ONTOLOGY MAPPING BETWEEN PSL AND XML-BASED STANDARDS FOR PROJECT SCHEDULING

Jinxing Cheng[1], Pooja Trivedi[2] and Kincho H. Law[3]

ABSTRACT

Many ontology standards, such as STEP, ifcXML, aecXML and PSL, have been proposed for the A/E/C industries. Different ontologies exist and they have many advantages for information exchange within specific domain applications. However, the existence of different ontologies may cause many interoperability issues. For example, while PSL, ifcXML and aecXML all have provisions related to project management and scheduling information, an aecXML-compatible or ifcXML-compatible application cannot directly exchange information with a PSL-compatible application. In this paper, we address this information exchange problem by building translators among PSL, ifcXML and aecXML in the project scheduling domain. We examine the ontology mapping among PSL, ifcXML and aecXML, and compare their expressive power. Using the parsers and translators developed, we successfully demonstrate the information exchange among PSL, ifcXML and aecXML for various projects.

KEY WORDS

Process Specification Language, PSL, aecXML, ifcXML, ontology, information exchange

[1] PhD Student, Civil and Environmental Engineering Department, Stanford University, Stanford, CA 94305-4020, email: cjx@stanford.edu

[2] Graduate Student, Computer Science Department, Stanford University, Stanford, CA 94305-4020, email: ptrivedi@stanford.edu

[3] Professor, Civil and Environmental Engineering Department, Stanford University, Stanford, CA 94305-4020, email: law@cive.stanford.edu

1. INTRODUCTION

Ontology standards play an important role to support interoperability among applications. For the last two decades, there have been many ontology standards proposed and adopted by various industries. In this paper we examine the interoperability issues between Process Specification Language (PSL) (Schlenoff et al. 2000) and XML-based standards, such as ifcXML (Liebich 2001) and aecXML (IAI 2002), for the project scheduling task. Both PSL and XML-based standards have ontologies defined for project management and scheduling applications.

The different ontology standards provide users various choices according to their specific needs. However, the different standards may hinder interoperability since different applications may opt to adopt different ontology standards. For example, an ifcXML-compatible or aecXML-compatible application cannot directly exchange information with a PSL-compatible application. To address this problem, we evaluate PSL, ifcXML and aecXML focusing on ontology mapping between these standards for project scheduling applications.

2. INTRODUCTION TO PSL

Process Specification Language (PSL) was initiated by NIST (National Institute of Standards and Technology) and is emerging as an international standard for the manufacturing industry. The goal of PSL is to create a process interchange language, which could be used to exchange process information among different applications. PSL is based on first order logic and situation calculus. PSL ontology includes a central core, an outer core and a set of extensions. The PSL core consists of object, activity, activity occurrence and timepoint as the primitive classes. The PSL outer core includes a small set of extensions, which are generic and pervasive in their applicability, such as subactivity extension, activity occurrence extension and states extension. The PSL extensions include a set of ontology modules motivated by different manufacturing applications.

PSL is based on a formal language KIF (Knowledge Interchange Format), which is a language designed for use in the interchange of knowledge among disparate computer systems (Genesereth and Fikes 1992). KIF supports declarative semantics, and is logically comprehensive. When combined with domain specific ontology, KIF is a powerful knowledge representation language.

A PSL file consists of a set of logic sentences. The following shows examples of PSL sentences:

(occurrence-of occ1 a1)
(before t1 t2)
(subactivity-occurrence occ1 occ2)

The first sentence *(occurrence-of occ1 a1)* means that activity occurrence *occ1* is a particular instance or occurrence of an activity *a1*. In the second sentence, two time points *t1* and *t2* are ordered, where *t1* is less than *t2*. The third sentence specifies that an activity occurrence *occ1* is a subactivity occurrence of an activity occurrence *occ2*.

3. INTRODUCTION TO IFCXML AND AECXML

XML (Extended Markup Language) is a meta-markup language that consists of a set of rules for creating semantic tags used to describe data (Young 2001). XML provides a mechanism to describe an object as a hierarchy of elements. Due to the popularity of XML, many efforts have been invested to propose XML schemas as ontology standards in the construction and manufacturing industry as well as for business applications. In the A/E/C domain, ifcXML and aecXML are the most popular XML schemas.

There exist tools that can automatically translate ifcXML from IFC/EXPRESS source (Liebich 2001). IfcXML enables the exchange of IFC data in XML formats. IFC (Industry Foundation Classes) is a data representation standard primarily for architectural and construction product data (ISO 1994). There have been efforts to extend IFC to include cost estimating and project management information (Thomas et al. 1999).

The ifcXML schema is a fairly extensive document with over 400 pages. IfcXML deals not only with geometry and product data, but also intends to support life cycle project information including architecture, HVAC, construction and facility management.

AecXML was initially proposed by Bentley Systems in 1998, and is now being part of the IAI (International Alliance of Interoperability). AecXML provides XML-based schemas to describe information specific for data exchange among participants involved in the design, construction, and operation of buildings, plants, infrastructure, and facilities (ISO 2002). The schemas currently under development include:

- COS (Common Object Schema)
- Infrastructure
- Structural
- FM (Facility Management)
- Procurement
- Project Management
- Plant
- Building Performance

There are many working groups within each domain to define XML schemas for specific applications. For example, the Project Management group is working on ApplicationForPayment schema, contract schema and other schemas. Table 1 summarizes the current state of the schemas posted on the URL site (http://www.aecxml.org).

Table 1: Schemas available in draft version

Working group	**Schemas available in draft version**
Infrastructure	LandXML
Project Management	ApplicationForPayment Contract
Plant	EquipmentDesignRequest
Building Performance	QbXML

In this work, we focus our research in the project scheduling and resource allocation area. While ifcXML schema exists, the “aecXML” schema[1] employed in this study is the draft

[1] The schema employed here is a version recommended by Professor Thomas Froese of University of British Columbia during his sabbatical at Stanford University.

version developed by Primavera Systems in 1998. The schema is focused on project scheduling, and was based on the Primavera Project Planner (P3)^{TM} software.

4. ONTOLOGY MAPPING BETWEEN PSL AND XML

4.1 MAPPING CONCEPTS BETWEEN PSL AND XML

The first task is to map the terms related to project scheduling defined in PSL, ifcXML and aecXML.

In a typical construction project, a project schedule consists of a set of activities and the dependency relationships among the activities. Construction activities can generally be categorized into one of three types: production activities, procurement activities and administrative activities. Each activity has certain attributes associated with it, such as start date and duration.

Dependency relationships describe the constraints defining the order in which the activities must occur to complete the project (Gould 2002). There are four typical dependency relations: Finish to Start, Finish to Finish, Start to Start, Start to Finish. Figure 1 depicts the dependency relationships and their respective definitions. For examples, the “Finish to Start” relationship between activity A and activity B means that B does not begin until A completes, and the “Finish to Finish” relationship means that A need to complete before B does.

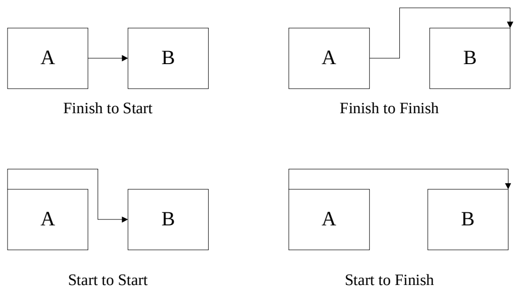


Figure 1: Dependency Relationships Among Activities

In PSL, there are four basic classes: object, activity, activity occurrence and timepoint. Each activity in a project schedule can be roughly mapped into an activity occurrence in PSL, while time point is used to specify the beginning point and the end point of an activity occurrence. In PSL extensions, PSL provides some terms to describe the dependency relationship among activities. For example, the terms *before-start* and *before-start-delay* in PSL correspond to the “Start to Start” relationship in a project schedule. The PSL sentence *(before-start occ1 occ2 occ3)* specifies that the beginning time point of *occ1* is earlier than the beginning time point of *occ2*, while *(before-start-delay occ1 occ2 occ d)* means that *occ2* begins at least *d* time points after *occ1* begins.

For ifcXML, each activity in a project schedule can be roughly mapped to a *Task* element. The scheduling information about an activity is expressed in the *WorkSchedule* and *ScheduleTimeControl* elements. The *WorkSchedule* element holds the overall scheduling information, such as the start time and duration, while the *ScheduleTimeControl* element holds further descriptions of scheduling information, such as *actualStart*, *earlyStart*, *lateStart* and *scheduleStart*. The *RelSequence* element is used to express the dependency relationships among activities.

The aecXML schema follows closely the Primavera P3^{TM} tool. Primavera Project Planner (P3)^{TM} provides advanced project scheduling functions. The mapping from a project schedule in P3^{TM} to aecXML schema is relatively straightforward. In the aecXML schema, there are activity, dependency and timepoint elements corresponding to the concepts in a project schedule.

4.2 EXPRESSING PSL IN XML-BASED SCHEMAS

IfcXML and aecXML are based on a meta-markup language (XML), while PSL is based on a formal logical language (KIF). XML has only limited representation capability to represent constraints and rules. Since PSL is built on first order logic, it is more expressive than XML-based schemas. Not all PSL sentences can be directly translated into XML, and some of the PSL logical statements are difficult to translate into ifcXML and aecXML.

Most PSL sentences that deal with basic facts can be expressed in XML format, for examples:

- *(Beginof occ1 t1)*
  This PSL sentence can be translated into ifcXML by creating a task *occ1* and an associated *WorkSchedule* element, where the value of *startTime* attribute is *t1*.
- *(before-start occ1 occ2 occ3)*
  For this sentence, we can create two tasks *occ1* and *occ2*, and define *occ3* as the task of the whole project. We can then use *RelSequence* element to express the relationship between *occ1* and *occ2*.

In general, the PSL sentences dealing with logic rules will be difficult to translate into XML directly. Some example situations are listed below:

- PSL sentences with existential or universal logic token (*forall* and *exists*), for examples:
  (forall $(v_1 \dots v_n)$ *(=>* $\psi\ \theta$*))*
  (exists $(v_1 \dots v_n)$ *(and* $\psi_1 \dots \psi_m\ \theta$*))*
- PSL sentences with deduction or logically equivalent tokens (=> and <=>), for examples:
  (=> occ1 occ2)
  (<=> occ1 ooc2)
- Some PSL sentences with logic relations, for example:
  (during occ1 occ2 occ3)
  The PSL sentence *(during occ1 occ2 occ3)* means that the beginning and ending time points of *occ1* are between the beginning and ending time points of *occ2*, and both *occ1* and *occ2* are subactivity occurrences of *occ3*. In ifcXML and aecXML, however, we do not have corresponding elements to express this logic relation. Consequently, it is difficult to translate this PSL sentence into ifcXML or aecXML.

One solution to this problem is to embed the whole PSL logic sentence as an XML attribute. For example, we can express the example PSL logic sentences in an XML structure as:

<LogicRules>

<PSLsentence ID="01" rule="(forall (v_1 ... v_n) (=> ψ θ))">

<PSLsentence ID="02" rule="(exists (v_1 ... v_n) (and ψ_1 ... ψ_m θ))">

</LogicRules>

Embedding PSL sentences in an XML structure does not provide the meaning of the rules, since most XML parsers do not support rule processing. Nevertheless, the sentence can be exchanged verbatim between applications, if necessary.

On the other hand, we can always translate the scheduling information in ifcXML or aecXML files into PSL, as long as the PSL ontology covers all the concepts in the ifcXML and aecXML schemas.

5. TRANSLATION BETWEEN PSL AND XML

5.1 BUILDING A PSL PARSER

Presently, no generic PSL parser exists. Our first task was to build a PSL parser for the project scheduling information in Java. The basic process of a PSL parser works as follows: When the PSL parser reads a PSL logic sentence, it calls a corresponding function, which in turn parses the information in the sentence according to the PSL syntax. The information will be stored as objects and vectors, so that other functions and applications can retrieve and use the information. Figures 2 shows the example code segment from the PSL parser.

```
private void parseDependency(String line, String type){
      String id,id1,id2;      float  ww;
      StringTokenizer st = new StringTokenizer(line, " \t\n\r\f()");
      st.nextToken();
      id1 = st.nextToken();   id2 = st.nextToken();
      id = "depend" + depid;
      dependency dep = new dependency(id, id1, id2);
      dep.setType(type);
      dependencies.addElement(dep);
      depid = depid + 1;
}
```

Figure 2: Sample Code of the PSL Parser

One simplification we made in the PSL parser is that all PSL sentences are expressed as relations rather than functions. PSL syntax is based on KIF. In KIF, each function can be expressed as a relation. For example, the following two PSL sentences are equivalent:

(begin-of A 17)

(= (begin-of A) 17)

The difference is that in the first sentence the term *begin-of* is used as a relation, while in the second sentence the term *begin-of* is used as a function. In KIF, each function has a unique value; for example, in the second sentence, the value of the function *(begin-of A)* is 17. In contrast, the value of a relation is either true or false; furthermore, relations can have disagreement on the last element. For examples, the relations *(before t1 t2)* and *(before t1 t3)* differ. As a result, every function is a relation, while not every relation can be expressed as a function. Therefore, using relations is usually more convenient than using functions and minimizes unnecessary confusions and complexities in implementing the PSL parser.

The PSL parser developed so far can only parse predefined PSL terms. We are currently investigating the possibility to build a generic PSL parser in Java using a parser generator. A parser generator is a tool that reads a grammar specification and converts it to a Java program that can recognize matches to the grammar. Java Compiler Compiler (JavaCC) is one of such popular parser generators for use with Java applications (SUN 2002). We are currently building a generic PSL parser using JavaCC.

5.2 BUILDING XML PARSERS

To build parsers for ifcXML and aecXML files, we use Apache Xerces Java Parser (Apache 2002). Xerces Java Parser provides XML parsing and generation, and implements the W3C XML and DOM (Document Object Model) standards, as well as the SAX (Simple API for XML) standard. The parsers are highly modular and configurable.

The basic processes of the ifcXML and aecXML parsers are essentially the same. Both parsers use Xerces API to parse the information from XML files, and store the information as a linked list. Other functions and applications can iterate over the list to retrieve the information. The difference between the ifcXML parser and the aecXML parser lies in that ifcXML file and aecXML file use different tags and tree structures to represent a project schedule. Figures 3 and 4 show the sample code segments from the ifcXML parser and the aecXML parser respectively.

```
public LinkedList getRelSequenceGroup(){
      LinkedList RelList = new LinkedList();
      NodeList nodes =document.getElementsByTagName("RelSequence");
      for(int i=0;i<len;i++){
            Element e=(Element)nodes.item(i);
            String[] attr = new String[5];
            attr[0] = (String)e.getAttribute("id");
            … …
            RelList.add(attr); }
      return RelList;
}
```

Figure 3: Sample Code of the ifcXML Parser

```
public LinkedList getDependencies(){
      LinkedList DependencyList = new LinkedList();
      NodeList nodes = document.getElementsByTagName("dependency");
      int len = (nodes != null) ? nodes.getLength() : 0;
      for(int i=0;i<len;i++){
          Element e=(Element)nodes.item(i);
          String[] attr = new String[7];
          attr[0] = (String)e.getAttribute("dependencyid");
          … …
          DependencyList.add(attr); }
      return DependencyList;
}
```

Figure 4: Sample Code of the aecXML Parser

5.3 TRANSLATION PROCESS

As shown in Figure 5, the translation process between PSL and XML is straightforward. To translate PSL files into XML files, first we use a PSL parser to parse the PSL logic sentences. We then map the terms in PSL into XML tags. Finally we construct the XML trees according to the ifcXML schema and aecXML schema, and output the file in the corresponding formats.

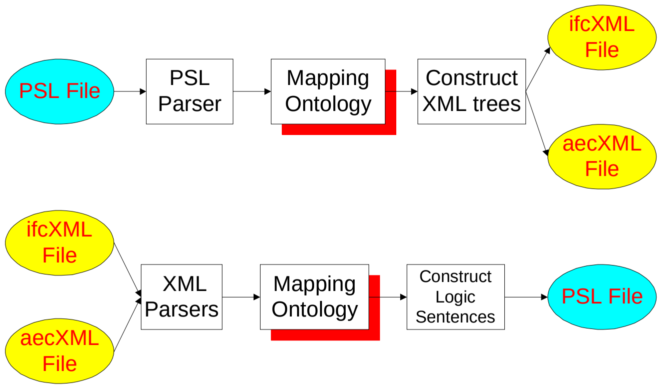


Figure 5: Mapping Process Between PSL and XML

The reverse process is similar. To translate ifcXML and aecXML files into PSL files, we use the ifcXML and aecXML parsers to parse the information from the XML files. We then map the XML tags into PSL terms, and output the PSL logic sentences according to PSL syntax.

6. EXAMPLES

6.1 EXAMPLE 1: SAMPLE PROJECT FROM VITE

We select a sample project from Vite^{TM} to test our translators among PSL, ifcXML and aecXML. Vite^{TM} is a project and organization modeling system designed to assist in developing organizational structures and identifying potential problems with project cost, time, or quality. As shown in Figure 6, the sample project is to design and fabricate a chip set for a new personal digital assistant (PDA) product. There are 12 activities in this project. Among the 12 activities there are three milestone activities: 'Start Project,' 'Ship Tapes to Foundry' and 'Fab, Test and Deliver.' The activity 'Design_Coordination' is to maintain the overall control of the project.

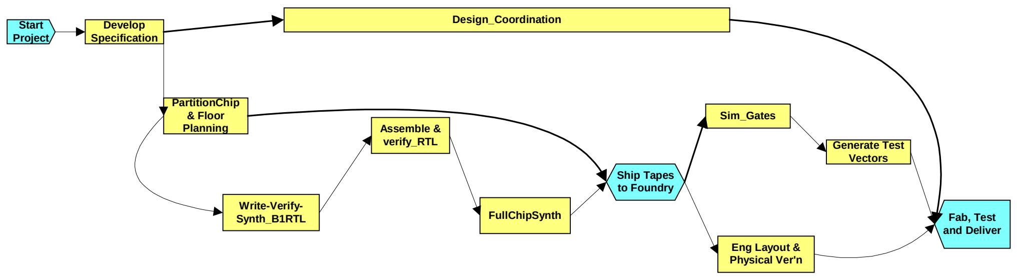


Figure 6: Original CPM Diagram in Vite

The scheduling information in the example project can be expressed in a PSL file. Figure 7 shows part of the PSL file, which includes a set of logic statements to describe activities and dependency relationships in the project schedule.

```
(and
       (project TUTO)
       (doc TUTO "TUTORIAL Project")
       (beginof TUTO 9/18/1998)
       (subactivity-occurrence ID100 TUTO)
       … …
)
(and
       (activity-occurrence ID100)
       (doc ID100 "Assemble and verify_RTL")
       (beginof ID100 12/17/1998)
       (duration-of ID100 18)
       (after-start ID100 ID170 TUTO)
       (after-start-delay ID100 ID170 TUTO 0)
)
```

Figure 7: Sample PSL File

Figure 8 shows part of the ifcXML file translated from the PSL file. As discussed earlier, a *Task* element in ifcXML maps to an activity in a project schedule. Thus the '*taskid*' attribute in ifcXML maps to the identifier of an activity occurrence in PSL. As a result, the *WorkSchedule* element is associated with the corresponding activity by using the same identifier as its identifier attribute. Similarly, the *RelSequence* element, which depicts the dependency relationships among activities, is associated with the predecessor and successor activities through its '*relatedProcess*' and '*relatingProcess*' attributes.

```
<WorkScheduleGroup>
     <WorkSchedule identifier="ID100" duration="18.0" freeFloat="0.0"
totalFloat="0.0"        startTime="12/17/1998" finishTime="1/4/99"/>
  </WorkScheduleGroup>
  <TasksGroup>
     <Task taskid="ID100" description="Assemble and verify_RTL"/>
     <Task taskid="ID700" description="FullChipSynth"/>
 </TasksGroup>
 <RelSequenceGroup>
     <RelSequence id="depend0" relatingProcess="ID100"
relatedProcess="ID170" timeLag="0.0"      sequenceType="after-start"/>
 </RelSequenceGroup>
```

Figure 8: Sample ifcXML file

Figure 9 shows part of the aecXML file translated from the PSL file. AecXML schema has most elements corresponding exactly to the concepts in a project schedule. Once we have mapped the PSL ontology to the concepts in a project schedule, translation between PSL and aecXML requires little extra efforts.

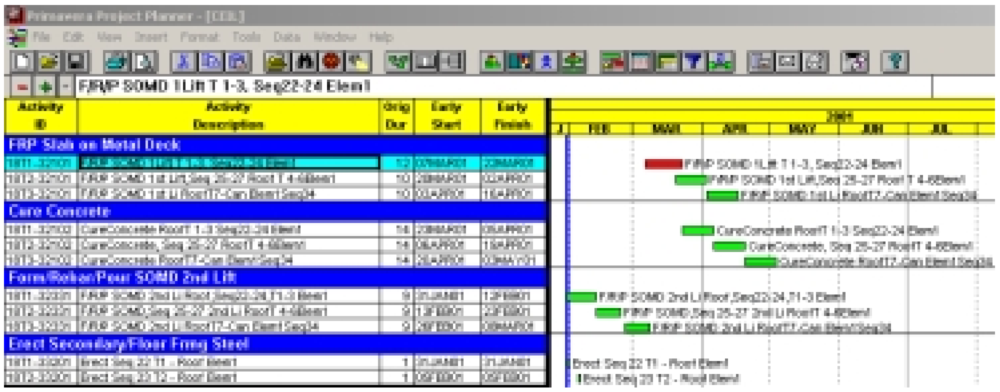
```
<activities>
 <activity activityid="ID100" description="Assemble and verify_RTL"
dependencyids="depend0" />
</activities>
<dependencies>
<dependency dependencyid="depend0" predecessor_activity_id="ID100"
successor_activity_id="ID170" relationship="after-start" lag="0.0" />
</dependencies><timepoints><timepoint timepointid="tp0"
activityid="ID100" atTime="12/17/1998" /> <timepoint timepointid="tp1"
activityid="ID100" atTime="1/4/99"/>
</timepoints><occurrences><occurrence activityid="ID100" begin="tp0"
end="tp1" duration="18.0" freefloat="0.0" totalfloat="0.0"
/></occurrences>
```

Figure 9: Sample aecXML file

As shown in Figures 7, 8 and 9, all information in the PSL file is successfully translated into ifcXML file and aecXML file, and vice versa.

6.2 EXAMPLE 2: MORTENSON CEILING PROJECT

To test the scalability of the prototype translators, the Mortenson Ceiling Project is employed to illustrate the ontology mapping process. The Mortenson Ceiling Project is a portion of the construction of the Walt Disney Concert Hall, built by Mortenson Construction, and designed by Frank O.Gehry & Associates. Figure 10 shows the schedule in Primavera Project Planner^{TM}. In this project, there are 191 activities and 459 dependency relationships.


Primavera Project Planner - [CEIL]
File Edit View Insert Format Tools Data Window Help
F/R/P SOMD 1Lift T 1-3, Seq22-24 Elem1

Activity ID	Activity Description	Orig Dur	Early Start	Early Finish
FRP Slab on Metal Deck				
1811-32101	F/R/P SOMD 1Lift T 1-3, Seq22-24 Elem1	12	07MAR01	20MAR01
1812-32101	F/R/P SOMD 1st Lift,Seq 25-27 Roof T 4-6Elem1	10	28MAR01	02APR01
1813-32101	F/R/P SOMD 1st Li Roof17-Can Elem1 Seq24	10	03APR01	16APR01
Cure Concrete				
1811-32102	CureConcrete RoofT 1-3 Seq22-24 Elem1	14	20MAR01	05APR01
1812-32102	CureConcrete, Seq 25-27 RoofT 4-6Elem1	14	06APR01	15APR01
1813-32102	CureConcrete RoofT17-Can Elem1Seq24	14	26APR01	03MAY01
Form/Rebar/Pour SOMD 2nd Lift				
1811-32201	F/R/P SOMD 2nd Li Roof,Seq22-24,T1-3 Elem1	9	31JAN01	12FEB01
1812-32201	F/R/P SOMD,Seq 25-27 2nd Li RoofT 4-6Elem1	9	13FEB01	23FEB01
1813-32201	F/R/P SOMD 2nd Li Roof17-Can Elem1 Seq24	9	26FEB01	08MAR01
Erect Secondary/Floor Frmg Steel				
1811-33201	Erect Seq 22 T1 - Roof Elem1	1	31JAN01	31JAN01
1812-33201	Erect Seq 23 T2 - Roof Elem1	1	09FEB01	09FEB01

Figure 10: Original Schedule in Primavera Project Planner

Using the PSL wrapper, we extract the scheduling information from Primavera Project Planner^{TM}, and convert it into a PSL file, as illustrated in Figure 11. There are more than 2000 logic sentences in the generated PSL file. Using the PSL to XML translators, we successfully translate the PSL file into ifcXML and aecXML files. Figures 12 and 13 show the corresponding ifcXML and aecXML files, respectively. This example demonstrates the robustness of the prototypes developed for information exchange between PSL and XML-based standards.

```
(and
       (activity-occurrence 1620-91101)
       (doc 1620-91101 "Hang Drywall@Studs (Dtl7050) - M1/3 Lvl 6 Elem1")
       (beginof 1620-91101 2/11/2002)
       (duration-of 1620-91101 10)
       (freefloat 1620-91101 0)
       (totalfloat 1620-91101 42)
       (after-end 1620-91101 1620-97101 CEIL)
       (after-end-delay 1620-91101 1620-97101 CEIL 1)
       (after-start 1620-91101 1630-91101 CEIL)
       (after-start-delay 1620-91101 1630-91101 CEIL 0)
)
… …
```

Figure 11: Sample PSL File

```
<WorkSchedule identifier="1620-91101" freeFloat="0.0" totalFloat="42.0"
startTime="2/11/2002" finishTime="2/21/102"/>
<Task taskid="1620-91101" description="Hang Drywall@Studs (Dtl7050) - M1/3
Lvl 6 Elem1" >
<RelSequence id="depend79" relatingProcess="1610-91101"
 relatedProcess="1610-97101" timeLag="1.0" sequenceType="after-start"/>
<RelSequence id="depend80" relatingProcess="1610-91101"
 relatedProcess="1620-91101" timeLag="0.0" sequenceType="after-start"/>
… …
```

Figure 12: Sample ifcXML File

```
<activity activityid="1620-91101" description="Hang Drywall@Studs (Dtl7050)
- M1/3 Lvl 6 Elem1" dependencyids="depend159,depend160"/>
<dependency dependencyid="depend159" predecessor_activity_id="1620-91101"
successor_activity_id="1620-97101" relationship="after-start" lag="1.0"/>
<dependency dependencyid="depend160" predecessor_activity_id="1620-91101"
successor_activity_id="1630-91101" relationship="after-start" lag="0.0"/>
<timepoint timepointid="tp102" activityid="1620-91101" description="begin
Hang Drywall@Studs (Dtl7050) - M1/3 Lvl 6 Elem1" atTime="2/11/2002"/>
<timepoint timepointid="tp103" activityid="1620-91101" description="end
Hang Drywall@Studs (Dtl7050) - M1/3 Lvl 6 Elem1" atTime="2/21/102"/>
<occurrence activityid="1620-91101" begin="tp102" end="tp103"
duration="10.0" freefloat="0.0" totalfloat="42.0"/>
… …
```

Figure 13: Sample aecXML file

7. CONCLUSIONS

Different ontologies exist for different application domains. IfcXML, aecXML and PSL are candidate ontology standards that could be employed by the construction industry. To achieve interoperability among different construction applications, we have developed translators for information exchange among PSL, ifcXML and aecXML and tested for a typical project schedule. We identify the PSL sentences, which could be difficult to translate into XML format. Our current research includes translating complex PSL logic sentences into ifcXML and aecXML structures. In addition, we are investigating the potential use of logic-based PSL for conflict resolution and consistency checking of a project schedule.

8. ACKNOWLEDGEMENTS

This work is partially sponsored by the Center for Integrated Facility Engineering at Stanford University, a Stanford Graduate Fellowship, and the Product Engineering Program headed by Dr. Ram D. Sriram at NIST. The Product Engineering Program gets its support from the NIST's SIMA (Systems Integration for manufacturing Applications) program and the DARPA's Radeo Program. The model of the Mortenson Ceiling Project is provided by Professor Martin Fischer and his research group at Stanford University. The authors would like to thank Professor Thomas Froese, University of British Columbia, for providing the aecXML schema and for his valuable comments and suggestions about the schema.

9. REFERENCES

Apache (2002), "Xerces Java Parser." The Apache Software Foundation, http://xml.apache.org/xerces-j/ (Accessed: 25 March 2002).

Froese, T., Fischer, M., Grobler, F., Ritzenthaler, J., Yu, K., Sutherland, S., Staub, S., Akinci, B., Akbas, R., Koo, B., Barron, A., and Kunz, J., (1999). "Industry Foundation Classes for Project Management-A Trial Implementation." *ITCON*, Vol. 4, 17-36.

Genesereth, M.R. and Fikes, R. (1992), "Knowledge Interchange Format Version 3.0 Reference Manual." Computer Science Department, Stanford University.

Gould E.F.(2002), *Managing the construction Process: Estimating, Scheduling, and Project Control*, Prentice Hall, 2002.

IAI (2002), "AecXML." International Alliance for Interoperability, http://www.aecxml.org (Accessed: 15 March 2002).

ISO (1994), 10303-1:1994, "Product data representation and exchange: Part 1: Overview and fundamental principles."

Liebich T. (2001), "XML schema language binding of EXPRESS for ifcXML." MSG-01-001(Rev 4), International Alliance of Interoperability, 2001.

Schlenoff, C., Gruninger M., Tissot, F., Valois, J., Lubell, J., and Lee, J.(2000), "The Process Specification Language (PSL): Overview and Version 1.0 Specification." NISTIR 6459, National Institute of Standards and Technology, Gaithersburg, MD.

SUN (2002), " Java Compiler Compiler(JavaCC) - The Java Parser Generator." Sun Microsystems, http://www.webgain.com/products/java_cc/ (Accessed: 25 March 2002).

Young, M.J. (2001), *Step by Step XML*, Microsoft Press, 2001.