Uschold, M., et al., The Enterprise Ontology, The Knowledge Engineering Review, 1998, Vol. 13, Special Issue on Putting Ontologies to Use.

[7] Pease, A., Core Plan Representation (CPR), http://www.teknowledge.com/CPR2/, November 13, 1998.

[8] Tate, A., Planning Initiative: Shared Planning and Activity Representation: SPAR Homepage, http://www.aiai.ed.ac.uk/~arpi/spar/, November 30, 1998.

[9] Lee, J., et al, The PIF Process Interchange Format and Framework, to appear in Knowledge Engineering Review, 1998, Vol. 13, No. 1, Cambridge Univ. Press.

[10] Workflow Management Coalition Members, Glossary: A Workflow Management Coalition Specification, Belgium, 1994.

[11] Uschold, M. and Gruninger M., Ontologies: Principles, Methods, and Applications, Knowledge Engineering Review, 1996, Vol. 1, pp. 96-137.

[12] Reiter, R., The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression, Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy, 1991, pages 418-440, Academic Press, San Diego.

[13] PSL Ontology – Current Theories and Extensions, http://www.nist.gov/psl/psl-ontology/, June 28, 1999.

[14] Hayes, P., A Catalog of Temporal Theories, Tech Report UIUC-BI-AI-96-01, 1996, University of Illinois.

[15] Smith, S.J.J., at al., Integrating Electrical and Mechanical Design and Process Planning, Proceedings of the IFIP Knowledge Intensive CAD Workshop, 1996, Carnegie-Mellon University (CMU).

[16] ISO 10303-1:1994, Product data representation and exchange: Part 1: Overview and fundamental principles.

---

[1] No approval or endorsement of any commercial product in this paper by the National Institute of Standards and Technology is intended or implied. This paper was prepared by United States Government employees as part of their official duties and is, therefore, a work of the U.S. Government and not subject to copyright.