ONTOLOGY ENGINEERING FOR DISTRIBUTED COLLABORATION IN MANUFACTURING

Line Pouchard,
Collaborative Technologies Research Center
Oak Ridge National Laboratory
Oak Ridge, TN 37831-6414
pouchardlc@ornl.gov

Nenad Ivezic
Manufacturing Engineering Laboratory
National Institute of Standards and Technology
100 Bureau Drive
Gaithersburg, MD 20899
nivezic@nist.gov

Craig Schlenoff
Manufacturing Engineering Laboratory
National Institute of Standards and Technology
100 Bureau Drive
Gaithersburg, MD 20899
schlenof@cme.nist.gov

KEYWORDS: Process Specification Language, ontology, inter-operability, agent-based systems.

ABSTRACT

The problems of inter-operability are acute for manufacturing applications, as applications using process specifications do not necessarily share syntax and definitions of concepts. The Process Specification Language developed at the National Institute of Standards and Technology proposes a formal ontology and translation mechanisms representing manufacturing concepts. When an application becomes 'PSL compliant,' its concepts are expressed using PSL, with a direct one-to-one mapping or with a mapping under certain conditions. An example of how to resolve semantic ambiguities is provided for the manufacturing concept 'resource'. Finally some ideas on how to use PSL for inter-operability in agent-based systems are presented.

1. INTRODUCTION

As enterprise integration increases in the manufacturing domain, developers face increasingly complex problems related to inter-operability. Independent contractors and suppliers who collaborate on demand within virtual supply chains to bring to market new products must share product-related data. Legacy vendor applications that are not designed to inter-operate must now share processes. When enterprises collaborate, a common frame of reference or at least a common terminology is necessary for human-to-human, human-to-machine, and machine-to-machine communication. Similarly, within a core enterprise where distributed collaboration between remote sites and production units take place, a common understanding of business- and manufacturing-related terms is indispensable. However, this common understanding of terms is often at best implicit in the business transactions and software applications and may not even be always present. Misunderstandings between humans conducting business-related tasks in teams, and ad-hoc translations of software applications contribute to the rising costs of interoperability in manufacturing. Moreover, manufacturing enterprise applications using distributed software agents are becoming increasingly ubiquitous. Software agents require a shared terminology and syntax in order to efficiently and effectively inter-operate.

Ontology engineering offers a direction towards solving the inter-operability problems brought about by semantic obstacles, i.e. the obstacles related to the definitions of business terms and software classes. Ontology engineering is a set of tasks related to the development of ontologies for a particular domain. An ontology is a taxonomy of concepts and their definitions supported by a logical theory (such as first-order predicate calculus).

Ontologies have been defined as an explicit specification of a conceptualization (Gruber 1993). Ontology engineering aims at making explicit the knowledge contained within software applications, and within enterprises and business procedures for a particular domain. An ontology expresses, for a particular domain, the set of terms, entities, objects, classes and the relationships between them, and provides formal definitions and axioms that constrain the interpretation of these terms (Gomez-Perez 1998). An ontology permits a rich variety of structural and nonstructural relationships, such as generalization, inheritance, aggregation, and instantiation and can supply a precise domain model for software applications (Huhns and Singh 1997). For instance, an ontology can provide the object schema of object-oriented systems and class definitions for conventional software (Fikes and Farquhar 1999). Ontological definitions, written in a variety of logical languages, are human-readable. They can also serve to automatically infer translation engines for software applications. By making explicit the implicit definitions and relations of classes, objects, and entities, ontology engineering contributes to knowledge sharing and re-use (Gomez-Perez 1998).

Throughout the manufacturing life cycle, software applications, such as process planning, process modeling, scheduling, workflow and simulation, use process information to describe the activities involved in production, resource requirements, ordering relations and temporal constraints. These applications do not usually inter-operate, although the output data and processes of one application may constitute the input of another. For each software application and vendor, a translator must therefore be written to allow for data and process sharing. Because the process definitions are not explicit, one encounters incompatibilities due to problems of synonymy and inconsistencies due to semantic plurality. Synonymy occurs when two objects or classes representing the same function are called with a different name or string. It is not obvious in machine communication that automobile (application A) = vehicle (application B). Semantic plurality occurs when the same names cover two different meanings in two applications. For instance, resource (application A) = consumable-resource, whereas resource (application B) = machine-tool. Point-to-point translators have traditionally been designed for problems of this kind. But when many applications need to inter-operate, the number of translators to be written increases exponentially, and so does the cost of implementing inter-operability.

The Process Specification Language (PSL), developed at the National Institute of Standards and Technology (NIST 1999), takes a different approach to developing inter-operable applications. PSL ‘s approach is to develop an ontology providing unambiguous definitions for manufacturing-related concepts and mechanisms to support translation of definitions among applications. PSL seeks to create a standard language for process specification with which to specify a process or flow of processes and supporting parameters and to serve as a common exchange language between manufacturing processes (Schlenoff et al. 1999a). PSL is a robust exchange technology enabling distributed collaboration among manufacturing applications. This paper shows how the Process Specification Language handles issues related to semantic problems when applied to manufacturing processes. It also offers some initial ideas on how to adapt PSL for use in agent-based systems for manufacturing enterprise integration.

2. SEMANTIC MODELING USING THE PROCESS SPECIFICATION LANGUAGE

Definitions of concepts within PSL are expressed using Knowledge Interchange Format (KIF) (Genesereth 1992) and use formal logic to define their semantics. The use of KIF provides the advantage of being able to prove the consistency and completeness of definitions and axioms. The PSL ontology is extensible, and built upon a core of axioms and extensions. There are three basic entities (activity, object, and timepoint) and four basic relations (participates_in, before, begin_of and end_of). The PSL core ontology and extensions are discussed elsewhere (Schlenoff et al. 1999a, 1998; Knutilla et al. 1997). PSL presently contains about 300 concepts distributed within 31 modules. Concepts intrinsic to process modeling, scheduling, and simulation are currently represented. To validate PSL as an exchange language, the first pilot implementation exchanging process information between an IDEF3-based process modeling tool and a C++ scheduler application was successfully implemented (Schlenoff et al. 1999b). The second pilot implementation demonstrating inter-operability between process planning and simulation applications is currently under way.

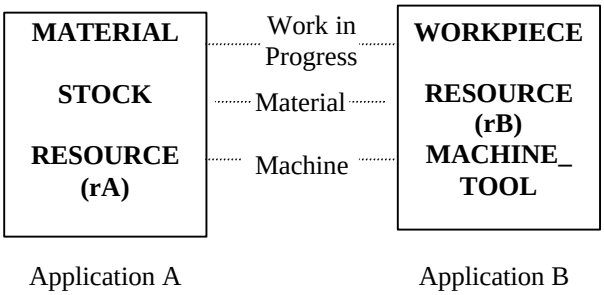

Figure 1: Semantic plurality for resource

An example of a semantic problem is the case of two process planning applications, Application A and Application B both containing general concepts for designating resources. The objects designating these concepts are "Material, Stock, and Resource" in Application A, and Workpiece, Resource, and Machine_Tool" in Application B. The semantic problems are expressed in Figure 1.

Semantic plurality occurs for *resource* where *resource* designates a machine entity in application A (rA) and a material entity in application B (rB). If the processes of application A need to be analyzed by Application B, translators between the two applications are written. However, systems designed with procedural and object-oriented languages contain implicit definitions of classes, objects, and relations. Formal definitions for objects and classes are not usually supplied with a system because the system does not need a formal definition of a class or object in order to invoke a method or apply inheritance. As a result, the semantic problem described above may be ignored during translation, introducing confusion. A translator is typically a set of compilation rules allowing the syntax of one application to be parsed to the other. (Ciocoiu 1998; Schlenoff 1999b). A translator may compile the syntactical rules of use for Resource (rA) of application A as rules for Resource (rB) of Application B.

PSL provides a neutral, standard description of the processes in manufacturing so that the concepts of application A can be unambiguously expressed using PSL concepts. The concepts of Application B are also expressed using PSL concepts. PSL thus provides an interlingua for manufacturing processes. A given application is said to be "PSL compliant" if there exists a proven translator to and from PSL for that application (Schlenoff 1999a, 1999b). In an domain where inter-operability between applications is becoming ubiquitous, the PSL approach reduces the number of translators from O (n2) to O (n) by requiring that an application only maps its concepts to PSL concepts, rather than mapping to all the other applications.

3. REPRESENTING CONCEPTS USING PSL

In our example, Application A's original syntax and terminology reads:{resourceA: inject_mold (x)}. First the concepts of application A must be formally represented using KIF syntax or any other formal representation that can be translated to KIF. The concepts of application A represented in KIF syntax express that inject_mold is a resource:

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

The ontology for Application A is also expressed. This non-trivial task is performed using technical documentation that is supplemented by interviews with developers and training by vendors. Expressing the ontology of application A means expressing what kind of resource is implied by *resourceA* in application A. Here *resourceA* is re-usable; it is a machine that can still be used by a process after another process that also requires resourceA completes its occurrence. The ontology of application A expresses the definition for *resourceA*, where ?r is the resource variable and ?a the process variable:

```
(forall (?r ?a)
      (<=>   (resourceA ?r)
          (exists (?a)
          (reusable ?r ?a)))
```

Once a definition of resource is provided for application A, using Application A terminology, the next step is to look for possible mappings between *resourceA* and PSL concepts. 3 possibilities occur: 1)a one-to-one mapping is possible, 2)a one-to-one mapping is possible under certain conditions, 3)PSL does not contain the concept in question and needs to be extended to accommodate application A. This mapping is done for every concept and relation contained in ApplicationA.

PSL contains a concept of resource defined as-- a resource is any object that is required by some activity -- where "activity" and "requires" are defined elsewhere in PSL. This definition is expressed with KIF syntax:

```
(defrelation resource (?r) :=
(and    (object ?r)
      (exists (?a)
```

```
                    (requires ?a ?r))))
```

PSL also specifies roles for resources and defines the concept of a reusable resource as -- a resource ?r is reusable by an activity ?a if any other activity that also requires ?r is still possible to perform after ?a completes its occurrence, in every possible future -- where "common," "occurs_over," "legal_interval," "legal," and "legal_activity" are defined elsewhere in PSL. This definition is expressed with KIF syntax:

```
        (defrelation reusable (?r ?a1) :=
        (forall (?a2 ?occ)
              (=>    (and    (common ?a1 ?a2 ?r)
                             (occurs_over ?a1 ?occ))
                     (forall (?b)
                  (=>    (forall (?s3)
                                (=>    (and
(legal_interval ?b)

(situation_during ?s3 ?b)

(occurs_during ?occ (legal ?s3)))
                                       (legal_activity ?a2
?s3)))))))
```

In our example, a mapping appears to exist, but more information about ApplicationA is needed before deciding for a one-to-one mapping. This information must be obtained from the documentation for Application A, and other sources (vendors). PSL specifies that a reusable resource is such that, as soon as one activity occurs, it is always possible to perform the next activity. An example of reusable resource according to PSL is a machine that does not require setup between activities. If the resource in application A satisfies this condition, we have a one-to-one mapping between the Application A concept, *resourceA*, and the PSL concepts, *resource* and *reusable*. This mapping is expressed as:

```
        (forall (?r)
                (<= >  (resourceA ?r)
                       (and (resource ?r)
                              (reusable ?r)))
        )
```

Another possibility is a non-direct, conditional mapping. An example occurs in the first pilot implementation where the ILOG (TM) scheduler offers the concept *ilcActivity*. The *ilcActivity* concept is narrower than the PSL concept of *activity*: it maps to the PSL *activity* concept only if the PSL *activity* is both *primitive* and a *non-deterministic resource activity*. PSL definition: -- An activity is a nondeterministic resource activity iff the reason that the activity is nondeterministic is because it is a nondeterministic selection activity with respect to some resource set. The one-to-one mapping with conditions between Application A concepts and PSL concepts is expressed:

```
        (forall (?a)
                (=> (and (nondet_res_activity
        ?a)
                         (primitive ?a))
                    (<=>   (ilcActivity ?a)
                         (activity ?a))))
```

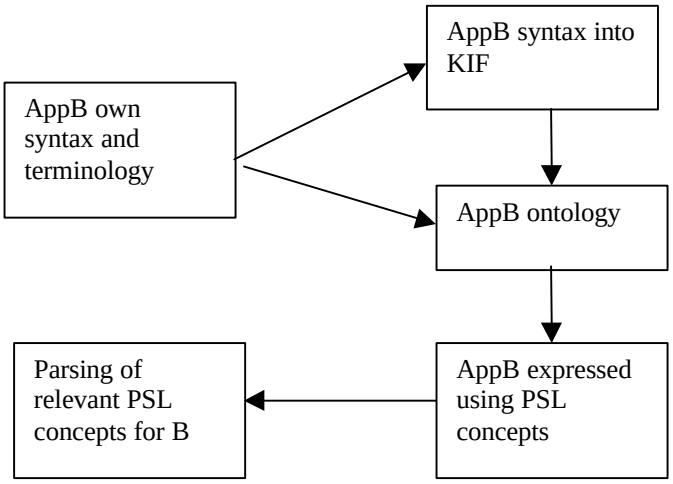

Figure 2: Steps for expressing an application concepts into PSL

Similar steps are followed in the translation of Application B's concepts into PSL (Figure 2): an ontology expresses the concepts of Application B, the syntax of Application B is expressed in KIF, and a translator is written for mapping the concepts of Application B to PSL concepts. The result of the second mapping is that every concept of application B is defined in terms of PSL concepts. In addition, a parsing of each PSL concept relevant to Application B concepts is done using the mapping. Our two applications A and B have now become "PSL compliant". When an engineer implements a model for a manufacturing task using Application A, the model is represented with PSL concepts, using a translator that is parsing the application to KIF. The model represented with PSL concepts is subsequently imported into Application B (Figure 3) (see Schlenoff 1999b for details of the implementation).

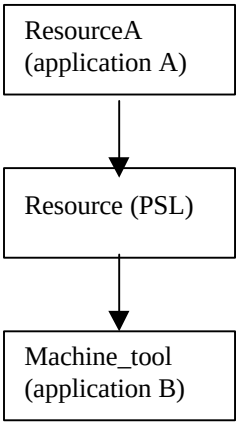

Figure 3: Importing a model from application A to application B

4. FUTURE TRENDS: PSL IN AGENT-BASED SYSTEMS

The Process Specification Language is a good candidate for developing a content language for manufacturing agent-based systems. Agent-based systems are implemented on different platforms, for different purposes, using sometimes proprietary languages. Among others, these systems encounter the problems of inconsistencies and incompatibilities described above when agents attempt to inter-operate(Genesereth 1994). Agent communication languages are attempting to partially resolve the problems of communication between agents by proposing common communication protocols and standards for communication languages. KQML and FIPA ACL are the best known agent-communication languages. They meet the challenges of inter-operability with mitigated success (Labrou et al. 1999). An agent-communication language should theoretically let heterogeneous agents communicate, but none currently do (Singh 1998). A significant part of the inter-operability issue is the lack of a shared content language and ontology. A PSL-based content language may possibly fulfill this role. This content language would enhance the inter-operability of agent-based systems that need to exchange data about their manufacturing processes, such as heterogeneous Supply Chain Management systems.

Like FIPA ACL, the PSL effort takes a standardization approach to the inter-operability problem. PSL was accepted as a Preliminary Work Item before Sub-Committee 4, Technical Committee 184, within the International Standards Organization. Agents for manufacturing can be designed to use the declarative semantics of PSL to exchange information with each other. Once agents have identified each other (through advertising services) and have established communication through a common protocol, they still need to refer to a common representation language for the content of the communication to be meaningful. Agents can refer directly to a language based on PSL as the ontology underlying message content. Alternatively, agents can use representations that are translatable (Finin et al. 1994). PSL may intervene at the content layer of an agent communication language for representing domain-specific knowledge needed by the agents to act on the messages they receive. For example, a KQML performative (a speech-act based message) uses a content argument and some optional arguments, such as ontology (Finin et al. 1994). The syntax of a KQML message can look like this:

```
(ask-one
        :sender jane
        :content (stack_paint ?stack)
        :receiver paint-server
        :ontology PSL)
```

where ask-one is a KQML performative, and sender, content, receiver, and ontology are its arguments. In this example, stack_paint is an intermediate queue, i.e. a resource-related concept that can be expressed using PSL terminology. Thus two agents using the same communication protocol (FIPA ACL or the same version of KQML) can exchange process information by using PSL as a content language for their interaction.

We have previously developed a manufacturing agent-based system (MABES) for design and analysis of discrete manufacturing systems with a collaborative user interface (Ivezic et al.1999). MABES supports analysis of transitions from traditional (i.e., push) to lean manufacturing, scheduling, and management setups (i.e., pull and takt) within a distributed agent-based framework. Further MABES development aims at effectively supporting distributed developers and users within a new collaborative agent framework. Within such a new framework, distributed teams could develop new system components (i.e., agents) that, under the assumption of shared communication and content languages, could inter-operate within a larger system.

The distributed agent-based system will allow closer human-agent interaction, deferral of responsibilities, and automated monitoring,

processing, and execution by the agent system. In order to support such an extended agent-to-human and agent-to-agent collaboration, a shared semantic definition of manufacturing concepts is required. PSL is a prime candidate for the content language definition to support collaboration and distributed development of simulation systems such as MABES.

5. CONCLUSION

This paper described how the Process Specification Language is used to overcome semantic-related obstacles, such as synonymy and semantic plurality, for the inter-operability of manufacturing processes. PSL uses KIF syntax and semantics and contains a core ontology and extensions related to manufacturing concepts. PSL 's appeal is that it proposes a standard content language for the exchange of processes between heterogeneous applications. PSL also holds a promise as a content language for agent communication in manufacturing agent-based systems by providing unambiguous concept definitions for the content layer of agent-communication languages.

REFERENCES

Ciocoiu, Mihai. 1998. "Translating IDEF3 to PSL." Technical Report 98-63. Department of Computer Science, University of Maryland, College Park, MD.

Fikes, R. and A. Farquahr. 1999. "Distributed Repositories of Highly Expressive Reusable Ontologies." *IEEE Intelligent Systems and their Applications* 14, no.2 (March-Apr.): 73~79.

Finin, T. R. Fritzson, D. McCay, R. McEntire. 1994. "KQML as an Agent Communication Languages." In *Proceedings of CIKM '94* (Gaithersburgh, MD, Nov.).

Genesereth, M. 1994. "Software Agents." *Communications of the ACM 37*, no. 7 (July): 48~53, 147.

Genesereth, M. and R.E. Fikes. 1992. "Knowledge Interchange Format, Version 3.0. Reference Manual." Technical Report Logic-92-1. Computer Science Department, Stanford University, Stanford, CA. (Jan.).

Gomez-Perez, A. 1998. "Knowledge Sharing and Re-Use." In *The Handbook of Applied Expert Systems*, J. Liebowitz, ed. Boca Raton, 10:1-10:36.

Gruber, T. 1993. "A Translation Approach to Portable Ontology Specifications." *Knowledge Acquisitions* 5, (May): 199-220.

Hunhs, M. N. and M. P. Singh. 1997. "Ontologies for Agents." *IEEE-Internet Computing* 1, no. 6 (Nov.-Dec. 1997): 81~83.

Ivezic, N., Potok, T. E., and Pouchard, L. C., 1999. "Multiagent Framework for Lean Manufacturing." *IEEE Internet Computing* 3., no. 5, (Sept.-Oct.): 58~59.

Knutilla, A.; C. Schlenoff; S. Ray et al. 1997. "Process Specification Language: Analysis of Existing Representations" NISTIR 6133. National Institute of Standards and Technology, Gaithersburgh, MD, (Sept.)

Labrou, Y., Finin, T., and Peng Y. 1999. "Agent Communication Languages: the Current Landscape." *IEEE Intelligent Systems and their Applications* 14, no.2 (Mar.-Apr.): 45~52.

National Institute of Standards and Technology (NIST) 1999. "Process Specification Language." http://www.mel.nist.gov/psl.

Schlenoff, C.; M. Gruninger; and M. Ciocoiu. 1999a. "The Essence of the Process Specification Language." *Transactions of the Society for Computer Simulation International*. Special Issue on Modeling and Simulation of Manufacturing Systems. Forthcoming.

Schlenoff, C.; M. Gruninger; M. Ciocoiu; and D. Libes. 1999b. "Process Specification Language (PSL): Results of the first Pilot Implementation." In *Proceedings of IMECE: International Mechanical Engineering Congress and Exposition* (Nashville, TN, Nov. 14-19).

Schlenoff, C.; R. Ivester; and A. Knutilla. 1998. "A Robust Process Ontology for Manufacturing Systems Integration." In *Proceedings of the 2nd International Conference on Engineering Design and Automation* (Maui, HI, Aug. 7-14).

Singh, M. 1998. "Agent Communication Languages: Rethinking the Principles." *IEEE Computer* 31, no. 12 (Dec.): 40~47.

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

Our work is also supported by Oak Ridge National Laboratory, managed by the Lockheed Martin Energy Research Corporation for the US Department of Energy. This article was written under contract no. DE-AC05-96OR22464.