Proceedings of IMECE:
International Mechanical Engineering Congress and Exposition
Nov 14-19, 1999, Nashville, Tennessee, USA

ME8B1

PROCESS SPECIFICATION LANGUAGE (PSL): RESULTS OF THE FIRST PILOT IMPLEMENTATION

Craig Schlenoff
National Institute of Standards and Technology

Mihai Ciocoiu
University of Maryland, College Park

Don Libes
National Institute of Standards and Technology

Michael Gruninger
University of Toronto

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 manufacturing functions may use different terms to mean the exact same concept or use the exact same term to mean very different concepts. Often, the loosely defined natural language definitions associated with the terms contain so much ambiguity that they do not make the differences evident and/or do not provide enough information to resolve the differences.

A solution to this problem is the development of a taxonomy, or ontology, of manufacturing concepts and terms along with their respective formal and unambiguous definitions. This paper focuses on the Process Specification Language (PSL) effort at the National Institute of Standards and Technology whose goal is to identify, formally define, and structure the semantic concepts intrinsic to the capture and exchange of process information related to discrete manufacturing. Specifically, it describes the results of the first pilot implementation, where PSL was successfully used as an interlingua to exchange manufacturing process information between the Knowledge Based System Inc.'s IDEF3-based ProCAP[1] process modeling tool and the C++ based ILOG Scheduler.

1.0 INTRODUCTION

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

The challenge of interoperability is especially apparent with respect to manufacturing process information. The term "manufacturing process information" refers to information describing the manufacturing operations needed to realize a product, including a high level description of the activity, resource requirements, ordering relations, temporal constraints, etc. Many manufacturing engineering, operations, and business software applications use process information, including production scheduling, manufacturing process planning, workflow, business process reengineering, simulation, process

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

realization, process modeling, and project management. Each of these applications utilizes process information in a different way, so it is not surprising that these applications' representations of process information are different as well. The primary difficulty with developing a standard to exchange process information is that these applications sometimes associate different meanings to the terms representing the information that they are exchanging. For example, in the case of a workflow system, a resource is primarily thought of as the information that is used to make necessary decisions. In a process planning system, a resource is primarily thought of as a person or machine that will perform a given task. If one were to integrate a process model from a workflow and a process planning application, their first inclination would most likely be to map one resource concept to the other. This mapping would undoubtedly cause confusion. Therefore, both the semantics and the syntax of these applications need to be considered when translating to a neutral standard. In this case, the standard must be able to capture all of the potential meanings behind the information being exchanged.

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

2.0 PROJECT APPROACH

The plan for the PSL project has five phases: requirements gathering, existing process representation analysis, language creation, pilot implementation and validation, and submission as a candidate standard. The completion of the first phase resulted in a comprehensive set of requirements for specifying processes [SCH96]. In the second phase, twenty-six process representations were identified as candidates for analysis by the PSL team and analyzed with respect to the phase one requirements [KNU98]. Nearly all of the representations studied focused on the syntax of process specification rather than the meaning of terms, or semantics. While this is sufficient for exchanging information between applications of the same type, such as process planning, different types of applications associate different meanings with similar or identical terms.

As a result of this, a large focus of the third phase involved the development of a formal semantic layer (an ontology) for PSL based on the Knowledge Interchange Format (KIF) specification [GEN92]. By using this ontology to explicitly and clearly define the concepts intrinsic to manufacturing process information, PSL was used to integrate two manufacturing process applications in the fourth phase of the project. This paper focuses on results of that pilot implementation.

3.0 OVERVIEW OF THE FIRST PILOT IMPLEMENTATION

3.1 Purpose

The purpose of the PSL pilot implementations is: 1) to grow and improve the initial process specification language, 2) to ensure that it is able to handle real-world exchange scenarios, and 3) to ensure that PSL can interface well with typical process-related software packages. There are multiple pilot implementations planned, each focusing on a different package/field within the area of manufacturing process (e.g., process planning, scheduling, simulation, workflow, etc.). Each pilot implementation will involve the exchange of process information between two or more process-related software packages using PSL as the interchange language.

3.2 Pilot Implementation Approach

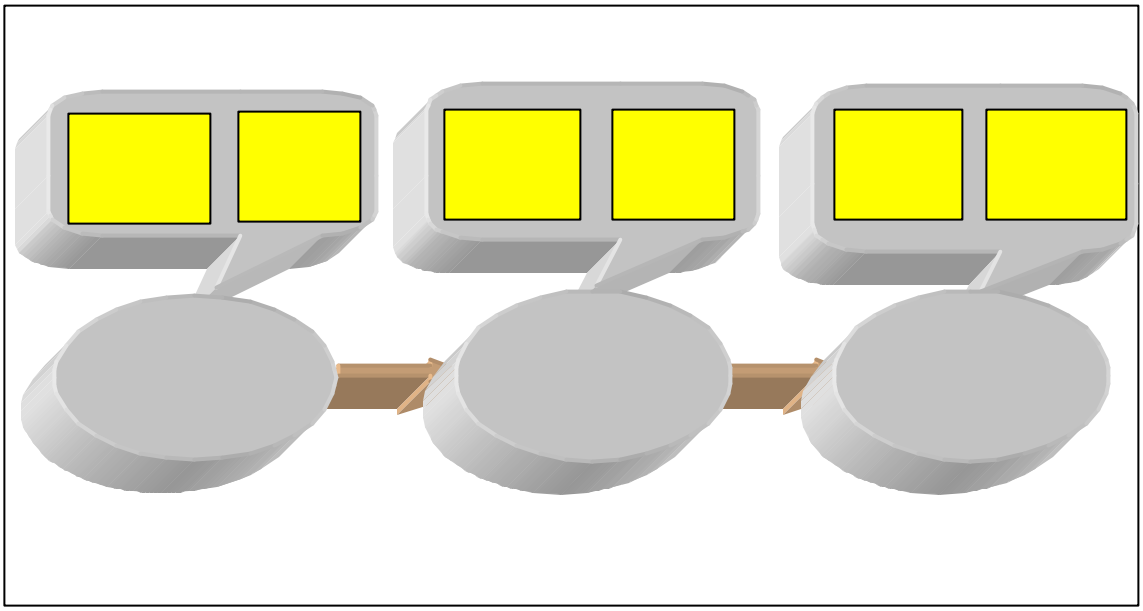
Specific steps in these pilots include: 1) identifying and clearly defining the concepts intrinsic to each application, 2) mapping the application's concepts to concepts within PSL (and extending PSL if necessary), and 3) writing translators between the application and PSL. As mentioned above, once an application becomes "PSL compliant" (i.e., once a proven translator is written to/from PSL), it becomes able to exchange information with every other application that is also "PSL compliant."

Figure 1 shows the steps that were performed in the pilot implementation to exchange process information between ProCAP and ILOG Scheduler. The manufacturing process information used for the exchange 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 [McK91]. This fictitious but real-world type exchange scenario is as follows:

Within a particular company, two departments are involved in process planning and scheduling for a given product. The manufacturing engineering department's role is to document and describe the types of processes that are necessary to produce the product, specify the order in which these processes must occur (including temporal constraints, where needed), and describe what types of resources are necessary for the creation of the product. Plant operations then takes this information, instantiates the processes and allocates resources needed (i.e. machines, people, materials and time), and optimizes the process with respect to makespan (i.e., minimize the amount of time

necessary to create the product). The challenge is that manufacturing engineering uses the ProCAP process modeling package and operations uses ILOG Scheduler to do its scheduling. These two systems currently cannot 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 ProCAP and PSL and between PSL and ILOG Scheduler. Then the manufacturing engineer creates their process plan using the ProCAP tool and runs the translator to convert it to PSL. This PSL file can currently be displayed in one of three ways: 1) as raw KIF, 2) as KIF in HTML format (in which all terms used in any definition are linked to the location where that term is defined), or in an object-oriented presentation especially designed for PSL. Once the process engineer feels comfortable that his plan is well represented in PSL, the operations 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 is desired.



3.3. Description of ProCAP and ILOG

ProCAP [KNO98] is a process-modeling tool developed by Knowledge Based Systems Inc. (KBSI). ProCAP is based upon the IDEF3 [MAY95] method of systems modeling. The IDEF3 method focuses on the abstraction and capture of knowledge about a given real-world system, including the temporal, causal, and logical relations between processes occurring within the system; the objects that participate in the process; and the state transitions of those objects. As a result, the method allows you to capture and describe not what happens at this or that particular time in a system, but instead what actually occurs in a system: the dynamic patterns that occur again and again among elements in a system.

ILOG Scheduler [ILO98] is a scheduling tool developed by ILOG Inc. ILOG Scheduler is a C++ library for constraint-based scheduling. This library is not a new programming language; it lets users use data structures and control structures provided by C++. Thus, the Scheduler part of the application can be completely integrated with the rest of that application (for example, the graphic interface, connections to databases, etc.) because it can share the same objects.

Scheduler is, in fact, an extension of the Solver C++ constraint-programming library, providing specially adapted classes and functions for managing the allocation of resources to activities over time.

3.4. Advantage of using a formal ontology during translation

PSL is unique from most other interchange languages for the primary reason that it is based on a formal ontology that explicitly and unambiguously defines all terms introduced within the language. This ontology provides at least two distinct advantages with respect to translation: 1) because all assumptions and definitions are explicitly represented in the ontology, translation (and in particular the mapping of concepts among different representations) becomes a theoretical and

provable exercise as opposed to being based on the opinions of the integrator, and 2) because PSL focuses on the meaning of concepts as opposed to what term is used to represent the concept, an application which is "PSL compliant" only has to become compliant with the definition stated within PSL as opposed to having to adopt the terminology the PSL happens to use. This allows software applications complete freedom in choosing the term that best describes the concept they are trying to bring across in their respective domain by being bound to PSL terminology.

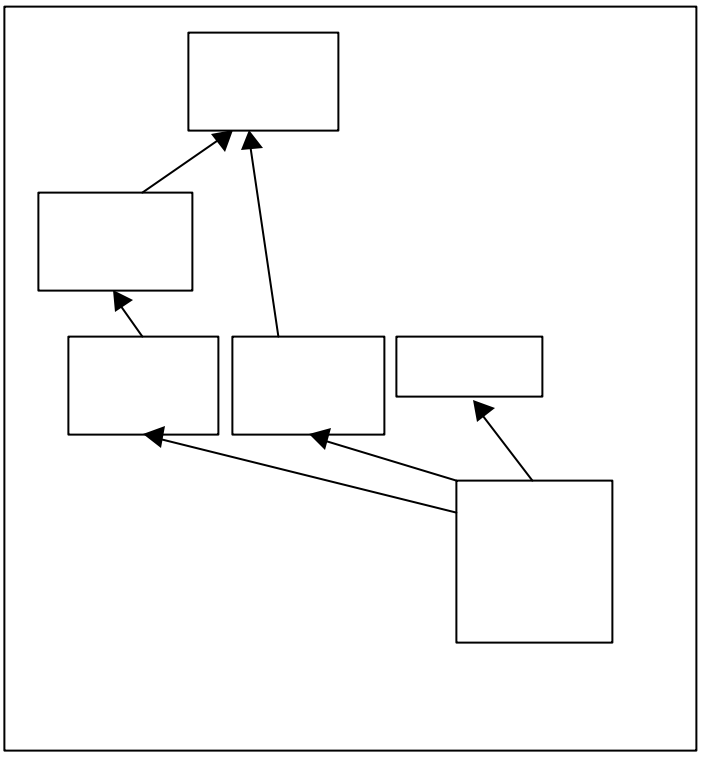
4.0. DESCRIPTION OF THE COMPONENTS USED IN THE EXCHANGE

4.1. The PSL Ontology

4.1.1. Initial Status of the PSL Ontology

An ontology is a lexicon of terminology along with a specification of the meaning of this terminology. Within the PSL Ontology, the meaning of terms is specified using KIF (Knowledge Interchange Format). Briefly stated, KIF is a formal language developed for the exchange of knowledge among disparate computer programs. KIF provides the level of rigor necessary to unambiguously define concepts in the ontology, a necessary characteristic to exchange manufacturing process information using the PSL Ontology.

The PSL Ontology has two major components -- the axioms of PSL-Core, and sets of extensions.



PSL-Core is used to specify the semantics of the primitives in the PSL Ontology. Primitives are those terms for which we do not give definitions; rather, we specify sentences that constrain the interpretation of the terms. There are three basic classes and four basic relations in the ontology of PSL-Core. The classes are OBJECT, ACTIVITY, and TIMEPOINT, and the relations PARTICIPATES-IN, BEFORE, BEGINOF, and END OF. ACTIVITIES, TIMEPOINTS (or "POINTS," for short), and OBJECTS are collectively known as entities, or things. These classes are all pair-wise disjoint.

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 require the lexicon of the second module. PSL-Core is therefore intended to be used as the basis for defining terminology of the extensions in the PSL Ontology.

Figure 2 illustrates the modules in PSL at the beginning of the pilot implementation; intuitively, they are the concepts required to define the terminology of the simplified process planning domain. There are four extensions to PSL-Core: Ordering Relations, Resource Roles, Processor Actions, and Resource Paths. The arcs in the diagram illustrate direct logical dependencies – if there is an arc from one module to another, then there exists a term in the second module which uses a term defined in the first module. Thus, the definition of the ordering relations and resource roles depends only on PSL-Core. The definition of processor actions uses resource roles, but not any ordering relations. Finally, resource paths are partially ordered sets of processor actions through which material resources flow; hence, the definition depends on both ordering relations and processor actions.

4.1.2. Concepts Introduced in ProCap

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 textual representation based upon the EPIF (Enhanced Process Interchange Format) [KNO95] 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

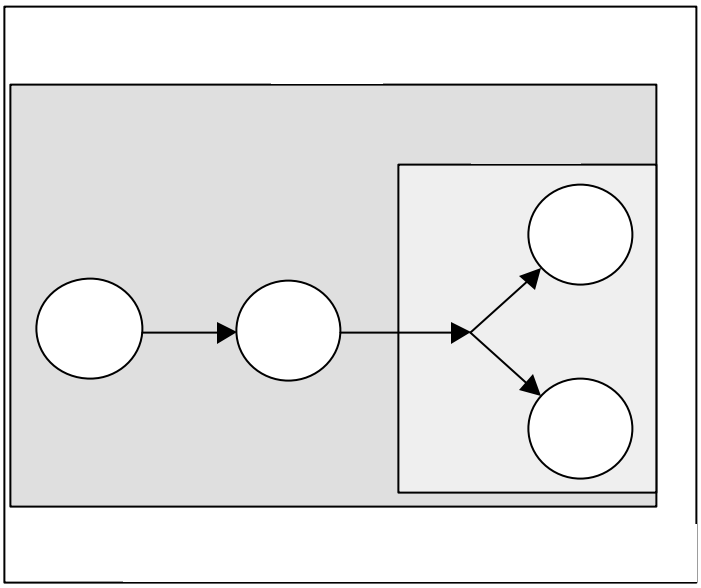
UOB's (Units of Behavior) are IDEF3's most fundamental building blocks that are used to represent activities. Furthermore, IDEF3 distinguishes UOB's (generic activities), UOB-uses (occurrences of UOB's in particular IDEF3 schematics), and UOB activations (collections of instances of UOB-uses that satisfy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

Junctions

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 [MAY95].

Links

The three generic types of links in IDEF3, namely, simple precedence links, constrained precedence links, and relational links are used 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 [MAY95].



4.1.3. ProCap-Related Extensions to the PSL Ontology

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.

Within the PSL splitting extension, three new terms, OR_SPLIT (representing a branch in which one or more ordering decisions are made from a set of choices**)**, AND_SPLIT (representing a branch where all ordering decisions are selected from a set of choices) and XOR_SPLIT, (representing a branch in which only one ordering decision is made from a set of choices) are introduced, representing the A. The OR_SPLIT (P1, J1) is shown in Figure 3.

Also introduced were two additional terms, SYNC_START and SYNC_FINISH**.** The SYNC_START term specifies that all subactivities within an activity start at the same time. The term SYNC_FINISH specifies that all subactivities within an activity finish at the same time.

4.1.4. Concepts Introduced in ILOG

ILOG Scheduler consists of an extensible library of C++ classes and functions that implement scheduling concepts such as activities and resources [ILO98]. 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.

There are three main classes within ILOG Scheduler:

- IlcActivity
- IlcResource
- IlcSchedule

An instance of the class IlcSchedule is an object that 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 that 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 Scheduler: endsAfter, endsAfterEnd, endsAfterStart, endsAt, endsAtEnd, endsAtStart, endsBefore, startsAfter, startsAfterEnd, startsAfterStart, startsAt, startsAtEnd, startsAtStart, and startsBefore.

ILOG Scheduler 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 that are simply required by activities (which are specified by the class IlcDiscreteResource) and resources that are provided by activities (which are specified by the class IlcReservoir).

4.1.5. ILOG-Related Extensions to the PSL Ontology

At the beginning of the pilot implementation of PSL, there were no extensions capable of defining concepts completely such as ILOG temporal or resource constraints. It was therefore necessary to design new extensions within PSL containing terminology whose definitions captured correctly and completely the intuitive meaning of the ILOG Scheduler concepts. The semantics of the ILOG classes and member functions which were used for extending PSL were gleaned from the Reference Manual for ILOG Scheduler 4.3 [ILO98].

Resource capacity constraints were already captured in the Resource Requirements Theory of PSL at the time of the pilot implementation. However, the biggest hurdle was 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 define the class of sets of resources that themselves behave as resources
- Resource Set-based Activities, which define classes of activities that use resource sets
- Substitutable Resources, which make the distinction between sets of arbitrary resources and sets of resources that can be substituted for others in an activity (e.g., the set of carpenters in a house construction activity)
- Homogeneous Sets, which define different classes of substitutable resources
- Resource Pools, which are equivalent to discrete capacity resources within ILOG Scheduler
- Inventory Resource Sets, which are equivalent to reservoirs within ILOG Scheduler
- Reasoning about Fluents, which define the notion of state within PSL

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

The classes of INTERVAL and BREAKABLE_ACTIVITIES within ILOG Scheduler 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.

4.2. IDEF3 to PSL (KIF) translator

4.2.1. Brief description

The PSL representation of an IDEF3 schematic is a set of KIF sentences. 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 also be seen as providing 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 [KNO95]. For a complete account of the compilation rules used, please refer to [CIO98].

Once the compilation rules were written, implementing the translator was a trivial task. The compilation rules were written as lisp macros, and the translator itself just expands those macros for all the forms of the IDEF3 file and stores the results in the PSL file.

The translator was later provided with a KIF expression simplifier. This simplifier was used to convert from the higher level KIF expressions generated by the compilation rules, that may contain a form of typed quantifiers and polymorphic operators, to a simple syntax form that uses typeless quantifiers and standard operators. A KIF-to-Prolog extension that was needed for subsequent translation steps was also included. This allowed the translator to generate three output formats for an IDEF3 file, that is, PSL terminology using KIF syntax, PSL terminology using the simplified KIF syntax, and PSL terminology using Prolog syntax.

4.2.2. Issues Faced and How They Were Resolved

In writing the compilation rules, there were three issues for which decisions had to be made as to how to solve them:

- how to encode IDEF3 junctions into PSL
- how to encode the type-instance relationships for objects and activities
- what was the right abstraction level in PSL at which to do the translation?

With respect to the first issue, IDEF3 junctions were mapped to PSL complex activities. Components of this were the activities after the junction (for Fan Out IDEF3 junctions), before the junction (for Fan In IDEF3 junctions), and the junction itself. In 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.

With respect to encoding the type-instance relationships for objects and activities, 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. For the moment, these predicates are defined in terms of PSL's occurrence predicate.

With respect to determining the right abstraction level in PSL at which to do the translation, it was determined that 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 (see Section 4.1) and the compilation rules were written in terms of those concepts.

4.3. KIF to HTML translator

4.3.1. Brief description

For demonstration purposes, KIF files were marked with HTML tags. This provided two benefits. Both definitions and uses were highlighted. All uses were hyperlinked to the definitions. For example, clicking on the word "during" brought up the KIF definition for "during." As the KIF data were spread across many files, this HTML markup provided the ability to follow a chain of definitions merely by clicking on anything unknown to the user.

4.3.2. Issues faced and How They Were Resolved

For simplicity, the markup software had no deep knowledge of KIF but merely applied a series of textual regular expression substitutions, replacing all words with HTMLized replacements. Two passes were used, one to build a diction of definitions and link destinations, and a second pass to decide whether to provide a use- or definition-style markup - or none at all if the word was unknown.

The only substantive implementation issue was related to speed. The markup software was written in Tcl [TCL94] and the first version was extremely slow; enough so that we would not be able run it real time for the demonstration. KIF input with 35000 words took approximately 20 minutes of processing (Sparc 10). After hotspot analysis, the software was rewritten - still in Tcl - with the same 35000-word input finally taking 20 seconds. The difference was using Tcl's built-in primitives such as *regexp* and *subst* instead of a manual string manipulation and sorting.

4.4. Object oriented presentation of PSL Ontology

4.4.1. Brief description

Once ProCAP's IDEF3 representation is converted to PSL (i.e., KIF), the user may be interested in looking at the representation to ensure that what came out of the translation is truly what they expected. However, since the native form of the neutral representation is KIF (a representation that many users would not feel comfortable looking at if they were not already familiar with the language), the user has the option to view the information in an object-oriented representation. This object-oriented representation was developed specifically for the PSL project. A snapshot of part of this representation is shown in

Figure 4. This is one of many representations that can be used to either textually or graphically display the information captured in PSL. There are currently other efforts within PSL that are creating mappings from the PSL semantic concepts to both the eXtensible Markup Language (XML) [W3C98] and the EXPRESS representation [ISO94].

4.4.2. Issues Faced and How They Were Resolved

During the creation of the object-oriented representation of PSL, many issues were faced. Some of the most important issues focused on ensuring that the representation:

- is easily understandable yet powerful enough to represent all concepts within PSL
- can capture data at multiple levels of abstraction
- is mapped directly back to the PSL ontology
- is extendable to be able to handle new process-related concepts as they arise
- introduces as few predefined terms as possible (to ensure extensibility)

There are still many issues remaining to be worked out with the representation of PSL, including whether or not the current object-oriented representation is going in a promising direction or whether we should take a different direction. For the purpose of this pilot implementation, we physically modeled the scenario in an HTML-like format and hard-linked it into the tool that ran the pilot implementation. This approach allowed us to work around the challenge of automatically generating this representation from the information within the PSL ontology (i.e., writing a KIF to object-oriented presentation translator).

Class Foundry
Annotation: [Description: "A description of the Foundry class for the CAMILE process"]**;**
Ontology: [Name: Gruninger, Version: 0.5] **;**
isWithinOperatingHours**: Condition : :**
isCurTimeInRange(CurrentTime, ["8:00", "24:00"]),
isCurDayInRange(CurrentTime, ["Monday", "Friday"]);

MaxOvertime**: Values: :**
2; // Max overtime allowed

PreventiveMaintenance**: ProcessSteps :**
isDueforPM(StartTime) **:**
DoPreventiveMaintenance()**;**
EndClass

Figure 4: Example Object-Oriented Presentation of PSL

This challenge will be addressed in future pilot implementations.

4.5. PSL (KIF) to ILOG translator

4.5.1. Brief description

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 translation 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 Scheduler is a mapping from KIF sentences to C++ code specifying class definitions and/or instances of ILOG Scheduler classes. This code can be compiled and executed to generate schedules based on the activities, resources, and constraints specified within the file.

4.5.2. Issues Faced and How They Were Resolved

The development of the ILOG/PSL translator faced three major issues:

- Specification of ILOG Scheduler ontology
- Implementation of semantic translator between PSL and ILOG Scheduler
- Implementation of syntactic translator between KIF and C++

The first challenge in the development of the semantic translator was the specification of the ILOG ontology, particularly given the object-oriented representation of ILOG Scheduler. Since ILOG Scheduler provides a set of object class definitions, we can consider the member functions of the objects to be representations of relations in the underlying ILOG ontology.

For each member function of an ILOG class, there corresponds a relation within the ILOG ontology, and for every relation in the ILOG ontology, there exists either an ILOG class or a member function of an ILOG class.

Using the translation definitions, the semantic translator between PSL and ILOG Scheduler was implemented. The input and output of the semantic translator is a set of KIF sentences; terminology in one ontology is simply substituted by their translation definitions.

The major challenge faced by the syntactic translator was understanding the relationship between KIF sentences and C++ objects. Ground KIF sentences (i.e. KIF sentences with no quantifiers or variables) within PSL are translated into instances of ILOG Scheduler classes. Quantified KIF sentences (i.e. KIF sentences containing quantifiers) within PSL are translated into class definitions within ILOG Scheduler. Conversely, instances of ILOG Scheduler classes are translated into ground KIF sentences and class definitions within ILOG Scheduler are translated into quantified KIF sentences.

Syntactic translation of ground atomic KIF sentences into instances of ILOG Scheduler classes is straightforward -- constructor functions within ILOG Scheduler are used to generate each slot value of an ILOG Scheduler instance. The actual implementation uses Prolog to extract the relations satisfied by activities and resources within the PSL theory. For each activity, the associated KIF relation is represented as a slot on the ILOG Scheduler instance that represents the activity; the appropriate accessor function assigns the object that is the argument of the KIF relation as the value of the slot on the ILOG Scheduler instance.

Syntactic translation of instances of ILOG Scheduler classes into ground KIF sentences is also straightforward -- within a C++ function, accessor functions within ILOG Scheduler are used to generate a sentence for each slot value of an ILOG Scheduler instance.

Syntactic translation of quantified KIF sentences requires a correspondence between the classes of sentences used within the PSL theory and the underlying sentences used to represent C++ classes. This presents a series of challenges:

- There may be logically equivalent sentences with different syntactic forms. In this case, a syntactic translator would need to be able to recognize and manipulate each of these different forms in order to translate to the correct C++ class definition.
- Translating arbitrary classes of KIF sentences into a particular set of C++ classes is not guaranteed. In general, the class of KIF sentences which can be directly represented as C++ classes is restricted to sentences in which the class variable is universally quantified, and the variables in the member functions are existentially quantified. That is, for every object that is an instance of a class, there exists another object that is equal to the value of the member function. To specify other classes of KIF sentences within C++ requires explicit definition of constraints as C++ classes themselves, such as IlcConstraint in ILOG Scheduler. However, even these represent restricted classes of KIF sentences.

Syntactic translation of ILOG Scheduler class definitions into KIF sentences was not implemented in the pilot project.

4.6. Web-based tool

4.6.1. Brief description

A World Wide Web (WWW) site effectively demonstrated the PSL-as-interlingua concept. The web pages both explained the concept and demonstrated the transformations involved. In practical terms, the user selected a set of definitions and then clicked on the given transformation, doing this through several transformations. At each step, the user downloaded or examined the resulting data or upload replacement or additional data.

The web site provided a convenient demonstration platform. It was relatively portable, being able to run on any WWW-connected Windows-based computer. The browser interface transparently melded multiple processes that run on the client and on the server host.

4.6.2. Issues Faced and How They Were Resolved

Most of the issues were typical integration issues - we face them in many projects. For example, some of the software ran only on Windows, while some of the software ran only on UNIX. Unifying this is tricky. Even worse, some software didn't exist. Real-time response (at least for existing software) was very desirable. An early integration issue, with important design consequences, was the choice of a front end - whether to use a browser or a traditional programmable graphical user interface such as Tk [TCL94]. A browser could provide uniformity and portability. A programmable GUI would provide vastly more flexibility. The choice of these inevitably dictated the interaction style of the demo.

To minimize effort, a browser was used to provide a basic infrastructure for the demo. As mentioned earlier, this limited flexibility and will be revisited for future demonstrations. However, for a walk-through-style demonstration, a browser provides a suitable interface. We had originally hoped to produce a portable demonstration, however as components of the software arrived to be integrated, we discovered some pieces that would only run on the PC and some that would only run on UNIX. The PC software required direct access to a graphics display. In those cases, the browser was directed to simply turn control over to the Windows executables. UNIX software ran via common gateway interface (CGI) scripts on a local Web server. The result appeared as a unified collection of tools, all accessible through a web browser. The web pages themselves were all generated dynamically using the Tcl language and cgi.tcl, which provided the ability to easily customize all pages simultaneously or independently.

5.0. CONCLUSION

This paper highlights some of the issues that arose during the first PSL pilot implementation. Although many of these issues were resolved, this exercise helped bring to light many of the challenges that exist in using an ontology-based neutral exchange representation for the integration of manufacturing applications.

This is the first of a series of pilot implementations in which PSL was/will be used to exchange process information among manufacturing software applications. Each pilot implementation will include software packages that focus on different areas within manufacturing (e.g., process planning, scheduling, simulation, workflow, execution, etc.). By focusing on a wide variety of areas, we will not only be able to expand the PSL to capture the concepts represented in each area, but we will also be able to help identify what types of process-related concepts are important to exchange between any two areas within manufacturing (e.g. what aspects of a process plan are of most interest to a scheduling application).

ACKNOWLEDGMENTS

This project is funded by NIST's Systems Integration for Manufacturing Applications (SIMA) Program. Initiated in 1994 under the federal government's High Performance Computing and Communications effort, SIMA is addressing manufacturing systems integration problems through applications of information technologies and development of standards-based solutions. With technical activities in all of NIST's laboratories covering a broad spectrum of engineering and manufacturing domains, SIMA is making information interpretable among systems and people within and across networked enterprises.

REFERENCES

[CIO98] Ciocoiu, Mihai, "Translating IDEF3 to PSL," University of Maryland Computer Science Technical Report 98-63, 1998.

[GEN92] Genesereth, Michael R. and Fikes, Michael R., "Knowledge Interchange Format Version 3.0 Reference Manual," Technical report, Logic Group, Stanford University, CA., 1992.

[ILO98] "ILOG Scheduler 4.3 Reference Manual," June 1998.

[ISO94] ISO 10303-11: 1994, Product data representation and exchange: Part 11: EXPRESS language reference manual

[KNO95] 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.

[KNO98] Knowledge Based Systems, Inc., "ProCAP Automated Process Modeling for Windows User's Manual," 1998.

[KNU97] Knutilla, A., et al, "Process Specification Language: Analysis of Existing Representations," NISTIR 6133, National Institute of Standards and Technology, Gaithersburg MD, October 1997.

[MAY90] Mayer, Richard J., Menzel, Christopher P., Painter, Michael K., deWitte, Paula S., Blinn, Thomas, and Perakath, Benjamin, "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, 1990.

[McK91] McKay, Kenneth N., Moore, John B., "CAM-I (Consortium for Advanced Manufacturing International) Report: Intelligent Manufacturing Management Program: State of the Art Scheduling," Survey 06-23-91, Technical Report R-91-IMM-01, 1991.

[POL98] Polyak, Stephen T. and Aitken, Stuart, "Manufacturing Process Interoperability Scenario," Technical report, AIAI-PR-86, Artificial Intelligence Applications institute (AIAI), Edinburgh, 1998.

[POL982] Polyak, Stephen T., Gruninger, Michael , Lee, Jintae and Menzel, Chris, "Applying the Process Interchange Format (PIF) to a Supply Chain Process Interoperability Scenario," 1998.

[SCH96] Schlenoff, C., et al., "Unified Process Specification Language: Requirements for Modeling Process," NISTIR 5910, National Institute of Standards and Technology, Gaithersburg MD, September 1996. (also available at http://www.nist.gov/psl/)

[TCL94] Ousterhout, John K., "Tcl and the Tk Toolkit," Addison-Wesley, ISBN 020163337X, 1994.

[W3C98] W3C Architecture Domain, "Extensible Markup Language (XML)," http://www.w3c.org/XML/, November 17, 1998.