SEMANTIC INTEGRATION THROUGH INVARIANTS

MICHAEL GRÜNINGER AND JOSEPH B. KOPENA

1. INTRODUCTION

Many tasks require correct and meaningful communication and integration among intelligent agents and information resources. A major barrier to such interoperability is semantic heterogeneity: different applications, databases, and agents may ascribe disparate meanings to the same terms or use distinct terms to convey the same meaning. The development of ontologies have been proposed as a key technology to support semantic integration—two software systems can be completely semantically integrated through a shared understanding of the terminology in their respective ontologies.

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, determining that they are correct, and providing a vehicle for executing the mappings, thus translating terms from one ontology into another.

Current approaches to semantic integration do not fully exploit the model-theoretic structures underlying ontologies. They are typically based on the taxonomic structure of the terminology ([11], [12]) or heuristics-based comparisons of the symbols of the terminology ([1, 8]). These approaches are well-suited to working with many ontologies currently under development, most of which define a terminology with minimal formal grounding and a set of possible models which does not contain a rich set of features and properties.

However, automated and correct approaches to semantic integration will require ontologies with a deeper formal grounding so that strong decisions may be made by automated process in comparing ontologies for integration. This article presents an approach to this goal, by presenting techniques based on the development of strong ontologies with terminologies grounded in properties of the underlying possible models. With these as inputs, semi-automated and automated components may be used to create mapping between ontologies and perform translations.

The Process Specification Language (PSL) ([5], [6]) is used in this article to demonstrate this approach to ontology construction and integration. It has been designed to facilitate correct and complete exchange of process information among manufacturing systems such as scheduling, process modeling, process planning, production planning, simulation, project management, workflow, and business process reengineering. Its primary function is as a neutral interchange ontology, providing a mediator between integration targets ([2]). As of June 2004, Part 1 of PSL has been accepted as an International Standard through project ISO 18629 within the International Organisation of Standardisation.

PSL consists of a core ontology which outlines basic objects that exist in the domain, and a multitude of definitional extensions that provide a rich terminology for describing process knowledge. These extensions are based on invariants, properties of the models preserved by isomorphism, which partition the first-order models of the core ontology.

Using these invariants, semantic mappings between application ontologies and PSL may be semi-automatically generated. In addition, the direct relationship between the PSL terminology and the invariants improves the ability to verify the generated results. These semantic mappings may then be used to perform integration between applications or ontologies. They may also be used to bootstrap an ontology to those applications which do not have an associated, explicit, formal ontology as well as to analyze the application.

This article first describes the integration architecture which outlines the entire process. A brief overview of PSL is then provided, particularly in regards to the construction of its terminology. The form and process of creating semantic mappings is then given. Application and ontology analysis and integration is then presented, followed by a concluding discussion of open problems in this area and a summary of this article.

2. An Architecture for Semantic Integration

This section describes the Interlingua Architecture, the basic approach to application integration employed in this work. Semantic integration is then presented in terms of this architecture as the tasks and questions which must be performed and answered.

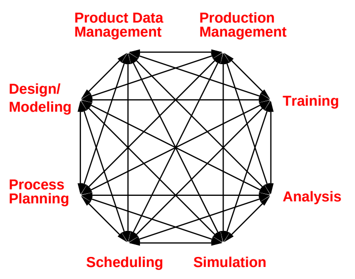
2.1. **The Interlingua Architecture.**

Informally, semantic mappings express the meaning of a term from one ontology in terms of the other ontology; each such mapping may simply link one term to another or may specify a complex transformation. More formally, semantic mappings are characterized by the notion of definable interpretation ([9]): Let $\mathscr{N}$ be a structure in the language $\mathscr{L}_0$ and $\mathscr{M}$ be a structure in the language $\mathscr{L}$. We say that $\mathscr{N}$ is definably interpretable in $\mathscr{M}$ iff we can find a definable subset X of M^n and we can interpret the symbols of $\mathscr{L}_0$ as definable subsets and functions on X so that the resulting structure in $\mathscr{L}_0$ is isomorphic to $\mathscr{N}$. An ontology T_1 is definably interpretable in an ontology T_2 iff every model of T_1 is definably interpretable in a model of T_2.

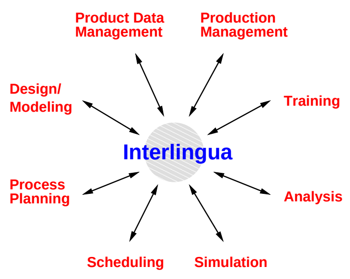
In current practice, semantic mappings are manually generated directly between the application ontologies. However, for software applications operating in open environments such as the Semantic Web, it cannot be assumed that the mappings have been generated prior to the interaction between the applications. In [7], a number of architectures have been proposed to support semantic integration in such an open environment. Each architecture is distinguished by the origins of the semantic mappings, the existence of a mediating ontology, and the degree of agreement that exists among the anticipated community of interacting software.

The *Interlingua Architecture* is adopted within this work, the distinguishing feature of which is the existence of a mediating ontology that is independent of the applications' ontologies and is used as a neutral interchange ontology ([2]). Semantic mappings between application and interlingua ontologies are manually generated and verified prior to application interaction time [3]. This process of creating the mapping between the application ontology and the interlingua ontology is identical to the process of creating a mapping directly between two application ontologies, the key difference of this approach being that application ontologies are integrated with the interlingua rather than each other.

The most obvious property of this approach is the dramatic reduction of the number of translators which must be constructed. The manual, point-to-point approach requires on the order of n^2 translators, one for each pairing, while the interlingua approach mandates only one translator per application. In addition to the initial costly development of a translator for each pairing under the point-to-point approach, if one application's ontology is



(a) Point-to-point translation requires on the order of n^2 translators be developed and maintained.



(b) With an interlingua, only n translators must be developed and maintained.

FIGURE 1. Translation pairings for a set of manufacturing process systems.

changed, each associated translator must be updated. Using an interlingua, only the translator to and from the interlingua must be maintained for each application. [1]. An example of these properties from the domain of systems for managing manufacturing processes is shown in Figure 1.

Importantly, the point-to-point approach does not work in environments which feature unanticipated software interactions. Interaction can only occur between pairs of software for which a specific translator has been previously developed. Using the interlingua model, a mapping between the application ontology and the interlingua is all that is necessary to interact with the community of software for which mappings to and from the interlingua have also been developed. This eliminates the problem of changes in applications mandating changes to all other systems, and allows existing software to seamlessly interoperate with newly introduced applications. This is not possible using manual, point-to-point mappings.

2.2. **Integration and Translation.** Under the Interlingua Architecture, there are two steps in translation: the execution of the mapping from the application ontology to the interlingua and subsequently from the interlingua to the target application's ontology. If the application ontologies and the interlingua ontology are specified using the same logical language, then translation can be accomplished by applying deduction to the axioms of the interlingua ontology in conjunction with the formal mapping rules ([3], [2]). In effect, a direct mapping rule from one application's ontology to the target application's ontology is inferred from the two separate rules. If these mapping rules have been verified to preserve semantics between the application and interlingua ontology, it is guaranteed that this translation between the applications also preserves semantics.

An important question is then whether the existence of the pre-defined mappings between the application ontologies and the interlingua ontology enables the automatic generation of a point-to-point mapping between the applications' ontologies. More formally, if $\mathscr{M}_1$ and $\mathscr{M}_2$ are both definably interpretable in $\mathscr{N}$, is $\mathscr{M}_1$ definably interpretable $\mathscr{M}_2$? Answering this question is equivalent to the task of semantic integration within the Interlingua Architecture; it is addressed in this work by comparing the mappings between application ontologies and the interlingua.

[1]See [13] for a more detailed discussion of the tradeoffs between the point-to-point and interlingua approaches.

3. Invariant-Based Ontology Design

This section briefly overviews the core of the PSL Ontology. It then introduces the notion of invariant-based ontologies and demonstrates how this approach has been used in defining the extensive PSL terminology.

3.1. **Core Theories of The Process Specification Language.**

The PSL ontology is a set of theories in the language of first order logic. Theories that introduce new primitive concepts are referred to as core theories, while theories containing only conservative definitions are referred to as definitional extensions[2].

All core theories within the ontology are consistent extensions of PSL-Core, (T_{psl_core}), which axiomatizes a set of intuitive semantic primitives that is adequate for describing the fundamental concepts of manufacturing processes. Specifically, PSL-Core introduces four disjoint classes: activities, activity occurrences, timepoints, and objects. Activities may have zero or more occurrences, activity occurrences begin and end at timepoints, and timepoints constitute a linearly ordered set with endpoints at infinity. Objects are simply those elements that are not activities, occurrences, or timepoints. Extensions to PSL-Core defining the core theories include axiomatizations of occurrence trees, discrete states, subactivities, atomic activities, and complex activities.

3.1.1. *Occurrence Trees.* The occurrence trees that are axiomatized in the core theory $T_{occtree}$ are partially ordered sets of activity occurrences—for a given set of activities, all discrete sequences of their occurrences are branches of a tree. An occurrence tree contains all occurrences of *all* activities, not simply the set of occurrences of a particular (possibly complex) activity. As each tree is discrete, every activity occurrence in the tree has a unique successor occurrence of each activity.

There are constraints on which activities can possibly occur in some domain. This intuition is the cornerstone for characterizing the semantics of classes of activities and process descriptions. Although occurrence trees characterize all sequences of activity occurrences, not all of these sequences will intuitively be physically possible within the domain. We will therefore want to consider the subtrees of the occurrence trees that consist only of *possible* sequences of activity occurrences; such a subtree is referred to as a legal occurrence tree.

3.1.2. *Discrete States.* The core theory T_{disc_state} introduces the notion of fluents (state). Fluents are changed only by the occurrence of activities, and fluents do not change during the occurrence of primitive activities. In addition, activities have preconditions (fluents that must hold before an occurrence) and effects (fluents that always hold after an occurrence).

3.1.3. *Subactivities.* The PSL Ontology uses the *subactivity* relation to capture the basic intuitions for the composition of activities. This relation is a discrete partial ordering in which primitive activities are the minimal elements.

3.1.4. *Atomic Activities.* The core theory T_{atomic} axiomatizes intuitions about the concurrent aggregation of primitive activities. This is represented by the occurrence of concurrent activities, rather than concurrent activity occurrences.

[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 by a .def suffix. As of June 2004, the ontology is in version 2.2.

3.1.5. *Complex Activities.* The core theory $T_{complex}$ characterizes the relationship between the occurrence of a complex activity and occurrences of its subactivities. Occurrences of complex activities correspond to sets of occurrences of subactivities; in particular, they form subtrees of the occurrence tree. An activity tree consists of all possible sequences of atomic subactivity occurrences beginning from a root subactivity occurrence. In a sense, activity trees are a microcosm of an 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.*

Each activity tree is composed of a set of isomorphic copies of a unique minimal activity tree consisting only of subactivity occurrences. Not every occurrence of a subactivity is a subactivity occurrence; there may be other external activities that occur during an occurrence of an activity, or subactivity occurrences may need to satisfy temporal constraints. Within models of $T_{complex}$, these constraints on subactivity occurrences are captured by different ways of embedding the minimal activity tree into the activity tree.

3.2. **Terminologies Based on Classification by Invariant.**

Many ontologies are specified as taxonomies or class hierarchies, yet few provide formal justification for their classification scheme. If we consider ontologies of mathematical structures, we see that logicians classify models by using properties of models, known as invariants, that are preserved by isomorphism.

For some classes of structures, invariants can be used to classify the structures up to isomorphism; for example, vector spaces can be classified up to isomorphism by their dimension. For other classes of structures, such as graphs, it is not possible to formulate a complete set of invariants. However, even without a complete set, invariants can still be used to provide a classification of the models of a theory. Figure 2 provides such an example from the domain of geometric shapes. Some invariants of objects in this domain are given in Figure 2(a). These are used in Figure 2(b) to define the class of regular shapes. Several shapes are classified against this definition and the results given in Figure 2(c).

Following this methodology, the set of models for the core theories of PSL are partitioned into equivalence classes defined with respect to the set of invariants of the models. Each equivalence class in the classification of PSL models is axiomatized using a definitional extension of PSL. Each definitional extension in the PSL Ontology is associated with a unique invariant; the different classes of activities or objects that are defined in an extension correspond to different properties of the invariant. In this way, the terminology of the PSL Ontology arises from the classification of the models of the core theories with respect to sets of invariants and intuitively corresponds to classes of activities and objects.

Models of the core theory of complex activities are classified with respect to invariants related to substructures of activity trees. In particular, the models can be completely characterized by four invariants—the minimal activity tree, the variation of the minimal activity tree across the occurrence tree, the way in which the minimal activity tree is embedded into the activity tree, and the distribution of activity trees within the occurrence tree. Each of these substructures can in turn be classified with respect to their own sets of invariants.

Many of the invariants corresponding to definitional extensions in the PSL Ontology are related to the automorphism groups or semigroups [3] for different substructures of the models. The following subsections demonstrate two examples of PSL invariants, one defined with respect to models of the theory $T_{occtree} \cup T_{disc\_state}$ and the other defined with respect to models of $T_{complex}$.

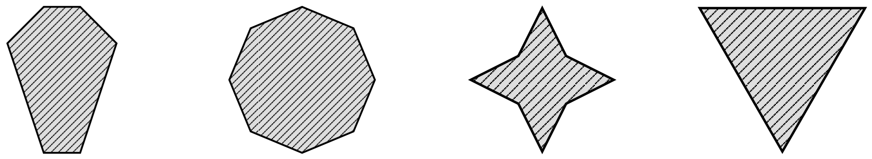
[3]An automorphism is a bijection from a structure to itself that preserves the extensions of the relations and functions in the structure; intuitively, it is a symmetry in the structure.

- Is the shape a polygon with $n \geq 3$ sides?
- Is the shape convex?
- Is the symmetry group *Symm* of the shape $\equiv D_n$, which consists of the rotations $R_{2k\mathrm{p}/n}$ for $k = 0, 1, \ldots, n-1$ and the reflections $R_{l_1}, \ldots, R_{l_n}$ about the lines $l_1, \ldots, l_n$ connecting the centroid to the vertices and midpoints?

(a) Several invariant properties of geometric shapes.

Regular polygons $\equiv$ convex polygons w/ $n \geq 3$ sides and symmetry group $\equiv D_n$.

(b) Definition for the class of regular shapes using the above invariants.



yes, $n = 6$;	yes, $n = 8$;	yes, $n = 8$;	yes, $n = 3$;
convex;	convex;	concave;	convex;
$Symm \not\equiv D_n$	$Symm \equiv D_n$	$Symm \not\equiv D_n$	$Symm \equiv D_n$
$\supset \neg$ regular	$\supset$ regular	$\supset \neg$ regular	$\supset$ regular

(c) Several shapes classified as regular or irregular through comparison to the definition.

FIGURE 2. The use of invariants in constructing a terminology of geometric shapes. Although not a complete set, these invariants do provide for formally defining terms in the language.

3.2.1. *Preconditions.* With respect to the models of the theory $T_{occtree}$, we can consider mappings that are permutations of activity occurrences that preserve legal occurrences of an activity **a** in an occurrence tree; this set of mappings forms a group, which is referred to as $OP(\mathbf{a})$. Each invariant related to occurrence constraints is based on subgroups of this group.

The most prevalent class of occurrence constraints is that of Markovian activities, activities whose preconditions depend only on the state prior to the occurrences. The class of Markovian activities is defined in the definitional extension *state_precond.def*, a portion of which is given in Figure 3.

The invariant associated with this extension is the group $\mathscr{P}^{\mathscr{F}}(\mathbf{a})$, which is the maximal normal subgroup of $Aut(\mathscr{F})$ that is also a subgroup of $OP(\mathbf{a})$. In this example, $\mathscr{F}$ is the structure isomorphic to the extension of the *prior* relation. $Aut(\mathscr{F})$ is the group of permutations that map activity occurrences only to other activity occurrences that agree on the set of fluents that hold prior to them. If $\mathscr{P}^{\mathscr{F}}(\mathbf{a}) = Aut(\mathscr{F})$, then these permutations preserve the legal occurrences of an activity, and the activity's preconditions are strictly Markovian; this is axiomatized by the *markov_precond* class in Figure 3. If $\mathscr{P}^{\mathscr{F}}(\mathbf{a})$ is only a subgroup of $Aut(\mathscr{F})$, then there exist additional non-Markovian constraints on the legal occurrences of the activity; this is axiomatized by the *partial_state* class in Figure 3.

(1) $$(\forall o_1, o_2)\, state\_equiv(o_1, o_2) \equiv$$
$$(\forall f)\,(prior(f, o_1) \equiv prior(f, o_2))$$

(2) $$(\forall a, o_1, o_2)\, poss\_equiv(a, o_1, o_2) \equiv$$
$$(poss(a, o_1) \equiv poss(a, o_2))$$

(3) $$(\forall a)\, markov\_precond(a) \equiv$$
$$((\forall o_1, o_2)\, state\_equiv(o_1, o_2) \supset poss\_equiv(a, o_1, o_2))$$

(4) $$(\forall a)\, partial\_state(a) \equiv$$
$$(\exists o_1)\,((\forall o_2)\, state\_equiv(o_1, o_2) \supset poss\_equiv(a, o_1, o_2))$$
$$\wedge (\exists o_3, o_4)\, state\_equiv(o_3, o_4) \wedge \neg poss\_equiv(a, o_3, o_4)$$

(5) $$(\forall a)\, rigid\_state(a) \equiv$$
$$(\forall o_1)(\exists o_2)\, state\_equiv(o_1, o_2) \wedge \neg poss\_equiv(a, o_1, o_2)$$

FIGURE 3. Classes of activities with state-based preconditions from the definitional extension $state\_precond.def$.

If $\mathscr{P}^{\mathscr{F}}(\mathbf{a})$ is the trivial identity group, then there are no Markovian constraints on the legal occurrences of the activity; this is axiomatized by the *rigid_state* class in Figure 3.

The additional relations in Figure 3 are defined to capture the action of the automorphism groups on the models. Two activity occurrences o_1, o_2 are *state_equiv* iff there exists a permutation in $Aut(\mathscr{F})$ that maps o_1 to o_2; the two activity occurrences are *poss_equiv* iff there exists a permutation in $OP(\mathbf{a})$ that maps o_1 to o_2.

3.2.2. *Classes of Complex Activities.* This second example concerns invariants for complex activities based on symmetries of the activity trees. The automorphism groups used here are defined as sets of mappings between branches that preserve different properties of the activity trees.

Two branches of an activity tree are occurrence-isomorphic if there is a one-to-one mapping of subactivity occurrences that preserves the activities—occurrences of an activity are mapped to occurrences of the same activity, but order is not preserved. In the activity tree in Figure 4, the mapping $(\mathbf{o_1}, \mathbf{o_2}, \mathbf{o_3}, \mathbf{o_4}) \leftrightarrow (\mathbf{o_7}, \mathbf{o_8}, \mathbf{o_5}, \mathbf{o_6})$ is an occurrence-isomorphism; a triangle is mapped to a triangle, a circle to a circle, and so on.

One invariant for activity trees considers the group of occurrence isomorphisms. An activity tree is *permuted* if this group is symmetric, *nondet_permuted* if it is the product of symmetric groups, *partial_permuted* if there are no branches that are fixed, and *simple* if there are no nontrivial permutations of subactivity occurrences in the activity tree.

Two branches of an activity tree are order automorphic if there is a one-to-one mapping of subactivity occurrences that preserves the ordering and which also allows permutation of the activities. In the activity tree in Figure 4, the mapping $(\mathbf{o_1}, \mathbf{o_2}, \mathbf{o_3}, \mathbf{o_4}) \leftrightarrow (\mathbf{o_5}, \mathbf{o_6}, \mathbf{o_7}, \mathbf{o_8})$ is an order-isomorphism, in which there is also the following mapping on the set of activities: $(\mathbf{a_1}, \mathbf{a_2}, \mathbf{a_3}, \mathbf{a_4}) \Leftrightarrow (\mathbf{a_3}, \mathbf{a_4}, \mathbf{a_1}, \mathbf{a_2})$. In Figure 4, this is depicted as mapping a triangle to a circle, and an inverted triangle to a square.

Another invariant for activity trees considers the group of order automorphisms. An activity tree is *ordered* if this group is the automorphism group of a tree, *nondet_ordered*

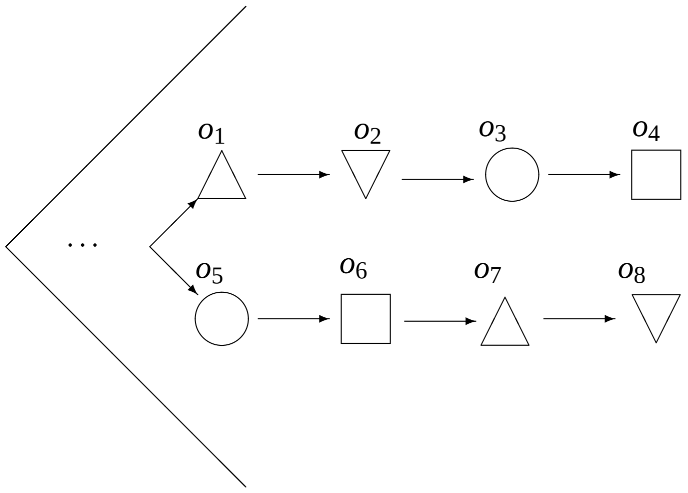


FIGURE 4. Example of an activity tree for a *Split* activity in OWL-S. Note that the diagram depicts two separate activity trees within a stylized legal occurrence tree. Also, occurrences of different subactivities are depicted by different shapes; triangles are occurrences of the activity a_1, inverted triangles are occurrences of a_2, circles are occurrences of a_3, and squares are occurrences of a_4.

if it is the product of the automorphism groups of a set of trees, and *unordered* if there are no nontrivial order automorphisms in the activity tree.

4. SEMANTIC MAPPING VIA TRANSLATION DEFINITIONS

As noted in Section 2, the generation of semantic mappings between two ontologies T_1 and T_2 is equivalent to the formal problem of determining whether T_1 is definably interpretable in T_2. Although in general an extremely difficult problem, the invariants used in the classification of the models of the ontologies can also be used to generate semantic mappings. Semantic mappings preserve models—each model of the ontology T_1 is mapped to an isomorphic substructure of a model of the ontology T_2. Since invariants are properties of the models that are preserved by isomorphism, semantic mappings must also preserve the invariants. Therefore, if models of T_1 and T_2 are characterized up to isomorphism by some sets of invariants, then T_1 is definably interpretable in T_2 iff there is a mapping of the invariants of T_1 to the invariants of T_2; a concept in T_1 will be mapped to a concept in T_2 iff the invariants have the same values.

Translation definitions specify the semantic mappings between PSL and application ontologies. Following the above discussion, they are generated using the organization of the definitional extensions, each of which corresponds to a different invariant. Every class of activity, activity occurrence, or fluent in an extension corresponds to a different value for the invariant. The consequent of a translation definition is equivalent to the list of invariant values for members of the application ontology class.

Translation definitions have a special syntactic form—they are biconditionals in which the antecedent is a class in the application ontology and the consequent is a formula that uses only the lexicon of the PSL Ontology. For example, the concept of *AtomicProcess* in the OWL-S Ontology ([10]) has the following translation definition:

$$(\forall a)\, AtomicProcess(a) \equiv primitive(a) \wedge markov\_precond(a) \wedge$$
$$(markov\_effects(a) \vee context\_free(a)).$$

The invariant corresponding to the *markov_precond* class was discussed in the Section 3.2.1; the invariants corresponding to the *markov_effects* and *context_free* classes are based on automorphism groups consisting of permutations of activity occurrences that preserve effects (i.e., fluents that are achieved or falsified by activity occurrences).

Similarly, the classes of activity trees associated with the occurrence isomorphism and order isomorphism groups described in Section 3.2.2 can be applied to specify translation definitions for additional classes of activities in OWL-S. In an OWL-S Split activity, sets of subactivities are performed in parallel. There exist nontrivial permutations of subactivity occurrences among the branches of the activity trees, as shown by the example in Figure 4, so that the translation definition is:

$$(\forall a)\, Split(a) \equiv uniform(a) \wedge (\exists o)\, occurrence\_of(o,a) \wedge$$
$$\neg simple(o) \wedge ordered(o) \wedge strong\_poset(o).$$

This translation definition has two parts; the first part declares that the activity is uniform—all activity trees for the activity are isomorphic. The second part states that there exists an activity tree for the activity which is ordered and not simple.

4.1. **Semi-Automatic Generation of Semantic Mappings.** The generation of semantic mappings through the specification of invariant values has been implemented in the PSL project's Twenty Questions mapping tool [4]. Each question corresponds to an invariant, and each value of the invariant is a possible answer to the question. Any particular activity, activity occurrence, or fluent will have a unique value for the invariant; however, if we are mapping a class of activities, occurrences, or fluents from some application ontology, then different members of the class may have different values for the same invariant. In such a case, one would respond to a question by supplying multiple answers. By guiding and supporting users in creating translation definitions without requiring them to work directly with first order logic axiomatizations, the Twenty Questions tool provides a semi-automated technique for creating semantic mappings.

Figure 5 gives a sample question corresponding to the invariant $\mathscr{P}^{\mathscr{F}}(\mathbf{a})$. The possible values for this invariant are subgroups of $Aut(\mathscr{F})$, so that exactly one of the following must be true: $\mathscr{P}^{\mathscr{F}}(\mathbf{a}) = Aut(\mathscr{F})$, $\mathscr{P}^{\mathscr{F}}(\mathbf{a}) < Aut(\mathscr{F})$, or $\mathscr{P}^{\mathscr{F}}(\mathbf{a}) = I$. Following the axiomatizations given in Figure 3 for the classes of activities corresponding to these values, selecting the first answer would generate the translation definition:

$$(\forall a)\, myclass(a) \equiv markov\_precond(a).$$

Selecting the first two answers would give the translation definition:

$$(\forall a)\, myclass(a) \equiv (markov\_precond(a) \vee partial\_state(a)).$$

In this latter case, some activities in *myclass* will have Markov preconditions while other activities will not.

This example raises the issue of validating the semantic mappings that are generated in this way—how can we determine the correctness of the mappings between an application ontology and the PSL Ontology? If the application ontologies are axiomatized, then we can verify the semantic mappings by proving that they do indeed preserve the models of the

[4]Available at `http://ats.nist.gov/psl/twenty.html`.

2. Constraints on Atomic Activity Occurrences based on State

Are the constraints on the occurrence of the atomic activity based only on the state prior to the activity occurrence?

- ☐ Any occurrence of the activity depends only on fluents that hold prior to the activity occurrence.
- ☐ Some (but not all) occurrences of the activity depend only on fluents that hold prior to the activity occurrence.
- ☐ There is no relationship between occurrences of the activity and the fluents that hold prior to occurrences of the activity.

FIGURE 5. One of the Twenty Questions, used to classify activities with state-based preconditions.

ontologies. This can be done in by demonstrating that the class of models of the application ontology is axiomatized by the interlingua, in this case PSL, together with the translation definitions.

4.1.1. *Bootstrapping Application Ontologies.* Software such as the Twenty Questions tool provides support for other tasks in addition to creating mappings between application ontologies and the interlingua. In particular, they may be used to define a formal ontology for applications which do not have an explicitly axiomatized ontology. This is afforded by the assumption of the *Ontological Stance* ([6]), the main tenet of which is that a software application may be modeled as if it were an inference system working on an axiomatized ontology.

The Ontological Stance is an operational characterization of the set of intended models for the application's terminology. In this sense, it should be treated as a semantic constraint on the application—it does not postulate a specific set of axioms, but rather a set of intended models. Given a software application, there exists a class of models $\mathscr{M}^A$ such that any sentence Φ is decided by the application to be satisfiable iff there exists $\mathscr{M} \in \mathscr{M}^A$ such that $\mathscr{M} \models \Phi$.

By answering the questions presented by the Twenty Questions tool, the application designer is capturing the application's set of intended models. Given correct input, the translation definitions generated by the tool together with the interlingua ontology define an explicit axiomatization of the application's previously implicit ontology.

To validate the attributed ontology, the generated translation definitions may be treated as falsifiable hypotheses and tested empirically. By the Ontological Stance, the application decides some sentence Φ to be provable iff $T_{psl} \cup T_{translation} \models \Phi$ where T_{psl} is the set of axioms for the PSL Ontology and $T_{translation}$ is the set of translation definitions that are being verified. In this way, it may be evaluated whether or not the attributed ontology correctly predicts inferences made by the software, and consequently whether or not the translation definitions accurately capture the semantics of the application.

5. COMPARISON OF SEMANTIC INTEGRATION PROFILES FOR INTEGRATION

The set of translation definitions for all concepts in a software application's ontology defines a *semantic integration profile* for that application. If the interlingua has m invariants and each invariant n values, then an application profile will have the form:

$$(\forall a)\, C_1^{onto}(a) \equiv (p_{11}(a) \vee \ldots \vee p_{1n}(a)) \wedge \ldots \wedge (p_{m1}(a) \vee \ldots \vee p_{mn}(a))$$

$$\vdots$$

$$(\forall a)\, C_k^{onto}(a) \equiv (p_{11}(a) \vee \ldots \vee p_{1n}(a)) \wedge \ldots \wedge (p_{m1}(a) \vee \ldots \vee p_{mn}(a))$$

Note that although the translation definition for the OWL-S Split class as presented in Section 4 appears to not follow this form, the variation is merely syntactic. Some invariants of the PSL ontology do not pertain directly to classes of activities, but rather indirectly through associated classes of sets of occurrences.

As noted in Section 2.2, translation between integration targets may be accomplished by applying deduction to the axioms of the interlingua, the semantic mappings, and the input to be translated. For example, given the following mappings from two application ontologies into PSL:

$$(\forall a)\, C_1^{alice}(a) \equiv unconstrained(a) \wedge (markov\_effects(a) \vee context\_free(a)).$$

$$(\forall a)\, C_1^{bob}(a) \equiv (unconstrained(a) \vee markov\_precond(a)) \wedge context\_free(a).$$

The following mappings between the two concepts may be inferred:

$$T_{psl} \models (\forall a)\, markov\_precond(a) \supset (C_1^{alice}(a) \supset C_1^{bob}(a)).$$

$$T_{psl} \models (\forall a)\, markov\_effects(a) \supset (C_1^{bob}(a) \supset C_1^{alice}(a)).$$

Such mappings will in general take the form of:

$$(\forall a)\, (p_{11}(a) \vee \ldots \vee p_{1n}(a)) \wedge \ldots \wedge (p_{m1}(a) \vee \ldots \vee p_{mn}(a)) \supset (C_i^{alice}(a) \supset C_j^{bob}(a)))$$

The antecedents of these sentences can be considered to be guard conditions that determine which activities can be shared between the two ontologies. This can either be used to support direct exchange, or simply as a comparison between the application ontologies. In this example, the *alice* can export any *unconstrained* activity description to *bob* and *bob* can export any *context_free* activity description to *alice*; however, *alice* cannot export *markov_precond* activity descriptions to *bob* and *bob* cannot export any *markov_effects* activity descriptions to *alice*.

Although inferred implicitly during translation, these relationships may be explicitly determined by the simple PROFILE-COMPARE algorithm presented in Figure 6. Explicitly inferring these mappings offers several capabilities. If run-time translation efficiency is important, then these point-to-point mapping rules could be generated upon first interaction and then cached as explicit rules to be used in subsequent interactions. A detailed discussion of such tradeoffs and overlaps between point-to-point and interlingua-based integration approaches is presented in [13].

In addition, by explicitly generating such mappings, it may be possible to use simpler inference engines to perform translation, rather than requiring a full first order reasoner to implicitly translate using axioms of the interlinqua, the semantic mappings, and the input to be translated. Importantly, such explicit mappings may also be used by the application designers to examine the structure of their application as well as to to evaluate relationships and coverage relative to the interlingua or other ontologies.

```
PROFILE-COMPARE($\mathbf{P_a}, \mathbf{P_b}$)
1  for each $C_a \in \mathbf{P_a}$
2  do for each $C_b \in \mathbf{P_b}$
3      do $\{g_a, g_b\} \leftarrow$ CONCEPT-COMPARE($C_a, C_b$)
4          OUTPUT('$g_a \supset (C_a \supset C_b)$')
5          OUTPUT('$g_b \supset (C_b \supset C_a)$')
```

```
CONCEPT-COMPARE($C_a, C_b$)
1  $R_a \leftarrow true; R_b \leftarrow true$
2  for $i \leftarrow 1$ to $m$
3  do $s \leftarrow$ VALUES($C_a, i$) $\cap$ VALUES($C_b, i$)
4      if $s \neq \emptyset$
5        then $R_a \leftarrow$ CONJUNCTION($R_a$, DISJUNCTION($s$))
6             $R_b \leftarrow$ CONJUNCTION($R_b$, DISJUNCTION($s$))
7        else if VALUES($C_a, i$) $\neq \emptyset \wedge$ VALUES($C_b, i$) $\neq \emptyset$
8               then error "No mapping."
9  return $\{R_a, R_b\}$
```

FIGURE 6. The PROFILE-COMPARE algorithm for determining relationships between ontologies, given the semantic integration profiles.

6. OPEN PROBLEMS

Several important issues related to semantic integration have not been addressed so far in this work, including:

- **Translation Definitions for Primitive Relations**

  All of the translation definitions generated by the Twenty Questions tool are restricted to semantic mappings using only the definitional extensions of the PSL Ontology; they do not provide general semantic mappings between concepts within the core theories of the ontology.

  Translation definitions are also restricted to mappings between the classes of the application ontology and the PSL Ontology; they do not map relations in the different ontologies. For example, different applications may impose restrictions on the *subactivity* relation in the composition of complex activities—in one ontology, the relation may not be transitive, while in the other ontology, the relation may be isomorphic to a bipartite graph consisting of primitive and nonprimitive activities. Even though both of these relations are definably interpretable within the PSL Ontology, the mappings do not use invariants, and there is no general way of generating a direct mapping between the two ontologies.

  This leads to the following question:

  Under what conditions does the existence of a semantic integration profile guarantee the existence of a definable interpretation of primitive relations with respect to the invariants in the profile?

- **Incomplete Sets of Invariants**

  The approach to semantic integration taken in this paper relies on the existence of a complete set of invariants for the models of the ontology. However, there are theories (e.g. graphs) for which such a set of invariants cannot be found. In such cases, two concepts may have equivalent semantic integration profiles (i.e., equivalent values for the invariants) yet not have isomorphic intended models.

In some cases, this may require the introduction of new core theories to axiomatize the intended models of the concepts. For example, a theory of resource requirements would be required to distinguish between different classes of manufacturing and logistics activities. Of course, this does not eliminate the problem if the models of the new core theories also do not have complete sets of invariants.

Given a theory whose models cannot be completely classified by some set of invariants, how can the translation definitions be augmented by more general relative interpretation axioms?

- **Recognizing Classes from Domain Theories**

  The PSL Ontology makes a distinction between the axioms of the ontology and the axioms of a domain theory that uses the ontology, which are characterized as syntactic classes of sentences that are satisfied elements of the models. For example, traditional precondition axioms are characterized as the class of sentences that are satisfied by $markov\_precond$ activities, and traditional effect axioms are equivalent to the class of sentences that are satisfied by $markov\_effect$ activities. On the other hand, many process ontologies used by software applications do not explicitly specify classes of activities, but only specify syntactic classes of process descriptions.

  Is it always possible to automatically determine the profile for a class using only the domain theory associated with elements of the class?

7. Conclusions

This paper has described how model-theoretic invariants of an ontology can be used to specify semantic mappings translation definitions between application ontologies and an interlingua. In particular, examples have been presented using the Process Specification Language (PSL) ontology as the neutral medium in integration.

The sets of models for the core theories of PSL are partitioned into equivalence classes defined with respect to the invariants of the models. Each equivalence class in the classification of PSL models is axiomatized using a definitional extension of PSL. The Twenty Questions tool that is based on these invariants and definitional extensions supports semi-automatic generation of semantic mappings between an application ontology and the PSL Ontology.

This approach can be generalized to other ontologies by specifying the invariants for the models of the axiomatizations. Future work in this area includes developing software to generate mappings based on profiles created with the Twenty Questions tool and application to translation between PSL and other ontologies (such as OWL-S) and translators for existing process modelers and schedulers.

Acknowledgements: This work was supported by the Precision Engineering Project within the Manufacturing Engineering Laboratory at the National Institute of Standards and Technology (NIST); Office of Naval Research (ONR) Grant N00014-01-1-0618; and NIST Grant #70NAN33H1026, funded by the National Science Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and not necessarily the supporting organizations.

REFERENCES

[1] Bouquet, P., Serafini, L., Zanobini, S., and Benerecetti, M. (2003) An Algorithm for Semantic Coordination. *Semantic Integration Workshop, International Semantic Web Conference 2003*, 21–26.

[2] Ciocoiu, M., Gruninger M., and Nau, D. (2001) Ontologies for integrating engineering applications, *Journal of Computing and Information Science in Engineering*, 1:45-60.

[3] Ciocoiu, M. 2002 *Ontology-based Semantics*, Ph.D. thesis, Department of Computer Science, University of Maryland, College Park.

[4] Duschka, O.M. and Genesereth, M.R. 1997. Infomaster - an information integration tool. In *Proceedings of the International Workshop on Intelligent Information Integration*. Freiburg, Germany.

[5] Gruninger, M. (2003) A Guide to the Ontology of the Process Specification Language, in *Handbook on Ontologies in Information Systems*, R. Studer and S. Staab (eds.). Springer-Verlag.

[6] Gruninger, M. and Menzel, C. (2003) Process Specification Language: Principles and Applications, *AI Magazine*, 24:63-74.

[7] Gruninger, M. and Uschold, M. (2003) Ontologies and Semantic Integration, in *Software Agents for the Warfighter*, Information Technology Assessment Consortium, to appear.

[8] Mahdavan, J., Bernstein, P., and Rahm, E. (2001) Generic Schema Matching with Cupid, *Proc. 27th VLDB Conference*, 49–58.

[9] Marker, D. (2000) *Model Theory: An Introduction*. Springer-Verlag.

[10] 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.

[11] Noy, N. and Musen, M. (2000) PROMPT: Algorithm and tool for automated ontology merging and alignment, Proceedings of AAAI-2000.

[12] Stuckenschmidt, H. and Visser, U. (2000) Semantic Translation Based on Approximate Reclassification. In Proceedings of the Seventh International Conference on Knowledge Representation and Reasoning, Breckenridge, Colorado.

[13] Uschold, M., Jasper, R. and Clark, P. 1999. Three Approaches for Knowledge Sharing: A Comparative Analysis. In *Proceedings of the Twelfth Workshop on Knowledge Acquisition, Modeling, and Management (KAW'99)*.

MANUFACTURING SYSTEMS INTEGRATION DIVISION, NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY, 100 BUREAU DRIVE, GAITHERSBURG, MD 20899-8260, `gruning@cme.nist.gov`

GEOMETRIC AND INTELLIGENT COMPUTING LABORATORY, DEPT. OF COMPUTER SCIENCE, DREXEL UNIVERSITY, 3141 CHESTNUT STREET, PHILADELPHIA, PA 19104, `tjkopena@cs.drexel.edu`