Semantic Integration through Invariants

Michael Grüninger
Manufacturing Systems Integration Division
National Institute of Standards and Technology
100 Bureau Drive
Gaithersburg, MD 20899-8260
gruning@cme.nist.gov

Joseph B. Kopena
Geometric and Intelligent Computing Laboratory
College of Engineering, Drexel University
3141 Chestnut St
Philadelphia, PA 19104
joe@plan.mcs.drexel.edu

Introduction

A semantics-preserving exchange of information requires mappings between logically equivalent concepts in each ontology. 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 ((5), (7)) emphasize the use of generic techniques that do not exploit the model-theoretic structures of the ontologies. In this paper we will show how the classification of models within the PSL Ontology can serve as the basis for generating semantic mappings between applications.

The Process Specification Language (PSL) ((2), (4), (6)) has been designed to facilitate correct and complete exchange of process information among manufacturing systems[1], such as scheduling, process modeling, process planning, production planning, simulation, project management, workflow, and business process reengineering. PSL is intended to be used as a mediating ontology that is independent of the applications' ontologies and that is used as a neutral interchange ontology ((1)). The semantic mappings between application ontologies and PSL can be semi-automatically generated from invariants (properties of models preserved by isomorphism). Since these invariants are also used to characterize the definitional extensions within the PSL Ontology, the semantic mappings can be verified prior to integration.

PSL Ontology

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].

Copyright © 2003, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

[1] PSL has been accepted as International Organisation of Standardisation project ISO 18629; as of June 2003, part of the work is under review as a Draft International Standard.

[2] The complete set of axioms for the PSL Ontology can be found at `http://www.mel.nist.gov/psl/psl-ontology/`.

Core Theories

All core theories within the ontology are consistent extensions of PSL-Core (T_{psl_core}). The purpose of PSL-Core is to axiomatize 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:

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.

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 (flu-

Core theories are indicated by a .th suffix and definitional extensions by a .def suffix. As of June 2003, the ontology is in version 2.0.

ents that always hold after an occurrence).

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.

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.

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, these sets are subtrees of the occurrence trees. 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.*

Definitional Extensions

Many ontologies are specified as taxonomies or class hierarchies, yet few ever give any 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, such as vector spaces, 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.

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. In particular, 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.

Many of the invariants with definitional extensions in the PSL Ontology are related to the automorphism groups[3] for different substructures of the models. For example, we can consider mappings that are permutations of activity occurrences that map the predecessor of a legal occurrence of an activity **a** to other predecessors of legal occurrences of **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 the case of Markovian activities, that is, 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* (see Figure 1). The invariant associated with this extension is the group[4] $\mathcal{P}^{\mathcal{F}}(\mathbf{a})$, which is the maximal normal subgroup of $Aut(\mathcal{F})$ that is also a subgroup of $OP(\mathbf{a})$. If $\mathcal{P}^{\mathcal{F}}(\mathbf{a}) = Aut(\mathcal{F})$, 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 1. If $\mathcal{P}^{\mathcal{F}}(\mathbf{a})$ is only a subgroup of $Aut(\mathcal{F})$, then there exist additional nonmarkovian constraints on the legal occurrences of the activity; this is axiomatized by the *partial_state* class in Figure 1. If $\mathcal{P}^{\mathcal{F}}(\mathbf{a})$ 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 1.

Additional relations 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 $Aut(\mathcal{F})$ 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.

Translation Definitions

Translation definitions specify the mappings between PSL and application ontologies. Such 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.

Translation definitions 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.

[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.

[4] In this example, $\mathcal{F}$ is the structure isomorphic to the extension of the *prior* relation. $Aut(\mathcal{F})$ 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.

$$(\forall o_1, o_2)\ state\_equiv(o_1, o_2) \equiv \quad (1)$$
$$(\forall f)\ (prior(f, o_1) \equiv prior(f, o_2))$$
$$(\forall a, o_1, o_2)\ poss\_equiv(a, o_1, o_2) \equiv \quad (2)$$
$$(poss(a, o_1) \equiv poss(a, o_2))$$
$$(\forall a)\ markov\_precond(a) \equiv \quad (3)$$
$$((\forall o_1, o_2)\ state\_equiv(o_1, o_2) \supset poss\_equiv(a, o_1, o_2))$$
$$(\forall a)\ partial\_state(a) \equiv \quad (4)$$
$$(\exists o_1)\ ((\forall o_2)\ state\_equiv(o_1, o_2) \supset poss\_equiv(a, o_1, o_2))$$
$$\wedge(\exists o_3, o_4)\ state\_equiv(o_3, o_4) \wedge \neg poss\_equiv(a, o_3, o_4)$$
$$(\forall a)\ rigid\_state(a) \equiv \quad (5)$$
$$(\forall o_1)(\exists o_2)\ state\_equiv(o_1, o_2) \wedge \neg poss\_equiv(a, o_1, o_2)$$

Figure 1: Classes of activities with state-based preconditions (from the definitional extension *state_precond.def*).

For example, the concept of *AtomicProcess* in the DAML-S Ontology ((3)) has the following translation definition:

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

This methodology has been implemented in the PSL project's Twenty Questions mapping tool[5]. Each question corresponds to an invariant, and each possible 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.

For example, consider the question displayed in Figure 2. The invariant corresponding to this question is $\mathcal{P}^{\mathcal{F}}(\mathbf{a})$, and the classes of activities corresponding to values of this invariant are axiomatized in Figure 1. 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.

[5]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 2: One of the Twenty Questions, used to classify activities with state-based preconditions.

When building translators, we are faced with the additional challenge that almost no application has an explicitly axiomatized ontology. However, we take the Ontological Stance ((4)), in which we model a software application as if it were an inference system with an axiomatized ontology, and use this ontology to predict the set of sentences that the inference system decides to be satisfiable. The Twenty Questions tool supports this by allowing the application designer to specify the intended semantics of her ontology by using the classes in the PSL Ontology.

Process Information Exchange Profiles

In addition to providing mappings between an application and PSL, we can also use the translation definitions to directly specify the relationship between two application ontologies. For example, suppose we have a scenario in which two software agents, Alice and Bob, need to exchange process information. Alice's designer specifies the semantic mapping (translation definitions) between Alice's ontology and the PSL ontology, and Bob's designer specifies the semantic mapping between Bob's ontology and the PSL ontology. When Alice and Bob first interact, they use these previously specified mappings to automatically generate the semantic mappings between each other's ontologies. In this way, the PSL Ontology mediates the mapping between the agent ontologies.

The set of translation definitions for all concepts in a software application's ontology is the profile for the application. If the PSL Ontology has m invariants and each invariant n values, then an application profile will have the form:

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

$$(\forall a)\, C_k^{onto}(a) \equiv$$

$$(p_{11}(a) \vee ... \vee p_{1n}(a)) \wedge ... \wedge (p_{m1}(a) \vee ... \vee p_{mn}(a))$$

For example, we may have:

$$(\forall a)\, C_1^{alice}(a) \equiv unconstrained(a)$$

$$\wedge(markov\_effects(a) \vee context\_free(a))$$

$$(\forall a)\, C_1^{bob}(a) \equiv$$

$$(unconstrained(a) \vee markov\_precond(a)) \wedge context\_free(a)$$

In general, we want to use PSL and the profiles to determine the relationship between the application ontologies. The mapping for the above example would be:

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

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

These mappings will in general take the form of:

$$(\forall a)\, (p_{11}(a) \vee ... \vee p_{1n}(a)) \wedge ... \wedge (p_{m1}(a) \vee ... \vee p_{mn}(a))$$

$$\supset (C_i^{alice}(a) \supset C_j^{bob}(a)))$$

The antecedents of these sentences can be considered to be guard conditions that determine which activities can be shared between Alice and Bob. This can either be used to support direct exchange between Alice and Bob, or simply as a comparison between the application ontologies for Alice and Bob. In the example, 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.

Summary

In this paper we have described how the use of model-theoretic invariants can be used to specify translation definitions between application ontologies and PSL. 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 the semiautomatic 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 DAML-S) and translators for existing process modelers and schedulers.

References

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

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.

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.

Menzel, C. and Gruninger, M. (2001) A formal foundation for process modeling, *Second International Conference on Formal Ontologies in Information Systems*, Welty and Smith (eds), 256-269.

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

Schlenoff, C., Gruninger, M., Ciocoiu, M., (1999) The Essence of the Process Specification Language, *Transactions of the Society for Computer Simulation* vol.16 no.4 (December 1999) pages 204-216.

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.