A modified copy of this paper has appeared in
Journal of Computing and Information Science in Engineering 1(1): 12-22.

Ontologies for Integrating Engineering Applications

Mihai Ciocoiu
Department of Computer Science
and Institute for Systems Research
University of Maryland
College Park, MD 20742
mihaic@cs.umd.edu
(not an ASME member)

Michael Gruninger
Institute for Systems Research
University of Maryland
College Park, MD 20742
gruning@cme.nist.gov
(not an ASME member)

Dana S. Nau
Department of Computer Science
and Institute for Systems Research
University of Maryland
College Park, MD 20742
nau@cs.umd.edu
(ASME Member)

November 28, 2000

Abstract

In all types of communication, the ability to share information is often hindered because the meaning of information can be drastically affected by the context in which it is viewed and interpreted. This is especially true in manufacturing, because of the growing complexity of manufacturing information and the increasing need to exchange this information among various software applications. Different representations of the same information may be based on different assumptions about the world, and use differing concepts and terminology – and conversely, the same terms may be used in different contexts to mean different things. Often, the loosely defined natural-language definitions associated with the terms will be too ambiguous to make the differences evident, or will not provide enough information to resolve the differences.

A solution to this problem is the use of taxonomies or ontologies of manufacturing concepts and terms, because ontologies provide a way to make explicit the semantics (i.e., the meaning) for the concepts used, rather than relying just on the syntax used to encode those concepts. Ontological techniques can be useful for giving unambiguous definitions of product and process capabilities and evolving designs and design requirements, unifying the differences in how knowledge is conceptualized across multiple product and process domains, and translating those definitions into the specialized representation languages of application systems.

This paper gives an overview of current research and development on ontologies as it relates to mechanical engineering applications, along with examples of how ontologies can be used to facilitate the exchange of information among manufacturing applications.

1 Introduction

One of the major problems facing enterprises today is the lack of interoperability among the various software applications that the enterprise uses. This problem is most acute for systems that must manage the heterogeneity inherent in various domains and integrate models of different domains into coherent frameworks—for example, in business process reengineering, where enterprise models integrate its processes, organizations, goals and customers, in distributed multiagent architectures, and in concurrent engineering and design.

Two of the primary challenges in achieving interoperability include the need to resolve semantic clashes, and the work involved in developing multiple translators:

Semantic clashes. In order to integrate two software applications, substantial difficulties can arise in translating information from one application to the other, because the applications may use different terminology and representations of the domain. Even when applications use the same terminology, they often associate different semantics with the terms. This clash over the meaning of the terms prevents the seamless exchange of information among the applications. What is needed is some way of explicitly specifying the terminology of the applications in an unambiguous fashion.

Multiple translators. Even if one can resolve the problem of integrating two specific applications, it may take a lot of work to do so—and this problem is compounded when one wants to integrate a set of more than two applications. Typically, point-to-point translation programs are written to enable communication from one specific application to another. However, as the number of applications has increased and the information has become more complex, it has been more difficult for software developers to provide translators between every pair of applications that must cooperate. What is needed is some way of explicitly specifying the terminology of the applications in an unambiguous fashion.

For example, as shown in Figure 1, a concurrent engineering project may require the exchange of information among systems for CAD, performance analysis, manufacturability analysis, product data management system, process planner, production management system, scheduler, and a simulation system. Current approaches for integrating such systems would require writing point-to-point translators for each pair of systems that need to share information. Furthermore, for each such pair of systems, exchanging information between them could require the resolution of semantic clashes.

To address these challenges, various groups within industry, academia, and government have been developing sharable and reusable models known as ontologies. All ontologies consist of a vocabulary along with some specification of the meaning or semantics of the terminology within the vocabulary. In doing so, ontologies support interoperability by providing a common vocabulary with a shared machine-interpretable semantics. Rather than develop point-to-point translators for every pair of applications as shown in Figure 1, one only needs to write translators between the terminology used in each application and the common ontology, as shown in Figure 2.

This paper will give an overview of several ways in which ontologies can be used to support the integration of engineering applications. We will survey existing ontologies that have been developed, particularly for enterprise modeling. We will then consider in detail the application of ontologies to support interoperability of manufacturing process applications.

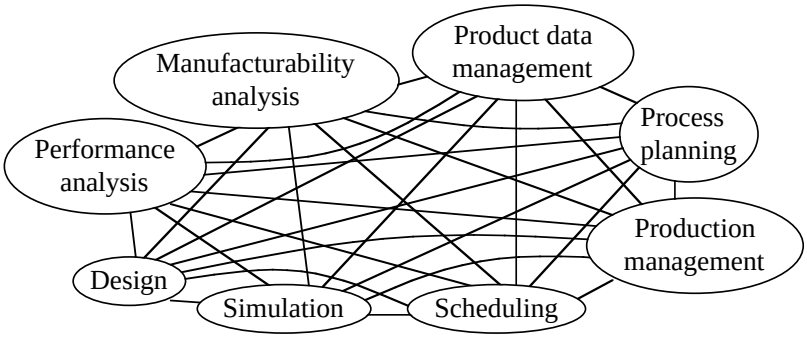

Figure 1: Information exchange among subsystems for concurrent engineering.

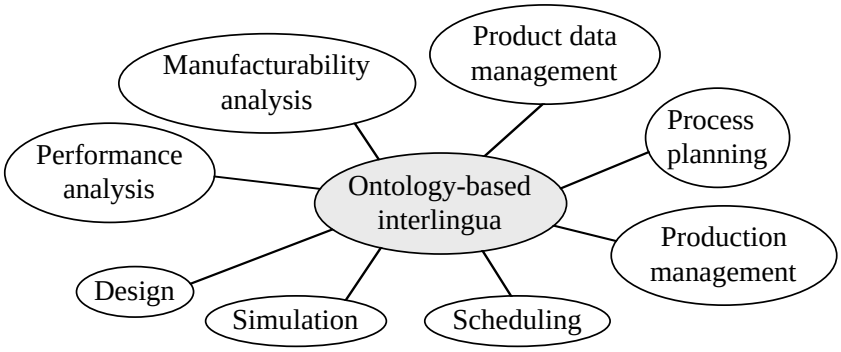

Figure 2: A common ontology drastically reduces the number of translators needed.

2 The Need for Semantics

Consider the two process planning applications in Figure 3. Both applications use the term *resource*, but in each application, this term has a different meaning: in one, it refers to the machine to be used to do a manufacturing operation, while in the other it refers to the piece of stock on which the operation is to be done. Similarly, the terms *material* in Application A and *workpiece* in Application B both mean some object which has been produced by one activity and consumed by another activity, i.e. work-in-process; however, these meanings are not evident from the terminology alone. Thus, simply sharing terminology is insufficient to support interoperability – in order to map the concepts in one application to the concepts in the other application, one needs to know the semantics of the terms in each application.

Ambiguity may be present even when two systems agree on the set of terms that they are using. Two systems may agree that purchase orders have customer name, product name, quantity, and due date. However, the two different systems may disagree about what these terms mean. For example, the supplier may interpret due date as the date at which its production must be complete

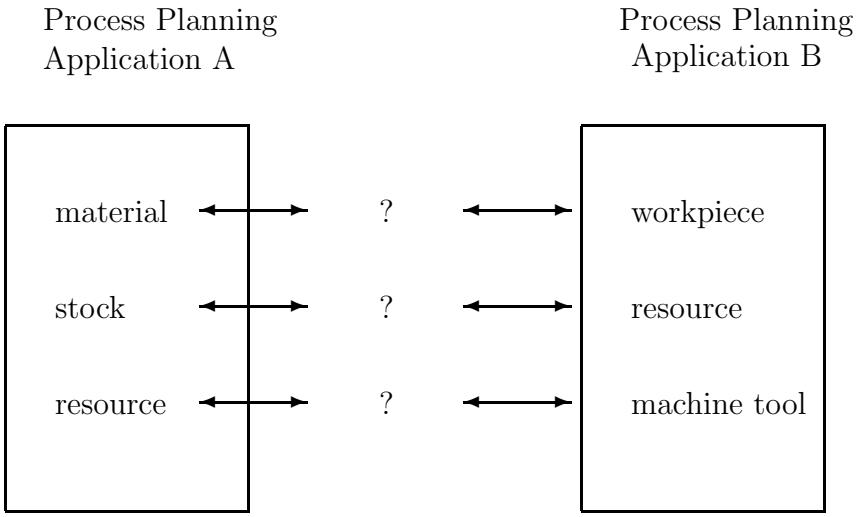

Figure 3: In this example, interoperability cannot be achieved simply by sharing terminology, because the term "resource" has different meanings in each application. Interoperability will require translation to be based on explicit definitions of the terms.

and the product is ready to ship, while the customer may interpret due date as being the date on which the product is received. Since in reality these two dates can be very different, supply chain problems will arise because of the ambiguity in the interpretations.

We therefore need some way of unambiguously specifying the semantics of the terminology of an application. Further, this specification must be computer-interpretable. Any translator requires the specification of semantics in some form; however, we are interested in automated translators, and if the semantics are not completely computer-interpretable, then humans will need to assist in the translation. In this section, we review the use of first-order logic as a means of defining such a specification, followed by an overview of how first-order logic can be used to specify the mappings among applications that preserve their semantics.

2.1 Ontologies

In order to be able to express concepts needed in some engineering application in a way that is effective for facilitating translation of those concepts, these concepts must be expressed in some language that is highly expressive yet free from the problems of imprecision and ambiguity associated with natural language.

Although there are several different views of what an ontology is, most researchers using ontologies for enabling interoperability would agree with Thomas Gruber's definition [1]:

> An *ontology* is an explicit specification of a conceptualization. The term is borrowed from philosophy, where an Ontology is a systematic account of Existence. For AI systems, what "exists" is that which can be represented. When the knowledge of a domain is represented in a declarative formalism, the set of objects that can be represented is called the universe of discourse. This set of objects, and the describable relationships among them, are reflected in the representational vocabulary with which

> a knowledge-based program represents knowledge. Thus, in the context of AI, we can describe the ontology of a program by defining a set of representational terms. In such an ontology, definitions associate the names of entities in the universe of discourse (e.g., classes, relations, functions or other objects) with human-readable text describing what the names mean, and formal axioms that constrain the interpretation and well-formed use of these terms. Formally, an ontology is the statement of a logical theory.

Although any ontology can constrain the meanings of terms, the various ontologies that have been developed can be distinguished by their degree of formality in the specification of meaning. Informal and semi-formal ontologies use natural language to either specify the semantics of the terminology; they can serve as a framework for shared understanding among people, but they are often insufficient to support interoperability, since any ambiguity can lead to inconsistent interpretations and hence hinder integration. Further, since the semantics of informal and semi-formal ontologies are not completely machine-interpretable, they do not support automated translation.

We will therefore focus on ontologies that provide an *axiomatic characterization* for some set of terms, i.e., a set of axioms that set up some constraints on the meaning of those terms so that they will correspond to some set of concepts that the user wants to express (for more detail, see [1]). How strong a set of constraints one might want to use in order to specify the semantics of the ontology may depend on what one is trying to accomplish:

- At one extreme, an ontology may simply be a *taxonomic hierarchy of classes* such as the one shown in Figure 4(a), without any constraints on the meaning of those terms other than axioms that specify the class-subclass relationships that those terms must satisfy (Figure 4(b) gives an example of how these axioms might be written). Thus, such an ontology may have many different models corresponding to many different possible worlds, some of which the user might not have even intended to model.

- At the other extreme, an ontology may be a much stronger set of axioms (as included in Cyc [2]) intended to set up detailed definitions of the properties of the world. Such an ontology would include other axioms in addition to the ones shown in Figure 4(b). These axioms would further constrain what the terms mean, and allow us to infer various additional properties of the world. Cyc's general ontology for commonsense knowledge contains more than 10,000 concept types that are organized into a hierarchy of concepts.

- An intermediate approach is to write an ontology that allows definitions, but only *conservative* ones [3], i.e., definitions that introduce terminology but do not add any constraints about what possible worlds might be models of the theory.

Most axiomatic approaches to ontologies use first-order logic [1] as the formal language for specification.

A first-order theory consists of a *first-order language*, a set of *axioms*, and a set of *rules of inference*. There are some restrictions on what the language, axioms, and rules of inference can be like, but we will not describe those restrictions here. The language is used to represent statements about the world, the axioms are the statements which we take to be "basic truths" about the world, and the rules of inference are used to derive consequences from the axioms.

[1] See [3] for a detailed description of first-order logic.

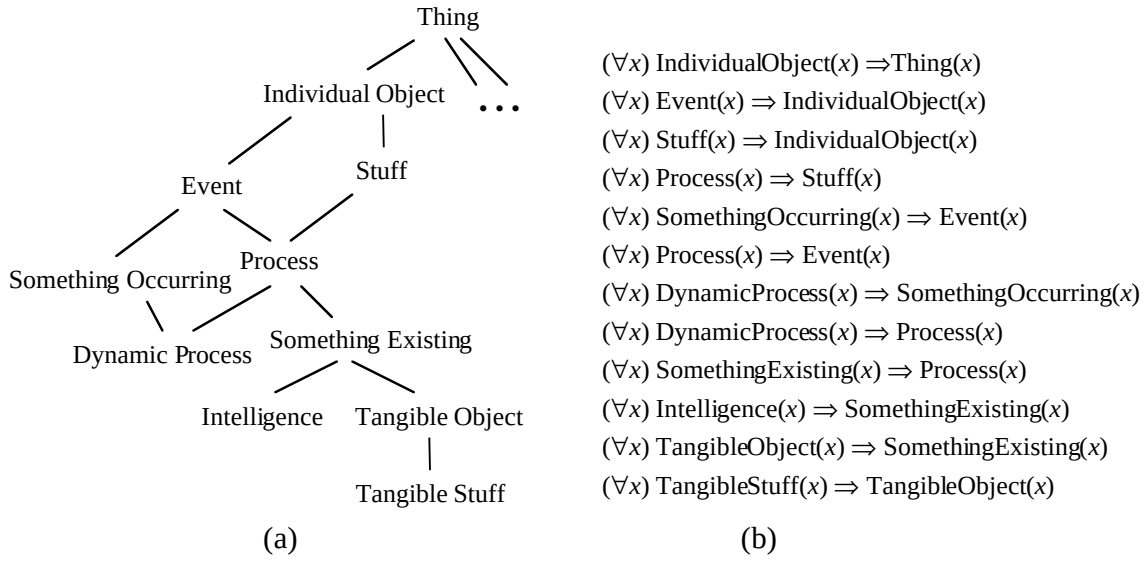

Figure 4: Part (a) is a taxonomic hierarchy of some of the categories [4] from an early version of Cyc. Part (b) shows a possible set of axioms one might use to represent this taxonomic hierarchy.

KIF [5], the "Knowledge Interchange Format" is a particular version of first-order logic, proposed as a standard for writing descriptions (ontologies) and specifically designed to make it useful as an interlingua between computer applications. Rather than using the standard mathematical notation for first-order logic, it has a simple list-based linear ASCII syntax, which facilitates computer interpretability. For example the statement that "all machines are resources", which might be represented in logic as

$$(\forall x)\,\text{machine}(x) \rightarrow \text{resource}(x),$$

is written in KIF as

```
(forall (?x) (=> (machine ?x) (resource ?x))).
```

KIF provides a built-in ontology for numbers, sets and lists, as well as for expressing knowledge about the properties of functions and relations. It also includes sublanguages for defining objects, relations and functions, for expressing knowledge about knowledge as well as for defining both monotonic and non-monotonic inference rules.

The semantics of a first-order theory rest on two key ideas:

Compositional truth assignment. There is some notion of truth – all sentences have a truth assignment of True or False. Sentences can be combined using connectives (such as and, or, not, implication. biconditional) and the truth assignment of the combination is determined by the truth assignments of its constituents. Similarly, we can construct sentences with quantifiers (forall and exists), and the truth assignment of such sentences depends on the syntactic form of the sentence and some domain of elements.

Model theory. A model of a theory is a set of elements (the domain) together with some truth assignment that satisfies all sentences in the theory. The truth assignment is typically specified by some structure over the set of elements in the domain. Examples of this structure include partial ordering, lattices, or vector spaces.

A crucial property of first-order logic is that it is sound and complete. Soundness is the property that guarantees that any sentence that is derivable/provable from a theory (set of sentences) is true in all models of the theory. Completeness is the converse – any sentence that is true in all models of the theory is provable from the theory. These properties are also equivalent to saying that a sentence is consistent with a theory is true in some model of the theory. We can therefore show that a theory is consistent by specifying some structure and proving that it indeed satisfies all sentences in the theory. Thus, we can be guaranteed that any inference that is done with a first-order theory preserves the semantics of the terminology of the theory.

2.2 Ontologies and Semantics

Ontologies support interoperability by enabling the specification of semantics-preserving mappings between the terminology of different applications. To understand what it means for semantics to be preserved by mappings, we must first consider the models of the axiomatization of the terminology of each application. The intuition is that a mapping from a term in one application to a term in a second application preserves the semantics if the models of the axioms for each term have a similar structure.[2] In the simplest case, the two terms have definitions that are logically equivalent. More generally, it may be necessary to specify additional constraints that one application must satisfy. For example, one scheduling application may be able to schedule nondeterministic process plans, while another may only be able to schedule deterministic process plans, even though both use the same term "process-plan"; in such a case, the mapping between the applications would need to include this constraint in the definitions in order to preserve the semantics.

To see how semantics-preserving mappings support interoperability, consider the problem from Figure 3. If "stock" in Application A and "resource" in Application B have logically equivalent definitions that state they are consumable with respect to some activity, then the mapping of these two terms preserves their semantics. Similarly, if "resource" in Application A and "machine tool" in Application B have logically equivalent definitions stating that they are reusable with respect to some activity, then the mapping of these two terms preserves their semantics. Figure 5 illustrates the KIF definitions of the terms from each application. With these definitions, interoperability can be achieved by translating process plans between the applications using the common definitions.

3 Ontologies and Interoperability

The primary concern of this paper is the use of ontologies for facilitating interoperability. Typical scenarios that require interoperability include the integration of legacy information systems, concurrent engineering, iterative process design, supply chain management, and business-to-business electronic commerce. In each of these scenarios, multiple independent teams and applications must seamlessly share information.

[2] An elaboration of this notion of similarity is discussed in [6].

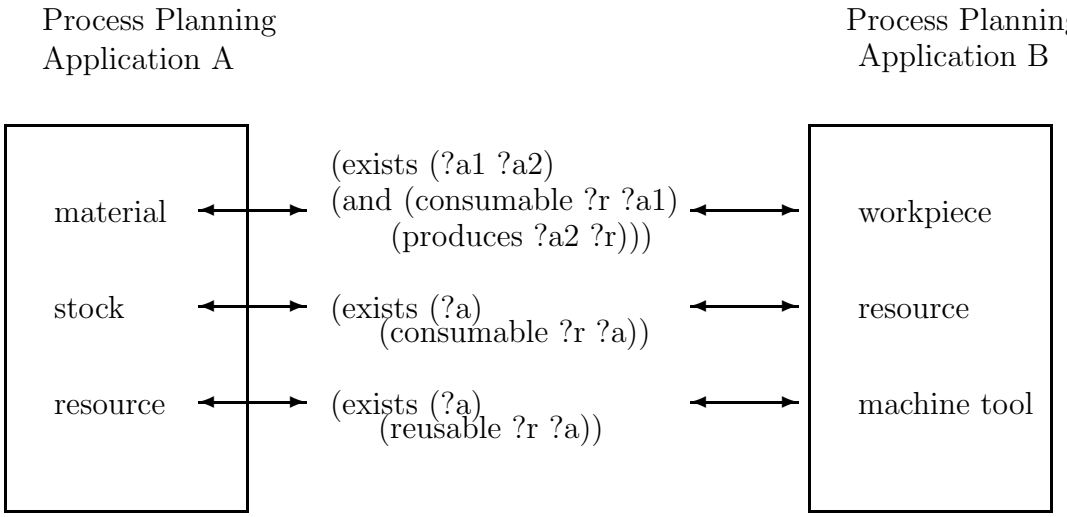

Figure 5: By explicitly providing definitions, the terminology of one application can be mapped to the terminology of the other application so that semantics are preserved.

For example, consider concurrent engineering. A design engineer produces a product specification using a CAD system. This design must be interpreted by systems for performance analysis and manufacturability analysis, and it must be integrated with the company's product data management system, which will represent not only the product's features and geometry, but may also include notions such as design intent, other product requirements, and a bill-of-materials decomposition. This design must also be shared with the process design team, who uses the bill-of-materials to specify a set of manufacturing processes that can produce a product with the desired features. Any version of the process design may be shared with the process planning team, who specify the various machines, tools, and materials that will be required by the manufacturing processes. If any problems arise with these processes, they must be communicated to the product designer, who may need to modify the design to guarantee manufacturability. The production planning team will need to share the process plan, since it must be included within the production plan, together with the process plans of other products. Schedulers take the production plan and add further constraints on the occurrence of various processes. If either the production planner or scheduler discover a problem (such as unanticipated bottleneck resources), the underlying process plan or production plan may need to be revised by earlier teams.

The problem in this scenario is that each of the teams will be using different software applications to represent and reason about the products and processes from their particular perspectives.

We will investigate two basic classes of solutions to the problem of enabling all these software applications to communicate. One is a *standardization approach* that has all applications share a common ontology, while the other uses the ontology as an *interlingua* and requires that translators are written to/from each software application.

3.1 Sharing a Common Ontology

When applied for solving the problem of interoperability among heterogeneous systems, an important question is, to what extent do we want to take the information used by these heterogeneous

systems and represent it as part of a single common ontology? At one extreme, the *standardization approach* is to use a single common ontology to represent the information used by all of the different systems. The opposite extreme is to use a different ontology within each system and to have a network of mediators and facilitators that enable translation among these different ontologies. Other intermediate approaches can also be devised. In this section we consider the use of single common ontologies for modelling enterprises.

Enterprise modelling ontologies are distinguished by their scope and the central role of integrating multiple ontologies. An enterprise model is a computational representation of the structure, activities, processes, information, resources, people, behaviour, goals and constraints of a business, government, or other enterprise. It can be both descriptive and definitional – spanning what is and what should be. The role of an enterprise model is to achieve model-driven enterprise design, analysis and operation. Enterprise modelling ontologies must therefore be able to represent concepts in the domains of activities, time, resources, products, services, organization, goals, and policies. Further, these must be integrated in order to support reasoning that requires the use of multiple ontologies, and to support interoperability among tools using different ontologies. For example, the notion of manufacturability requires reasoning about the product properties, preconditions and effects of activities, and the capabilities of resources. In this section, we will first consider the role that ontologies can play in this integration, and then examine in more detail several enterprise modelling ontologies that have been developed and applied to industrial problems.

3.1.1 TOVE

The TOVE (TOronto Virtual Enterprise) project [7] created an integrated suite of ontologies to support enterprise engineering. Since this must be a shared terminology for the enterprise that every application can jointly understand and use, the ontologies span knowledge of activity, time, and causality ([8, 9]), resources [10], cost [11], quality [12], , organization structure [13], product [14] and agility [15]. All of the TOVE ontologies are specified using KIF; for example, the following KIF sentence is the definition of the class of processor actions within the TOVE Resource Ontology:

```
(defrelation processor_action (?a)
  (exists (?r1 ?r2 ?r3)
          (and    (uses ?a ?r1)
                  (or     (consumes ?a ?r2)
                          (modifies ?a ?r2))
                  (or     (produces ?a ?r3)
                          (modifies ?a ?r3)))))
```

What this definition says is that a processor action is one in which there exist at least three objects such that one is a resource that is used by the activity and can be reused later, another object is a resource is either consumed or modified by the action, and the third object is either produced or modified by the action. Intuitively, the resource that is used by the action will be a machine, the resource that is consumed will be the input material, and the remaining object will be the output material of the action. Any action satisfying these conditions will be a processor action.

In addition to being formally defined using KIF, the TOVE ontologies have been implemented using the Prolog programming language [16]. By using Prolog's built-in facilities for logical inference, systems that use TOVE can automatically deduce from the axioms of the ontologies the answers to many "common sense" questions about the enterprise.

The TOVE ontologies were developed in cooperation with several companies and have been applied to the design and analysis of enterprise models within supply chain management, project management, and business process engineering. In particular, [15] discusses the application of the TOVE ontologies to the analysis of customer relationship management processes within IBM Canada. In other work, the ontologies were used to model the supply chain of BHP Steel (Australia) and assist in the construction of management scenarios.

3.1.2 Enterprise Ontology

The Enterprise Project at the University of Edinburgh [17] supports an environment for integrating methods and tools for capturing and analyzing key aspects of an enterprise, based on an ontology for enterprise modelling. This ontology (the Enterprise Ontology) has five top-level classes for integrating the various aspects of an enterprise:

- Meta-Ontology: Entity, Relationship, Role, Actor, State of Affairs
- Activities and Processes: Activity, Resource, Plan, Capability
- Organisation: Organisational Unit, Legal Entity, Manage, Ownership
- Strategy: Purpose, Strategy, Help Achieve, Assumption
- Marketing: Sale, Product, Vendor, Customer, Market

The Enterprise Ontology is semi-formal – it provides a glossary of terms expressed in a restricted and structured form of natural language supplemented with a few formal axioms using Ontolingua. For example, the following expression is the specification of the concept of "Plan" within the Enterprise Ontology:

```
(define-frame Plan
              :own-slots
              ((Documentation
              "The Activity-Spec in the Intended-Purpose Relationship")
               (Instance-Of Class) (Subclass-Of Activity-Spec)
              :template-slots
              ((Intended-Purpose (Minimum-Slot-Cardinality 0)
                                 (Slot-Cardinality 1)
                                 (Slot-Value-Type State-Of-Affairs)) )
              :axioms
              (<=> (Plan ?plan)
                   (exists (?soa)
                           (Intended-Purpose ?plan ?soa)))
              :issues
             ("This definition is equivalent to: An Activity-Spec that
               is associated with an Intended-Purpose."
              "This is a special Role-class.") )
```

In other words, a plan is an instance of the class of activity specifications, and is always associated with an intended purpose. The KIF axiom states that for every plan there exists a state-of-affairs that is the intended purpose of the plan.

Lloyd's Register has used the Enterprise Ontology for more effective modelling and re-engineering of business processes for strategic planning. IBM UK intends to exploit the Enterprise Ontology in modelling its own internal organisation as well as providing technical input via its Business Modelling Method BSDM (Business Systems Development Method).

3.1.3 IDEF Ontologies

The ontologies developed at KBSI are intended to provide a rigorous foundation for the reuse and integration of enterprise models [18]. Thus the ontologies play two important roles – providing a neutral medium for integrating modelling tools within a software environment, and a framework within which to interpret individual enterprise models, to draw logical connections between models, and to detect inconsistencies when integrating models.

This emphasis on semantic integration requires a formal axiomatisation of the classes and relations within an enterprise model. The ontology is a first-order theory consisting of a set of foundational theories along with a set of enterprise models which are extensions to these theories for some specific set of logical constants that specify the entities within the enterprise model. The approach has been used to provide axiomatisations of IDEF0, a method for modeling the decisions, actions, and activities of an organization or system [19], and IDEF1X, a method for designing relational databases [20].

3.2 Ontologies as Interlingua

While the single ontology approach works quite well and is straightforward to use when designing new systems from scratch, its applicability is pretty much restricted to that case. On the other hand, the multiple local ontologies case is generally applicable, but very difficult to implement in practice.

An interesting approach, situated somewhere in-between the aforementioned cases is the *Interlingua* approach. This approach tries to overcome the applicability problems of the single ontology approach, while keeping the translation problem's complexity at a manageable level. The idea is to have a shared ontology and use it as an Interlingua for translating between the communicating systems' local ontologies. Since it allows the communicating systems to have their own, local ontologies, this approach is applicable to systems that were build without any prior intent for them to communicate.

In current practice, this requires the implementation of point-to-point translators between every pair of applications as shown in Figure 1, so that $N(N-1)$ translators are required for N applications. Further, if any new application is introduced, N new point-to-point translators need to be developed.

The need for point-to-point translators arises from the fact that the applications do not share the terminology or a common semantics for any terminology that is shared. An ontology addresses this issue by providing both the terminology and semantics that all applications can share. The common ontology simplifies the problem of writing translators between diferent engineering applications, by reducing the problem of writing $N(N-1)$ point-to-point translators to that of writing N translators (one from each application's terminology to the common ontology) as shown in Figure 2.

However, this approach also requires that the systems' ontologies be described relative to the shared ontology, by writing *semantic definitions* (sometimes called *translation rules*, *compilation rules*, etc) By relating concepts from the communicating systems' local ontologies to their definitions in the common ontology, such semantic definitions make the translation problem computationally tractable.

In the pages that follow we will briefly introduce two projects that use this approach and focuse on building extensible ontologies for process representation, intended to be used as interlinguas for translating information ranging from manufacturing activities to higher-level business processes. We will then give an in-depth example of using the latter ontology as an interlingua for enabling manufacturing process information exchange between two different applications.

3.2.1 PIF

The goal of the Process Interchange Format (PIF) project [21] was the development of an interchange format to support the automatic exchange of process descriptions among a wide variety of business process modelling and support systems, such as workflow tools, process simulation systems, business process reengineering tools, and process repositories. Rather than develop ad hoc translators for each pair of process descriptions, PIF serves as the common format for all systems in order to support interoperability.

PIF is a formal ontology which is structured as a core ontology plus a set of extensions known as partially shared views (PSVs). The intuition is that all systems agree on the definitions of terms within the core, but will agree on the definitions for other terms if they are defined in common PSVs.

The PIF-Core ontology consists of the classes and relations used to describe the basic elements of any process. The top-level of the ontology defines the classes *activity, object, agent*, and *timepoint*, along with the relations *performs* (over agents and activities), *uses, creates, modifies* (all three over activities and objects), *before* (over timepoints), and *successor* (over activities).

The PIF Ontology was specified using the Frame Ontology. An example of a process description using PIF comes from [22]. The first sample frame illustrates the compositional relationships of this activity specification. We can also see that this particular process, "replenish inventory" is in fact a specialization of a more generic activity which relates to "receiving inputs."

```
(define-frame REPLENISH-INVENTORY
              :own-slots
              ((Instance-Of RECEIVE-INPUTS)
               (Components RECEIVE-ORDER, GATHER-ORDER-DETAILS,
                CHECK-ORDER-DETAILS, DETECT-CONDITION, REQUEST-PREPARE-PAYMENT,
                SEND-ORDER, TAKE-DELIVERY,REQUEST-RELEASE-PAYMENT)
               (Name "Replenish Inventory")
               (Documentation "Inventory Replenishment is trigerred at the
                Retailer based on a balance between sales volume and inventory")))
```

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)))
```

Figure 6: The KIF axioms from PSL-Core that axiomatize intuitions about time.

The temporal relationships between these activites is expressed via a series of "successor" frames:

```
(define-frame SUCC-1
              :own-slots
              ((Instance-Of SUCCESSOR)
               (Preceding-Activity RECEIVE-ORDER)
               (Succeeding-Activity GATHER-ORDER-DETAILS)))
```

The relationships that these activities have to other PSV elements, such as the "actor", or to those PSV objects it "uses", etc. would be expressed in other frames included in the process description.

3.2.2 PSL

The goal of the NIST Process Specification Language (PSL) project ([23, 24, 25]) is to create a process specification language to facilitate complete and correct exchange of process information among manufacturing applications. Included in these applications are scheduling, process planning, simulation, project management, workflow, business process reengineering, and product realization process modeling. Although primarily an ontology for processes, it lays the foundations for integration of ontologies required for enterprise modelling by adopting the following structure:

The PSL Ontology has three major components – the axioms of PSL-Core, core theory extensions, and definitional extensions. All concepts within PSL are specified using KIF.

PSL-Core is used to specify the semantics of the primitives in the PSL Ontology. Primitives are those terms which do not have conservative definitions; rather, there are sentences which constrain the interpretation of the terms. All axioms and definitions within PSL are specified in KIF. PSL-Core is an adaptation of the concepts in PIF-Core, so that there are four basic disjoint classes

and four basic relations in the ontology of PSL-Core. The classes are *activity, activity-occurrence, timepoint*, and *object*, the relations *participates-in, before*, and the functions *beginof, endof*. PSL-Core itself consists of sixteen KIF axioms; Figure 6 shows some of the axioms in PSL-Core that constrain the interpretation of terms related to intuitions about time.

All other terms in the ontology are given definitions using the set of primitive terms. The defined terms 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 requires 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.

4 Case study: Translating from IDEF3 to ILOG using PSL

This section provides an example of using a common ontology as a Interlingua for facilitating manufacturing process information exchange between two different applications: ProCAP, a process modelling tool based upon the IDEF3 [26] method of systems modelling, and ILOG, a C++ library for constraint-based scheduling [27].

The example, extracted from the first pilot implementation of PSL [28], was based on a report developed by Ken McKay as part of the CAM-I (Computer Aided Manufacturing International) State of the Art Scheduling Survey [29]. The exchange scenario is as follows:

Within a particular company, two specific departments are involved in the planning and scheduling of a given product. The planning department's role is to document and describe the types or processes that are necessary to produce the product, specify the order in which these processes must occur (including temporal constraints, where available), and describe what types of resources are necessary for the creation of the product. The scheduling department then takes this information, instantiates the types of process and resources specified (i.e., assign machines, people, and specific times to those process and resources), and optimizes the process with respect to makespan (i.e., minimize the amount of time necessary to create the product). The challenge is that the planning department uses ProCAP process modeling package and the scheduling department uses ILOG Scheduler to do its scheduling. These two systems currently can not share information.

The company decides to use PSL to allow these two systems to work together (and to set the infrastructure so that other applications and departments can be integrated in the future). A translator is written between IDEF3, ProCAP's modelling language, and PSL and another between PSL and ILOG Scheduler. The person in the planning department creates their plan using the ProCAP tool and runs the translator to convert it to PSL. Once the planner feels comfortable that his plan is well represented in PSL, the scheduler can import this plan into his ILOG Scheduler, instantiate the plan such that specific processes and resources are assigned, and then optimize the plan in ILOG Scheduler with respect to whatever variable desired.

This approach (using PSL as an Interlingua instead of writing a direct translator) has the advantage that in order to translate from a given language, one needs only to write a translator from that language to/from PSL in order to make the language PSL compliant. Once this is done, information can be exchanged from that language to any other PSL compliant language.

Here's a description of the methodology used within the first PSL pilot implementation for writing the two translators mentioned above, as well as the issues that occured, and how they were solved in the pilot implementation.

Writing each of the translators consisted of two parts:

1. Identifying the concepts present in the respective language, extending the PSL ontology if needed – it was needed for both translators – in order to accomodate theese concepts as well as defining these concepts in terms of the PSL concepts.

2. Writing the translators proper, based on the concept definitions above.

4.1 IDEF3 to PSL Translator

IDEF3 is a graphical language designed for capturing information about the objects and processes involved in a system. It offers both a process-centered and an object-centered perspective, and it includes the ability to capture and structure descriptions of how a system works from multiple viewpoints. However, apart from its graphical element, there is no standard textual representation.

A preliminary such textual representation based upon the EPIF (Enhanced Process Interchange Format) [30] which is being developed by Chris Menzel at Knowledge Based Systems, Inc. was selected as the basis for this pilot implementation. Below is a description of the major components of this representation.

UOB's (Units of Behavior) are IDEF3's most fundamental building blocks that are used to represent activities. Furthermore, IDEF3 distinguishes UOB's (generic activities), UOB-uses (occurrences of UOB's in particular IDEF3 schematics), and UOB activations (collections of instances of UOB-uses that satisfy the temporal and logical constraints imposed by an IDEF3 schematic).

Branching is represented in IDEF3 using junctions. A process can branch (converge or diverge) into multiple parallel (AND-junction) or alternative (OR-junction or XOR-junction) sub-processes. Also, branching can be done in asynchronous (default) or synchronous mode. Particular junctions exist for all those combinations, and their exact semantics are defined in [26].

Links are used in IDEF3 schematics mainly to specify temporal constraints among the UOB's of a process schematic. Additional constraint links can also be used to express logical, causal, natural and conventional relations [26].

In order to be able to capture the semantic concepts of IDEF3, some extensions were created for the PSL language. Those extensions fall into several broad categories, which deal with: splitting of processes, synchronous splits, type-instance relationships for both activities and objects, and temporal sequencing of activities.

PSL Splitting Extensions. Within the PSL splitting extension, three new predicates, or_split, and_split and xor_split, were introduced. Each predicate has two activity arguments. The second argument denotes the process containing the junction. The first argument is a complex activity denoting the junction whose subactivities are the branches of the junction.

Synchronizing Axioms. We introduced two predicates, sync_start and sync_finish. The predicate sync_start specifies that all sub-activities instances of an activity start at the same time. The predicate sync_finish specifies that all sub-activities instances of an activity finish at the same time. In particular, the subactivities of split junctions can be synchronized using these relations.

Temporal Sequencing of Activities. A new predicate, follows, was introduced which is intended to capture the links concepts in IDEF3. Specifically, it is meant to capture the relationship between two activities in which one occurs directly after another.

The PSL representation of an IDEF3 schematic is a set of KIF sentences together with everything they imply. The translation process can be described by a set of compilation rules that associate KIF sentences with the IDEF3 constructs (writing such compilation rules can be also seen as providing a formal, declarative semantics into PSL to IDEF3 constructs)

The notion of compilation was defined relative to a process specification for each type of IDEF3 declaration and rules for the compilation of the individual slots of a declaration were written in the style used in [30]. For a complete account of the compilation rules used, please refer to [31].

Once the compilation rules were written, implementing the translator was a trivial task. The compilation rules were written as macros in the Lisp programming language [32], and the translator itself just expands those macros for all the forms of the IDEF3 file and stores the results in the PSL file. Examples of the IDEF3 file and translated PSL file can be found in Figure 7.

Issues faced. In writing the compilation rules, there were several issues to be resolved. Some of these issues included:

1. How to encode IDEF3 junctions into PSL?
2. How to encode the type-instance relationships for objects and activities?
3. What was the right abstraction level in PSL to which to do the translation?

Solutions. For the first pilot implementation, the above issues were resolved in the following ways:

1. IDEF3 junctions were identified with PSL complex activities, having as sub-activities the activities after (for Fan Out IDEF3 junctions) respectively before (for Fan In IDEF3 junctions) the junction in the IDEF3 schematic, having the additional occurrence constraints imposed by the type of junction used. This way, the ordering constraints imposed by the junction in the IDEF3 schematic get imposed by the ordering constraints of the activity - subactivities relationships in PSL.
2. Both activities and objects are type-level in IDEF3 and in PSL. However, they are encoded as predicates in IDEF3, but as objects in PSL. So, going with the PSL encoding, we choose to reify both activity and object types, introducing two new predicates to express their type-instance relationship. For the moment, these predicates are defined in terms of PSL's occurrence predicate.
3. As far as the IDEF3 to PSL translator is concerned, the level of abstraction at which the translation is done doesn't really matter. One has to think of translation rules that go all the way to the bottom (i.e. the PSL primitive concepts) and define the IDEF3 constructs in terms of these. Alternatively, once semantic agreement has been checked by going to the lowest level, the compilation rules can be written in terms of higher level concepts reused from the PSL Ontology, or in terms of concepts defined in a PSL extension.

However, to facilitate the style currently chosen for translating out of PSL, the higher level approach was chosen. That is, new high level semantic concepts were defined in terms of the PSL Ontology and the compilation rules were written in terms of those concepts.

4.2 PSL to ILOG Translator

ILOG Schedule [27] consists of an extensible library of C++ classes and functions that implement scheduling concepts such as activities and resources. The library enables the representation of scheduling problems as a collection of scheduling constraints, such as activity durations, release dates and due dates, precedence constraints, resource availability, and resource sharing. These constraints in turn are used as input for ILOG Solver, which can solve the constraints to provide schedules, in which activities are assigned to resources over different time intervals.

ILOG Schedule Ontology. There are three main classes within ILOG Schedule's ontology:

- IlcActivity
- IlcResource
- IlcSchedule

An instances of the class IlcSchedule is an object which represents a schedule. Any schedule is associated with a time interval, during which all activities in the schedule must occur.

The class IlcActivity is the root class for all activities which may occur in a schedule. All activities have a start time and an end time; the duration of an activity is the difference between these times.

Activities within a schedule satisfy precedence constraints. These constraints are used to define orderings over the occurrences of the activities. The following precedence constraints are defined in ILOG Schedule: endsAfter, endsAfterEnd, endsAfterStart, endsAt, endsAtEnd, endsAtStart, endsBefore, startsAfter, startsAfterEnd, startsAfterStart, startsAt, startsAtEnd, startsAtStart, startsBefore.

ILOG Schedule provides two predefined classes of activities: IlcIntervalActivity and IlcBreakableActivity. An instance of IlcIntervalActivity is an activity which occurs without interruption from its start time to its end time and which requires the same resources throughout its occurrence. An instance of IlcBreakableActivity is an activity whose occurrence can be interrupted.

Activities may also require resources, as specified by resource constraints. An activity consumes a resource if some amount of the resource capacity must be available during the occurrence of the activity and the capacity is non recoverable after the occurrence of the activity. An activity produces a resource if some amount of the resource capacity is made available through the occurrence of the activity. An activity requires a resource if some amount of the resource capacity must be available during the occurrence of the activity and the capacity is recoverable after the occurrence of the activity.

There are two main subclasses of IlcResource – resources with capacity (IlcCapResource) and resources with arbitrary states (IlcStateResource). Capacity-based resources in turn have two subclasses – resources which are simply required by activities (which are specified by the class IlcDiscreteResource) and resources which are provided by activities (which are specified by the class IlcReservoir).

PSL Extensions for ILOG Schedule. The translator for PSL and ILOG is based on the mapping between the concepts in the ILOG Schedule ontology and the PSL ontology. The biggest hurdle in specifying this mapping 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:

- Set Theory, which defines the basic notion of a set of objects;
- Resource Sets, which defines how 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 which can be substituted for others in an activity (e.g. the set of carpenters in a house construction activity).
- Homogeneous Sets, which defines different classes of substitutable resources
- Resource Pools, which are equivalent to discrete capacity resources within ILOG Schedule;
- Inventory Resource Sets, which are equivalent to reservoirs within ILOG Schedule.

The resource constraints in ILOG, such as requires, consumes, and produces, were completely defined within the Resource Roles extension of PSL, which had been specified before the pilot implementation.

The classes of state resources within ILOG Schedule led to the Reasoning about Fluents extension, which defined the notion of state within PSL.

The classes of interval and breakable activities within ILOG Schedule were defined in the Duration-based Complex Actions extension.

The precedence constraints among activities in a schedule within ILOG were defined in the Temporal Ordering extension of PSL.

Implementation of the Translator. The translator for ILOG/PSL consists of two parts – a semantic translator and a syntactic translator. The semantic translator maps concepts in ILOG to concepts in PSL by specifying the tranlation definitions between the terminology of the ILOG ontology and terminology within the corresponding PSL extensions. These translation definitions have the following form: – each relation in the ILOG ontology has a definition using only PSL terminology, and conversely, each relation in the PSL ontology has a definition using only ILOG terminology.

Syntactic translation of PSL to ILOG Schedule is a mapping from KIF sentences to C++ code specifying class definitions and/or instances of ILOG Schedule classes. This code can be compiled and executed to generate schedules based on the activities, resources, and constraints specified within the file. The resulting ILOG file can be found in Figure 7.

IDEF3 Source File

```
(define-UOB Make-Interior
    :documentation "Make Interior"
    :objects (A002-bench)
    :constraints (and (duration Make-Interior 5)
                      (BeginOf Make-Interior 7)))

(define-UOB Make-Trim
    :documentation "Make Trim"
    :objects (A002-bench A005-bench))
...
```

Resulting PSL file

```
(and (doc make-interior "Make Interior")
     (forall (?a :  (activation-of ?a make-interior))
       (exists (?o1 :  (instance-of ?o1 a002-bench)) (in ?o1 ?a)))
     (and (duration make-interior 5) (beginof make-interior 7)))

(and (doc make-trim "Make Trim")
     (forall (?a :  (activation-of ?a make-trim))
      (exists
       (?o1 ?o2 :  (instance-of ?o1 a002-bench)
        (instance-of ?o2 a005-bench))
       (and (in ?o1 ?a) (in ?o2 ?a)))))
...
```

Resulting ILOG file

```
IlcSchedule DefineProblem(IlcManager m, IlcIntVar& makespan) {
   IlcInt horizon = 100;
   IlcSchedule make_gt350_proc(m, 0, horizon);
   ...
   IlcIntervalActivity make_trim_1= MakeActivity(make_gt350_proc,3,"make_trim_1");
   IlcIntervalActivity make_interior_1=
       MakeActivity(make_gt350_proc,3,"make_interior_1");
   m.add(make_interior_1.startsAfterEnd(final_assembly_1,0));
   m.add(make_trim_1.startsAfterEnd(final_assembly_1,0));
   ...
}
```

Figure 7: An example translation from IDEF3 to ILOG using PSL.

4.3 Results and Further Reference

Both translators were run successfully on the scenatio provided. A more detailed description of the first PSL pilot implementation can be found in [28], or on the web at `http://www.mel.nist.gov/psl/` An in-depth presentation of the IDEF3 to PSL translator is available in [31].

5 Translation Issues

When using ontologies such as PSL as first-order Interlinguas, translating into them should be an easy task, due to their expressivity. It can be typically done by writing compilation rules that map each concept of a language with the corresponding Interlingua formula. Writing such compilation rules can be seen as providing a declarative semantics for the language, and the Interlingua can be thought of as a semantic description language. Previous work on the IDEF3 to PSL translator has shown that 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 a first order Interlingua to some target language seems to be more difficult. The usual target languages are a lot less expressive than the first order Interlingua, and the same approach does not work, since there is no way to write "definitions" in the target language for some of the Interlingua's constructs. Some form of re-aggregation has to be done, that is, the construct in the target language needs to be built whenever its Interlingua semantic definition is satisfied. The problem is that there are infinitely many ways of expressing the same thing in the first order Interlingua, and we want to build the target language construct whenever its first order definition, or some logically equivalent form of it can be inferred. Therefore, a first-order theorem prover has to be used for 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. The Interlingua 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 info specially 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 such an Interlingua:

- one has to be able to write translation rules (or be able to infer them) for all concepts in the Interlingua in order to build such a translator. (this task is made even harder by the extensibility requirements of Interlinguas intended for translation, like PSL)
- a translator that does full power first-order theorem proving at translation time will necessarily be very inefficient.

5.1 Deducing Translators

An approach to solving this problem is to develop a methodology for specifying compilation rules from arbitrary languages into first order Interlinguas, that is, a method for writing semantic descriptions for a language constructs into first order logic. 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 is expressed by its semantic description, together with the (formally specified) semantics of the Interlingua [6].

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

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

- It will solve the difficult problem of translating out of the Interlingua.
- 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 the Interlingua to be extensible, without requiring one to rewrite the "out of Interlingua" translators each time an 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).

6 Related Efforts

Ontology based methods hold great promises for the future, from enabling business-to-business electronic commerce to empowering the semantic world-wide-web, by extending already existing syntactic standards. In particular, the emergence of business-to-business electronic commerce and virtual enterprises is leading to a recognition of the need for shared understanding among very different enterprises.

In addition to the work within academia on the development of new ontologies and methodologies for building ontologies, there are also several related efforts within industry that are attempting to construct common data models to facilitate the seamless exchange of information among engineering applications. These common models can be considered to be informal ontologies – their terminology provides a common vocabulary, although their semantics are not formally specified.

The most prevalent efforts are based on the Extensible Markup Language (XML) [33], which is the universal format for structured documents and data on the World-Wide Web. XML is an extension of HTML that allows uses to structure the information in a document by specifying their own tags that correspond to pieces of information on the webpage. However, XML only provides a syntactic representation of the knowledge, so that operation is dependent upon each trading partner agreeing to use particular tag sets and using these consistently. There are several efforts that are creating such agreements in fields ranging from chemistry to geographical information systems.

DARPA's RaDEO program supported research, development, and demonstrations of enabling technologies, tools, and infrastructure for the next generation of design environments for complex electro-mechanical-optical (EMO) systems. Resulting from this challenge, notions of personal, organizational, and product webs that built on existing infrastructure such as the Internet and World Wide Web were estabilished.

There are also several industry consortia, typically specific to some market sector, that are developing common data models. The steel industry is using e-STEEL [34] to support an Internet-enabled online exchange among customers and suppliers. The Workflow Management Coalition [35] is a group of workflow vendors who have specified a common process description language to facilitate the exchange of workflow process descriptions among each others' workflow tools. The Public Petroleum Data Model Association [36] delivers a vendor-independent standard petroleum data model that serves as the industry foundation for managing information in the global business of oil and gas exploration and production. The ANX Network, which is being developed by the AIAG's Implementation Task Force – made up of representatives of the Big Three automakers and several major Tier One suppliers – has the potential to offer the auto industry significant savings by organizing infrastructure requirements in support of growth in networked applications. Biztalk [37] and the San Francisco project are efforts to provide a common repository of business objects that can be exchanged between different object-oriented software applications.

All of these efforts are ripe for the application of ontologies. Future research in ontologies will be required to extend these informal data models by providing them with a precise unambiguous formal semantics. In order to achieve this, one important issue for ontology research will be to find ways for implementing ontologies without reducing expressivity of ontologies as specification mechanisms.

7 Summary and Future Directions

One major hindrance to achieving interoperability among different engineering applications is the use of different terminology and domain representations in the application software to be integrated. Many current approaches for facilitating interoperability (including those based on XML) focus on sharing terminology, under the assumption that the shared terms mean the same thing in each of the applications being integrated. However, even when applications use the same terminology, they often associate different semantics with the terms. This clash over the meaning of the terms prevents the seamless exchange of information among the applications.

Ontologies support interoperability by providing semantics for terminology in a computer-interpretable format. This in turn enables the specification of semantics-preserving mappings between the terminology of different applications. Logic-based languages (such as KIF) will play a critical role in all of this, in order to guarantee that the translations produced by these languages conform correctly to the intended semantics of the the original applictions.

Two approaches for using ontologies to support interoperability include using a shared common ontology (as with enterprise modeling approaches), and using the ontology as an interlingua (in which applications are integrated by the use of translators through the ontology). Here, the use of languages for representing ontological information in a computer-interpretable form will again play a critical role. Automated translators require computer-interpretability; translators require semantics in some form, but if it is not completely computer-interpretable, then humans must assist in the translation. The use of the PSL ontology as an interlingua for translating process

information from the ProCAP process modelling tool to ILOG Schedule illustrates the feasibility of using ontology-based translators to support interoperability.

The next step will be the facilitation of translator development through the automatic inference of translation rules among arbitrary applications based only on the semantic descriptions of their ontologies. We believe that greater benefits still can become available when implementing ontologies, and the next generation of ontology based tools will not only be based on those shared specifications, but will treat ontologies and their model-theoretic implications as part of their reasoning process. Such computer enforceable ontologies will minimize the need for writing hand-crafted translators, while still facilitating the achievement of interoperability.

Acknowledgements.

This work was supported in part by the following grants and contracts: National Institute for Standards and Technology 70NANB6H0147 and 70NANBOH0016, Army Research Laboratory DAAL01-97-K0135, Air Force Research Laboratory F306029910013 and F30602-00-2-0505, and National Science Foundation DMI-9713718. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of authors and do not necessarily reflect the views of the sponsors.

References

[1] Thomas R. Gruber. Toward principles of the design of ontologies used for knowledge sharing. In *International Workshop on Formal Ontology in Conceptual Analysis and Knowledge Representation*, Padova, Italy, 1993.

[2] D. Lenat. Cyc: A large-scale investment in knoweldge infrastructure. *Communications of the ACM*, 38:33–38, 1995.

[3] H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972.

[4] John F. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks Cole Publishing Co., 2000.

[5] Michael R. Genesereth and Richard E. Fikes. Knowledge Interchange Format Version 3.0 Reference Manual. Technical report, Logic Group, Stanford University, CA., 1992.

[6] Mihai Ciocoiu and Dana S. Nau. Ontology-based semantics. In Anthony G. Cohn, Fausto Giunchiglia, and Bart Selman, editors, *Principles of Knowledge Representation and Reasoning. Proceedings of the Seventh International Conference*, pages 539–546, Breckenridge, Colorado, April 11-15 2000. Morgan Kaufmann.

[7] Gruninger M. and M.S. Fox. Enterprise modelling. *AI Magazine*, 19:109–121, 1998.

[8] Gruninger M. Ontologies for enterprise engineering. *Enterprise Engineering and Integration: Building International Consensus*, 1997.

[9] M. Gruninger and J. Pinto. A theory of complex actions for enterprise modelling. In *AA AI Spring Symposium Series 1995: Extending Theories of Action: Formal Theory and Practical Applications*, pages 71–79. AAAI Press, 1995.

[10] F. Fadel, M.S. Fox, and M. Gruninger. A generic enterprise resource ontology. In *Proceedings of the Third Workshop on Enabling Technologies - Infrastructures for Collaborative Enterprises*, pages 86–92. West Virginia University, 1994.

[11] D. Tham, M.S. Fox, and M. Gruninger. A cost ontology for enterprise modelling. In *Proceedings of the Third Workshop on Enabling Technologies - Infrastructures for Collaborative Enterprises*, pages 111–117. West Virginia University, 1994.

[12] H. Kim, M.S Fox, and T. Bilgic. An ontology for quality management: Enabling quality problem identification and tracing. *BT Technology Journal*, 17(4), 1999.

[13] M.S. Fox, M. Barbuceanu, and M. Gruninger. An organisation ontology for enterprise modelling: Preliminary concepts for linking structure and behaviour. *Computers in Industry*, 29:123–134, 1995.

[14] J Lin, M.S Fox, and T. Bilgic. A requirements ontology for concurrent engineering. *Concurrent Engineering: Research and Applications*, 4:279–291, 1996.

[15] M. Gruninger, K. Atefi, and M.S. Fox. Ontologies to support process integration in enterprise engineering. *Computational and Mathematical Organization Theory*, 2000.

[16] W. Clocksin and C. Mellish. *Programming in PROLOG*. Springer-Verlag, 1981.

[17] M. Uschold, M. King, S. Moralee, and Y. Zorgios. The enterprise ontology. *Knowledge Engineering Review*, 13:47–76, 1998.

[18] F. Fillion and C. Menzel. An ontology-based environment for enterprise model integration. In *IJCAI Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995.

[19] Inc. Knowledge Based Systems. IDEF0 Overview. http://www.idef.com/idef0.html.

[20] Inc. Knowledge Based Systems. IDEF1x Overview. http://www.idef.com/idef1x.html.

[21] J. Lee, Gruninger, Y. M., Jin, T. Malone, Tate, and G. A., Yost. The pif process interchange format and framework. *Knowledge Engineering Review*, 2:1–30, 1998.

[22] S. Polyak, J. Lee, M. Gruninger, and C. Menzel. Applying the process interchange format (pif) to a supply chain process interoperability scenario. In *European Conference on Artificial Intelligence*, pages 247–253. John Wiley and Sons, 1998.

[23] C. Schlenoff, A. Knutilla, and S. Ray. Unified process specification language: Requirements for modeling process. Technical Report NISTIR 5910, National Institute of Standards and Technology, Gaithersburg, MD, 1996.

[24] M. Gruninger, C. Schlenoff, and A. Knutilla. Using process requirements as the basis for the creation and evaluation of process ontologies for enterprise modeling. *ACM SIGGROUP Bulletin Special Issue on Enterprise Modelling*, 18(3), August 1997.

[25] Craig Schlenoff, Michael Gruninger, and Mihai Ciocoiu. The essence of process specification. submitted, by invitation, to the Special Issue on Modeling and Simulation of Manufacturing Systems for the Society for Computer Simulation International, 1999.

[26] Richard J. Mayer, Christopher P. Menzel, Michael K. Painter, Paula S. deWitte, Thomas Blinn, and Benjamin Perakath. Information Integration for Concurrent Engineering (IICE) IDEF3 Process Description Capture Method Report. Technical report, Knowledge Based Systems Inc., KBSI-IICE-90-STR-01-0592-02, 1995.

[27] *ILOG Scheduler 4.3 Reference Manual*, June 1998.

[28] Craig Schlenoff, Mihai Ciocoiu, Don Libes, and Michael Gruninger. Process specification language: Results of the first pilot implementation. In *Proceedings of the International Mechanical Engineering Congress and Exposition (IMECE)*, November 1999.

[29] Kenneth N. McKay and John B. Moore. Intelligent manufacturing management. state of the art scheduling survey 06-23-91. Technical Report R-91-IMM-01, CAM-I (Consortium for Advanced Manufacturing International), 1991.

[30] Knowledge Based Systems Inc. Foundations for Product Realization Process Knowledge Sharing. Technical report, U.S. Department of Commerce, NOAA Contract No. 50-DKNB-7-90095, 1995.

[31] Mihai Ciocoiu. Translating IDEF3 to PSL. Technical report, University of Maryland, Computer Science Technical Report, CS-TR-3950, 1998.

[32] Paul Graham. *ANSI Common Lisp.* Prentice Hall, 1996.

[33] W3C. Extensible Markup Language (XML) 1.0. http://www.w3.org/TR/REC-xml/.

[34] e-STEEL: the Steel Commerce Solution. http://www.e-steel.com/home.shtml.

[35] Workflow Management Coalition. http://www.aiim.org/wfmc/.

[36] Public Petroleum Data Model Association. http://www.ppdm.org/.

[37] Biztalk. http://www.biztalk.org/.