Process Specification Language For Project SCHEDULING informatio exchange

Jinxing Cheng[1], Michael Gruninger[2], Ram D. Sriram[3] and Kincho H. Law[4]

Abstract

Many project scheduling and management software systems are being employed in the construction industry. Standards-based translation is one way to achieve interoperability. This study discusses the applicability of the Process Specification Language (PSL) for exchanging project scheduling information among different applications. PSL was initiated by the National Institute of Standards and Technology (NIST) and is emerging as a standard exchange language for process information in the manufacturing industry. This paper explores how PSL can be used for exchanging project scheduling information among software programs in project management. Furthermore, we investigate how PSL could be utilized to reason about potential conflicts and to perform consistency checking on project scheduling information.

Keywords

Process Specification Language (PSL), information exchange, consistency checking, project management

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

[2] Project Leader, Ontology Development, ISO Standardization, National Institute of Standards and Technology, Gaithersburg, MD 20899-0001, USA, email: michael.gruninger@nist.gov

[3] Group Leader, Design Process Group, Manufacturing Systems Integration Division, National Institute of Standards and Technology, Gaithersburg, MD 20899-0001, USA, email: sriram@cme.nist.gov

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

1 Introduction

As the use of information technology increases in the construction industry, the need for software applications to interoperate has become increasingly important. With the variety of construction applications that can be employed in a construction project, large volumes of project information are created from different sources. Different members of a project team may use different application software for disparate purposes; examples may include Primavera Project Planner™ (P3) or Microsoft Project™ for scheduling, Vite SimVision™ for project organization, Timberline's Precision Estimating™ for estimating cost, and 4D Viewer (McKinney and Fischer 1998) for the view of construction progress. It is not unusual that project data are re-entered from one application to another.

To achieve interoperability, computer applications need to agree on a standard ontology. An ontology is an explicit specification of a topic that includes a set of terms and the relationships among these terms (Guarino 1997). Ontologies can be represented in many ways. Examples include graphical notations (e.g. UML) and logic-based representations (e.g. KIF). Currently, many product models and data standards, such as STEP (ISO 1994), IFC (IAI 1997), and aecXML (IAI 2002), exist to provide interoperability among different applications in the AEC domain. Examples include: (1) a product model for roofing systems developed using STEP (Vanier 1998); (2) the CIMsteel project, in which product modeling and data exchange for the construction steelwork industry was accomplished with STEP (Garas and Hunter 1998); (3) the use of Unified Modeling Language (UML) to represent product and process information in steelwork construction projects (Anumba et al. 2000); and (4) an IFC-based model to exchange information about maintenance management (Hassanain et al. 2000). Most of these existing standards focus, however, more on product data than on process information.

Standards for business process and workflow information have also been proposed to achieve interoperability in workflow management. For example, the Business Process Modeling Language (BPML) was proposed as a meta-language to model business processes (Arkin 2002). The Workflow Management Coalition (WFMC) has developed standards to enable the interoperability among multiple workflow software products (Fischer 2002). However, these languages cannot be directly applied for the exchange of process information in manufacturing or construction applications. The Process Specification Language (PSL) was designed specifically for exchanging process information among manufacturing applications (ISO 2003) and is currently undergoing a standardization process at the international level. In this paper, we evaluate the effectiveness of using PSL for the exchange of process information in construction management applications.

In addition to interoperability, maintaining the consistency of project information also poses a challenge since project information can come from various sources. In a complex project, a tremendous amount of information is being exchanged among the different project participants and software applications. Controlling the information flow and ensuring the validity of information exchanged between computer applications are among the challenges in project management. For example, in a distributed engineering environment, one project team may choose to use Primavera Project Planner™, while Microsoft Project™ is preferred by another for project scheduling. With multiple project participants utilizing different software applications, conflicts may arise due to partial changes, miscommunications, etc. Presently, there is no systematic approach to check the consistency of the scheduling information from different applications.

Few solutions have been considered for solving the data inconsistency problem. For example, a central database can be used as the common repository for different applications to maintain data persistency and consistency, such as the approach adopted for the Collaborative Dynamic Project Management (CDPM) system (Penã-Mora and Dwivedi 2002). However, this centralized database approach only partially solves the consistency problem in that while it eliminates version conflicts, it does not address any logic conflicts. Heuristic approaches have also been proposed, such as the 4D WorkPlanner Time-Space Conflict Analyzer (4D TSConAn), for categorizing and detecting spatial conflicts (Akinci et al. 2002). It is difficult, however, to generalize such heuristic approaches to handle the conflicts that are outside the defined domain problem.

The Process Specification Language (PSL) provides a logic-based representation, which is not only useful for the exchange of process information between application software, but also potentially useful for discovering and resolving conflicts. This study evaluates the applicability of PSL as an interchange language for construction project management applications, and explores the mechanism of using PSL to maintain the consistency of the project knowledge base.

This paper is organized as follows: Section 2 briefly introduces PSL and discusses the motivation and the major components of PSL. Mapping the concepts between PSL and project management applications is discussed in Section 3. Section 4 describes the parser and the wrappers developed for the exchange of project scheduling information using PSL. Section 5 discusses the potential use of PSL for consistency checking using a logic-based reasoning tool. Examples on information exchange and consistency checking are given in Section 6 to demonstrate the current prototype environment. Finally, Section 7 summarizes the results described in this paper.

2 Introduction to PSL

Representing activities and the constraints on their occurrences is an integral aspect of commonsense reasoning, particularly in project management, enterprise modeling, and manufacturing. In addition to the traditional concerns of knowledge representation and reasoning, the need to integrate software applications in these areas has become increasingly important. However, interoperability is hindered because the applications use different terminology and representations of the domain. These problems arise most acutely for systems that must manage the heterogeneity inherent in various domains and integrate models of different domains into coherent frameworks. For example, such integration occurs in business process reengineering, where enterprise models integrate processes, organizations, goals, and customers. Even when applications use the same terminology, they often associate different semantics with the terms. This clash over the meaning of the terms prevents the seamless exchange of information among the applications. Typically, point-to-point translation programs are written to enable communication from one specific application to another. However, as the number of applications has increased and the information has become more complex, it has been more difficult for software developers to provide translators between every pair of applications that must cooperate. What is needed is some way of explicitly specifying the terminology of the applications in an unambiguous fashion.

The Process Specification Language (PSL) has been designed to facilitate correct and complete exchange of process information among manufacturing systems (Schlenoff et al. 1999b, Menzel and Gruninger 2001)[5]. Included in these applications are scheduling, process modeling, process planning, production planning, simulation, project management, workflow, and business process reengineering. This section gives a brief overview of PSL; detailed description of PSL can be found in the PSL specification (ISO 2003).

The PSL Ontology is a set of first-order theories organized into PSL-Core and a partially ordered set of extensions. All extensions within PSL are consistent extensions of PSL-Core, although not all extensions within PSL need be mutually consistent. Also, the core theories need not be conservative extensions of other core theories. A particular set of theories is grouped together to form the Outer Core; this is only a pragmatic distinction, since in practice, they are needed for axiomatizing all other concepts in the PSL ontology. The relationships among the core theories are depicted in Figure 1.

The purpose of PSL-Core is to axiomatize a set of intuitive semantic primitives that is adequate for describing the fundamental concepts of manufacturing processes. Consequently, this characterization of basic processes makes few assumptions about their nature beyond what is needed for describing those processes, and the Core is therefore rather weak in terms of logical expressiveness. Specifically, the Core ontology consists of four disjoint classes: activities, activity occurrences, timepoints, and objects. Activities may have zero or more occurrences, activity occurrences begin and end at timepoints, and timepoints constitute a linearly ordered set with endpoints at infinity. Objects are simply those elements that are not activities, occurrences, or timepoints.

[5] PSL has been accepted as project ISO 18629 within the International Organisation of Standardisation, and as of October 2002, part of the work is under review as a Draft International Standard. The complete set of axioms for the PSL Ontology can be found at {http://www.mel.nist.gov/psl/psl-ontology/}.

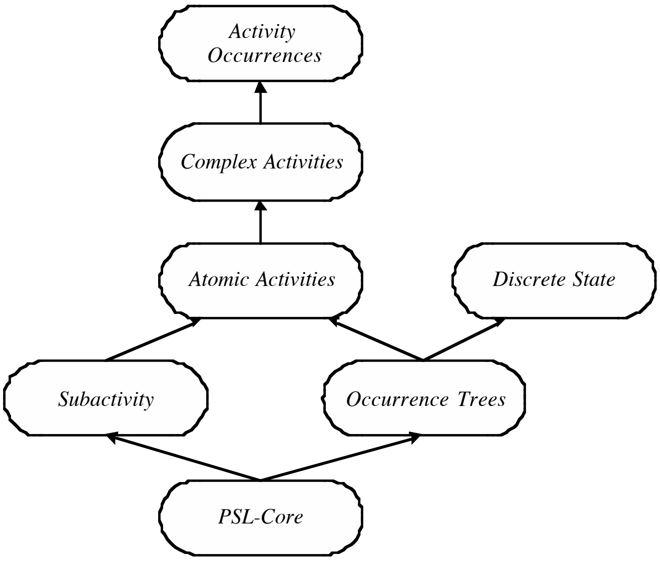

Figure 1. Core Theories of the PSL Ontology

PSL-Core is not strong enough to provide definitions of the many auxiliary notions that become necessary to describe all intuitions about manufacturing processes. To supplement the concepts of PSL-Core, the ontology includes a set of extended theories that introduce new terminology. These Outer Core theories provide the logical expressiveness to axiomatize intuitions involving concepts that are not explicitly specified in PSL-Core. The basic Outer Core theories include Occurrence Trees, Discrete States, Subactivities, Atomic Activities, Complex Activities, and Activity Occurrences. An Occurrence Tree is the set of all discrete sequence of activity occurrences. Discrete States denote states and their relationships to activities. Subactivities are defined to represent an ordering for aggregations of activities. Atomic Activities are defined to capture concurrent aggregation of primitive activities. Complex Activities characterize complex activities and the relationship between occurrences of an activity and occurrences of its subactivities. Activity Occurrences ensure that complex activity occurrences correspond to branches of activity trees. The remaining core theories in the PSL Ontology include: Subactivity Occurrence Ordering (axiomatizing different partial orderings over subactivity occurrence), Iterated Occurrence Ordering (axioms necessary for defining iterated activities), Duration (augmenting PSL-Core with a metric over the timeline), and Resource Requirements (which specifies the conditions that must be satisfied by any object that is a resource for an activity).

Table 1. Definitional extensions of PSL

Definitional Extensions	**Core Theories**	**Example Concepts**
• Activity Extensions	• Complex Activities	• Deterministic/nondeterministic activities • Concurrent activities • Partially ordered activities
• Temporal and State Extensions	• Complex Activities • Discrete States	• Preconditions • Effects • Conditional activities • Triggered activities
• Activity Ordering and Duration Extensions	• Subactivity Occurrence Ordering • Iterated Occurrence Ordering • Duration	• Complex sequences and branching • Iterated activities • Duration-based constraints
• Resource Role Extensions	• Resource Requirements	• Reusable, consumable, renewable, and deteriorating resources

There is a further distinction between core theories and definitional extensions. Core theories introduce primitive concepts, while all terminology introduced in a definitional extension have conservative definitions using the terminology of the core theories. The definitional extensions are grouped into parts according to the core theories that are required for their definitions. Table 1 gives an overview of these groups together with example concepts that are defined in the extensions. The definitional extensions in a group contain definitions that are conservative with respect to the specified core theories; for example, all concepts in the Temporal and State Extensions have conservative definitions with respect to both the Complex Activities and Discrete States theories.

3 PSL for Project Management Applications

PSL was designed to exchange process information among manufacturing applications. In a pilot implementation at NIST, PSL was successfully used to exchange manufacturing process information between the IDEF3-based ProCAP and the C++ based ILOG Scheduler (Schlenoff et al. 1999a). Although PSL was initially created mainly for the manufacturing industry, the core theories can be extended to construction project management and scheduling applications.

In our research, we first selected a typical project management tool, Primavera Project Planner™ (P3), as the benchmarking application to help define the core concepts for construction project management. Primavera P3 is a software tool for organizing, planning, and managing activities, projects, and resources. The following discussion focuses on the semantic mapping between Primavera P3 and PSL.

To achieve interoperability using PSL, semantic mapping is needed for various reasons. The same term may have different meanings in different applications and universes of discourse. For example, the term *successor* in PSL means that there are no other activities occurring between the two activities; however, in P3 the term does not have such an implication and only indicates that one activity cannot start before the other. On the other hand, the same concept in different applications may be represented differently using different terms. For instance, the terms *Successor* and *Predecessor* in P3 are used to describe the dependency relationships; in PSL, however, other terms, such as *after-start* and *after-start-delay*, are used to describe the same concepts. To exchange project scheduling information, we first need to map the concepts in different applications onto formal PSL ontology.

A typical construction project 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, procurement, and administrative activities. Each activity has attributes associated with it, such as start date, duration, etc. 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 relationships: Finish to Start, Finish to Finish, Start to Start, Start to Finish. Figure 2 depicts the dependency relationships and their respective definitions. For example, the "Finish to Start" relationship between activity *A* and activity *B* means that *B* starts only after *A* completes, and the "Finish to Finish" relationship indicates that *A* needs to complete before *B* does.

Each activity in a project schedule can be mapped onto an activity occurrence in PSL, while the timepoint is used to specify the beginning and the end points of an activity occurrence. PSL extensions provide terms to describe the dependency relationships among activities. For example, the term *before-start* in PSL corresponds to the "Start to Start" relationship, while the lag in the "Start to Start" relationship corresponds to the PSL term *before-start-delay*. The PSL expression *(before-start occ1 occ2 a3)* specifies that both *occ1* and *occ2* are subactivity occurrences of the activity *a3*, while the beginning timepoint of *occ1* is earlier than the beginning timepoint of *occ2*. In addition, the expression *(before-start-delay occ1 occ2 a3 d)* implies that *occ2* begins at least *d* timepoints after *occ1* begins. Table 2 lists the terms that are used in Primavera P3 and PSL to describe activities and dependency relationships.

In addition to activity and relationship information, resource allocation also plays an important role in project scheduling. A project schedule is not completely specified unless the necessary resources are allocated. Resources include people, material, and equipment required to finish the work. Resources can be mapped onto the lexicon *resource* in PSL, which identifies the object required by an activity.

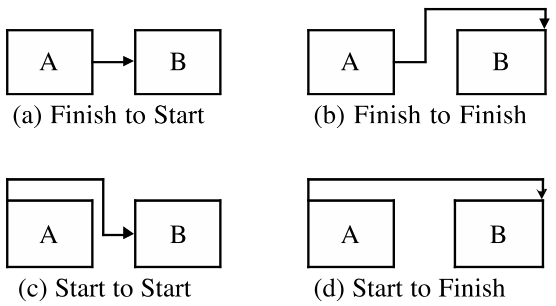

Figure 2. Dependency Relationships Among Activities

Table 2. Mapping of Activities and Dependency Relationships

Concepts in Primavera P3	PSL terms
Activity	Activity occurrence
Predecessor, Successor	Activity occurrence, before-start, before-finish, after-start, after-finish
Start to Start	before-start
Start to Finish	before-finish
Finish to Start	after-start
Finish to Finish	after-finish
Dependency Lag	before-start-delay, before-finish-delay, after-start-delay, after-finish-delay

Semantic mapping between PSL and project management applications is not always straightforward. For example, the *total float* concept in Primavera P3 cannot be directly mapped to a corresponding PSL term. In Primavera P3, *total float* indicates the maximum amount of time a task can be delayed without postponing the whole project. To express the *total float* concept, we need a set of PSL expressions. For example, assuming that in Primavera P3 there is a project (*proj1*) with the scheduled completion date on March 10, 2003, the activity *A* is scheduled to finish on October 7, 2002 with a total float of 3 days. To express the *total float* concept in the above example, we need to use the following PSL expressions.

(=> (beforeEQ (endof A) 10/10/2002) (beforeEQ (endof proj1) 03/10/2003))
(=> (before 10/10/2002 (endof A)) (before 03/10/2003 (endof proj1)))

Here October 10, 2002 is the completion date of the activity *A* if it is delayed by exactly 3 days. The first PSL expression implies that if *A* is delayed by no more than 3 days, the project will be completed on time with the end date of the project remains to be March 10, 2003. The second PSL expression indicates that if the end date of activity *A* is beyond October 10, 2002, the project completion date will then be postponed beyond March 10, 2003.

Generally speaking, PSL has more expressive power than many project management tools, including uncertainty, conditioning, probability, universal and existential relations, etc. As an example, the following PSL expressions can be used to indicate that a construction activity may require different resources depending on the result of other activities.

(activity-occurrence pourConcrete)
(doc pourConcrete "Pouring Concrete")
(=> (beforeEQ (endof formColumns) 11/20/2002) (demand constructionWorker pourConcrete 3))
(=> (before 11/20/2002 (endof formColumns)) (demand constructionWorker pourConcrete 6))
(after-start pourConcrete formColumns proj1)

Here, the activity *pourConcrete* requires different resources depending on its predecessor *formColumns*. If the activity *formColumns* is not completed before November 20, 2002, then the activity *pourConcrete* would require more construction workers. This conditioning expression, however, cannot be represented or encoded using project management tools that primarily handle deterministic scheduling.

Let's look at a mapping example between Primavera P3 and PSL. Figure 3 shows the major activities involved in the schedule of a typical residential building project. The project schedule is shown as a PERT (Primavera's Easy Relationship Tracing) chart from Primavera Project Planner™. In the project, the activity "Frame House" needs to finish before either the activity "Frame Roof" or "Install HVAC" can start. After the completion of these two activities, the activity "Install Drywall" can proceed. Figure 4 shows the ASCII outputs of the scheduling and resource information of the project plan from Primavera P3. For example, as shown in Figure 4, the activity "Frame House" starts on August 5, 2002 and lasts 15 days, while the activity "Install Drywall" needs the resource "drywall" to proceed.

The scheduling information in Primavera P3 can be described precisely using PSL. Figure 5 shows portion of the PSL expressions for the example project. Here, *ResProject* is the project identifier of the example residential building project. The PSL expressions *(after-start ID100 ID110 ResProject)* and *(after-start-delay ID100 ID110 ResProject 0)* specify that the activity *ID110* ("Frame Roof") needs to start after the completion of the activity *ID100* ("Frame House") with no lag between the two activities. The PSL expression *(available drywall ID130)* indicates that the resource *drywall* is available for the activity *ID130* ("Install DryWall"), while the PSL expression *(demand drywall ID130 2220)* specifies that the activity *ID130* requires 2200 sqft of *drywall*.

4 Information Exchange using PSL

To exchange project scheduling information among different project management applications, we need to develop wrappers for each application. Figure 6 shows the wrappers currently prototyped for information exchange between a variety of application software, including Primavera Project Planner™ (P3), Vite SimVision™, Microsoft Project™, and 4D Viewer, using PSL. The PSL wrappers are used to retrieve and transfer information between the applications.

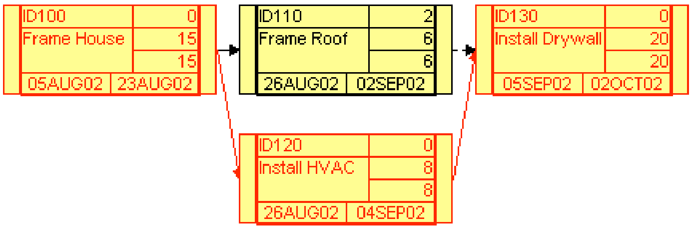

Figure 3. Example Dependency of a Scheduling Chart in Primavera P3

```
   ACT             TITLE            ES        EF        TF     RD
---------- ---------------- -------- --------  ----- ----   ID100
Frame House       5AUG02   23AUG02      0    15

    ID130       Install Drywall  5SEP02    2OCT02       0    20        ......   ACT
RES        RUT        QTC            QAC

       ----------- ---------    ----     ------------- ------------     ID130
DRYWALL    sqft       2200.00        2200.00 ......
```

Figure 4. Schedule

and Resource Information from Primavera P3

```
 (and      (activity-occurrence ID100)     (doc
ID100 "Frame House")     (beginof ID100 08/05/
2002)     (duration-of ID100 15)     (after-start
ID100 ID110 ResProject)     (after-start-delay
ID100 ID110 ResProject 0)     ...... ) (and
(resource drywall)

   (available drywall ID130)   (demand drywall
ID130 2220) ) ......
```

Figure 5. PSL Expressions For the

Example Chart in Primavera P3

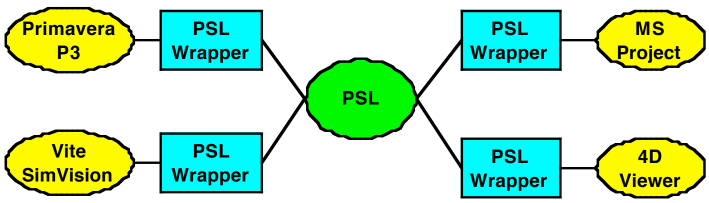

Figure 6. PSL in the Information Exchange

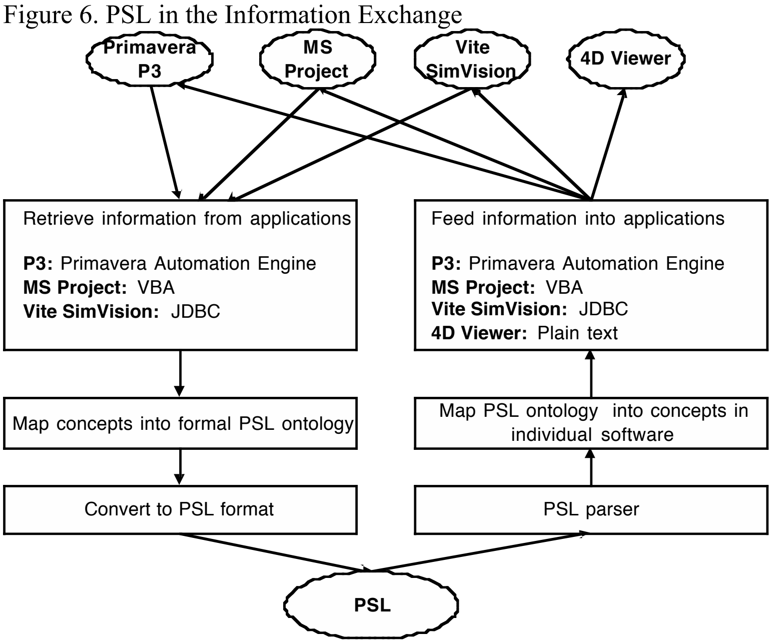

Figure 7. PSL Wrappers

There are three basic steps involved in exchanging project information using PSL. The first step is to retrieve the project information from an application and to update the project model. Semantic mapping is then performed to translate between the formal PSL ontology and the concepts in the project management tools. Finally, the project data are syntactically translated between PSL files and the applications.

The information exchange process is depicted in Figure 7. To map the information from applications to PSL, different wrappers have been implemented for various project management applications. In addition, these wrappers are also used to map the information from PSL to the applications.

After mapping the concepts in Vite SimVision™ onto PSL, we use Java Database Connectivity (JDBC) to parse the relevant information stored in the Access database created by Vite SimVision™, translate the information into PSL, and create a PSL file. For the PSL to Vite SimVision™ translation, the information in the PSL file is parsed and rewritten into VNB (Access database) file format.

For Primavera P3, the Primavera Automation Engine (RA) is employed. The RA is a set of object-oriented, OLE 2.0-based API, which allows object-oriented programming access to the P3 scheduling engine and other applications. We use RA to communicate with P3, such as retrieving project scheduling information from P3 and transferring project scheduling information to P3.

For Microsoft Project™, VBA (Visual Basic for Application) is employed. The process here is very similar to the communication protocols for Primavera P3.

For 4D Viewer (McKinney and Fischer 1998), the scheduling information from the PSL file is retrieved and converted into ASCII format required by the 4D Viewer.

A PSL parser has also been developed to read the project scheduling information from PSL files. One simplification we made in the PSL parser is that PSL sentences are expressed as relations rather than functions. In PSL, each function has a unique value; for example, in the PSL expression *(endof A),* the activity *A* can only have one unique completion date. In contrast, the value of a relation is either true or false; furthermore, relations can have disagreement on the last element. For example, the relations *(before t1 t2)* and *(before t1 t3)* differ. As a result, every function can be expressed as an equivalent relation with axioms that ensure the uniqueness of values, 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.

It should be noted that only the information that is common to the applications can be exchanged. As shown in Figure 8, Primavera Project Planner™ (P3) includes scheduling, resource, and cost information, while Vite SimVision™ provides scheduling, resource, communication, and organizational information. Scheduling and resource information, which is common to both applications, can be exchanged through PSL. However, not all scheduling and resource information is exchanged between these two applications, since the granularity of such information may be different. For example, Primavera P3 includes more detailed scheduling information than Vite SimVision™; in other words, not all scheduling information in Primavera P3 is needed by and transferred to Vite SimVision™.

The PSL parser developed so far can only deal with parsing predefined terms in PSL. We are currently investigating the possibility of building a generic PSL parser using JavaCC (Java Compiler Compiler (SUN 2002)). The generic PSL parser can read a grammar specification and convert it to a Java program that can recognize matches to the grammar.

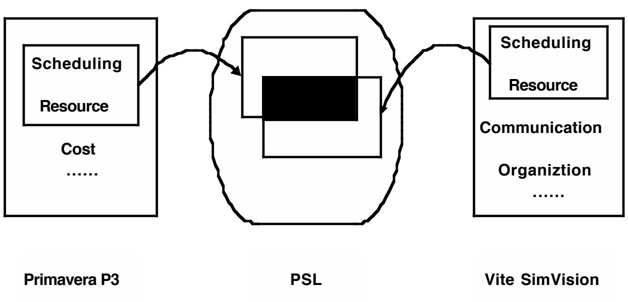

Figure 8. Exchange Information between Primavera P3 and Vite SimVision through PSL

5 Consistency Checking using PSL

Conflicts can occur from time to time during the course of a construction project. Design changes, unexpected weather conditions, labor actions, and procurement delays are all common bases for conflicts. In a distributed engineering environment, conflicts can occur more often due to partial changes and miscommunications. For example, a subcontractor may change its sub-schedule without realizing the potential impact on other project participants.

PSL can be used to check for consistency and to resolve some of the conflicts. We can use PSL to check the logic conflicts in the project base, where information comes from heterogeneous applications. For example, as illustrated later, our initial investigation shows that it is possible to detect version conflicts and cyclic dependency relationships between Primavera Project Planner™ and Microsoft Project™. With the conflicts found, it will be relatively easy to trace back to the sources of the conflicts. In addition, project personnel can check assumptions using PSL. For instance, suppose one would like to find out whether an activity can start on a specific date, say on November 15, 2001 without causing conflicts with other activities or prolonging the project. With PSL, we can add one piece of knowledge, which in PSL format would be *(beginof activity 2001-11-25)*, into the PSL knowledge base, and reason on the whole knowledge base. If no conflict is found during the reasoning, project personnel can infer that the assumption is reasonable; in other words, in this example, the activity can start on November 15, 2001.

Figure 9 depicts the basic process for detecting the conflicts or inconsistency of project information in the prototype implementation. PSL wrappers are employed to retrieve project information from different applications. In this work, we employ a theorem-prover---Otter (Organized Techniques for Theorem-proving and Effective Research)---as the logic reasoning tool (McCune 1994, Wos and Pieper 2000). Otter infers conclusions from given hypothesis and takes two types of input: logic clauses and first order logic sentences. Internally, Otter converts all inputs into logic clauses and applies inference rules to all possible logic clauses to infer new facts or conclusions. To utilize Otter, a translator has been built to convert PSL files and PSL axioms into first order logic sentences that Otter can understand.

The reasoning process using Otter can be summarized in Figure 10. Otter first infers new conclusions from the existing knowledge base. For the new knowledge, Otter rewrites it and checks whether it is subsumed by the existing knowledge. If not, the new knowledge will be added to the existing knowledge base; otherwise, it will be deleted. Usually, the reasoning process will stop either when Otter finds conflicts, or when no more conclusions can be inferred.

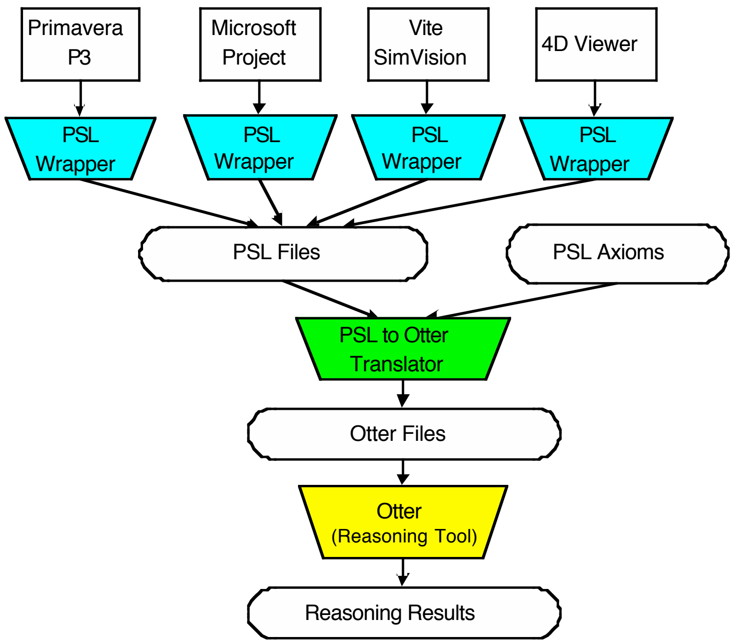

Figure 9. Consistency Checking Using PSL

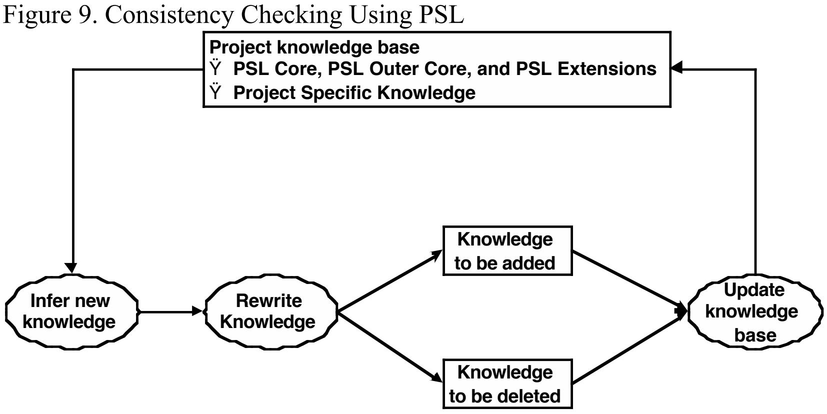

Figure 10. Simplified Reasoning Process in Otter

The knowledge base includes two main parts: (1) axioms and definitions from PSL Core, PSL outer core, and PSL Extensions; and (2) facts of individual project from heterogeneous sources. The reasoning among the axioms and definitions can significantly slow the reasoning process without producing essential results. We therefore partition the inputs into two lists: the axioms on the usable list and the project specific facts on the SOS (set of support) list. The performance of Otter can be significantly improved by separating the project specific knowledge and the PSL axioms/definitions. For example, in the chip design project to be presented in Section 6.2, Otter takes only 3 seconds to complete the reasoning, as compared to several hours without partitioning.

6 Demonstrations

This section presents several examples to demonstrate the concepts described in this paper. In Section 6.1 we show two examples which illustrate the use of PSL for information exchange. Section 6.2 shows an example that demonstrates how PSL can be used for consistency checking.

6.1 information exchange using PSL

6.1.1 Example 1: A Chip Design Scenario

We select a sample project from Vite SimVision^{TM} to test PSL for the exchange of project scheduling information. A Vite SimVision^{TM} project is composed of a traditional CPM diagram and additional links showing failure dependence, reciprocal information, and management structure. The example scenario, as shown in Figure 11, 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: (1) Start Project, (2) Ship Tapes to Foundry, and (3) Fab, Test and Deliver. The activity “Design_Coordination” maintains the overall control of the project.

Using PSL, we successfully exchange scheduling information among Vite SimVision^{TM}, Primavera Project Planner^{TM} (P3), and Microsoft Project^{TM}. Figure 12 shows some selected logic sentences from the PSL file particular to this project. These logic sentences specify the properties of the project and activities in the project. For example, the expression *(beginof TUTO 9/18/1998)* specifies that the TUTO project starts on 9/18/1998. The expression *(after-start ID190 ID200 TUTO)* specifies that the task ID190 should finish before the task ID200 starts.

Figures 13 to 15 illustrate the generated schedule in Vite SimVision^{TM}, P3, and Microsoft Project^{TM}. Figure 13 is the original Gantt chart of the sample project in Vite SimVision^{TM}. Figures 14 and 15 show the regenerated project schedule in P3 and Microsoft Project^{TM}, respectively. As shown in the figures, project scheduling information is successfully exchanged among these three applications. Activities have the same start date and duration in all three applications. The critical paths are also the same in all three applications.

In this example scenario, the scheduling information from Vite SimVison^{TM} is retrieved and converted into a PSL file. The information in the PSL file is then parsed and used to regenerate the project schedule in Primavera Project Planner^{TM} and Microsoft Project^{TM}. The successful information exchange among these applications shows the potential of PSL as an interchange standard in construction project management.

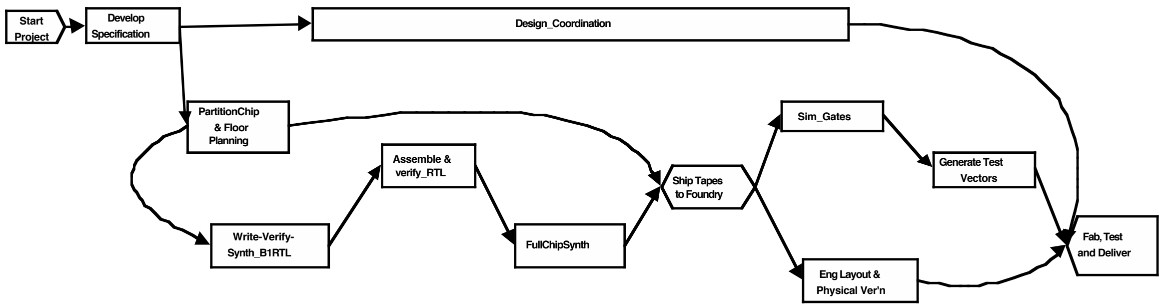

Figure 11. Original CPM Diagram in Vite SimVision

(and (project TUTO) (doc TUTO "TUTORIAL Project") (beginof TUTO 9/18/1998) (subactivity-occurrence ID100 TUTO) ……) (and (activity-occurrence ID190) (doc ID190 "PartitionChip & Floor Planning") (beginof ID190 10/19/1998)
(duration-of ID190 42) (after-start ID190 ID200 TUTO) (after-start-delay ID190 ID200 TUTO 0) ……)

Figure 12. Sample PSL File

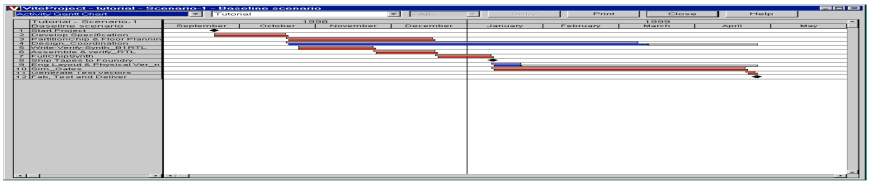

Figure 13. Original Gantt Chart in Vite SimVision

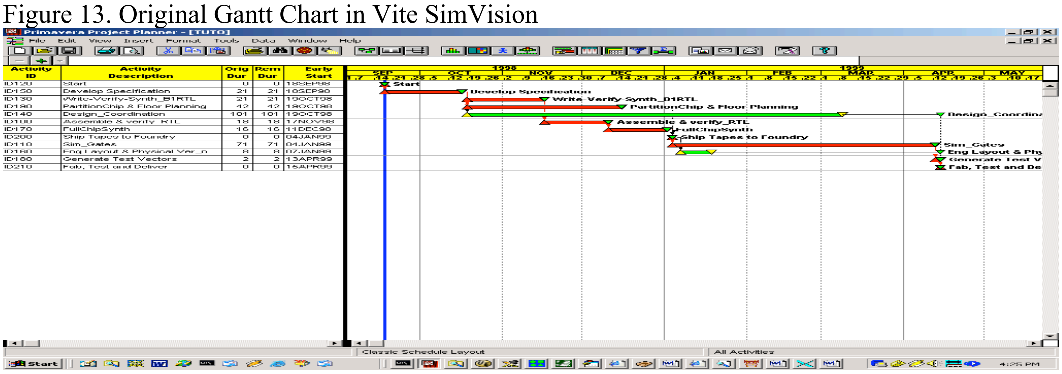

Figure 14. Regenerated Schedule in Primavera Project Planner using PSL

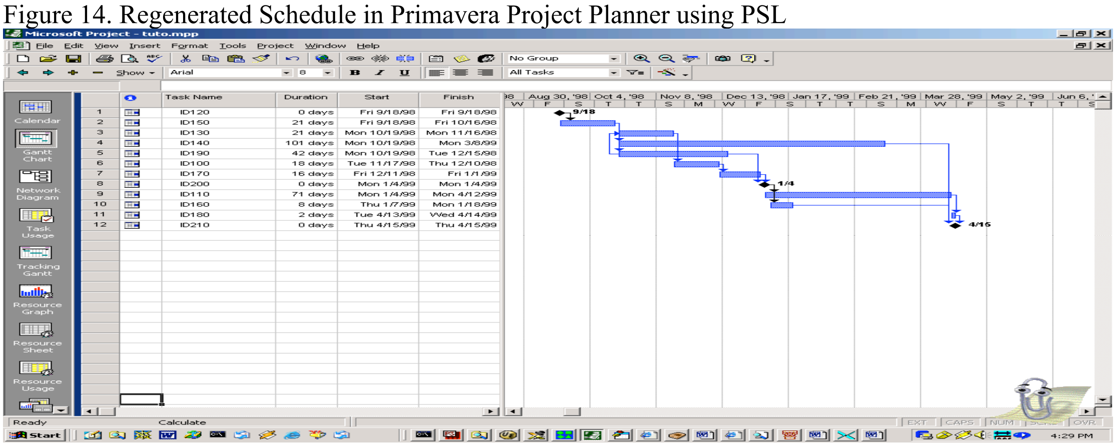

Figure 15. Regenerated Schedule in Microsoft Project using PSL

6.1.2 Example 2: Mortenson Ceiling Project

We demonstrate the scalability and applicability of PSL as an interchange standard through the Mortenson Ceiling Project, which is part of the Walt Disney Concert Hall, built by Mortenson Construction and designed by Frank O. Gehry & Associates. There are 191 activities and 459 dependency relationships in this example project. We use PSL as the data standard to exchange project scheduling information among Primavera P3, Microsoft Project™, and 4D Viewer. The PSL file of this project contains more than 2000 logic sentences. Figures 16 to 18 show selected results of this example demonstration. Figure 16 is the original Gantt chart of the ceiling project in P3. Figure 17 shows a snapshot of the construction progress in 4D Viewer on March 25, 2001. The scheduling information originally in Primavera Project Planner™ (P3) is successfully transferred to Microsoft Project™ using PSL, as shown in Figure 18.

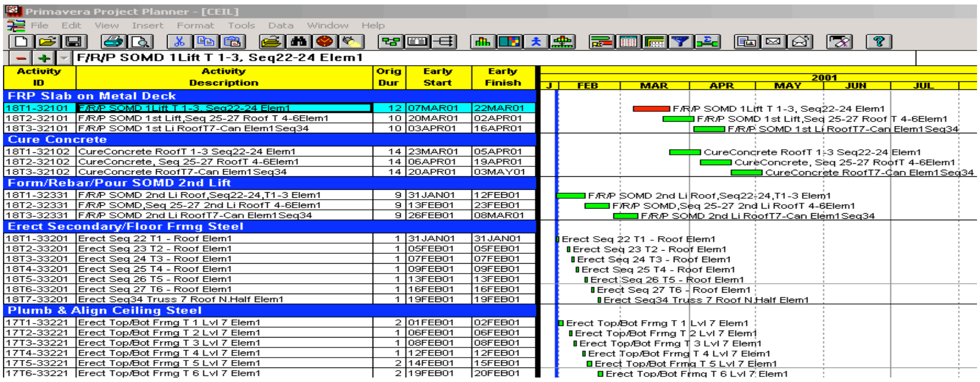

Figure 16. Original Schedule in Primavera P3

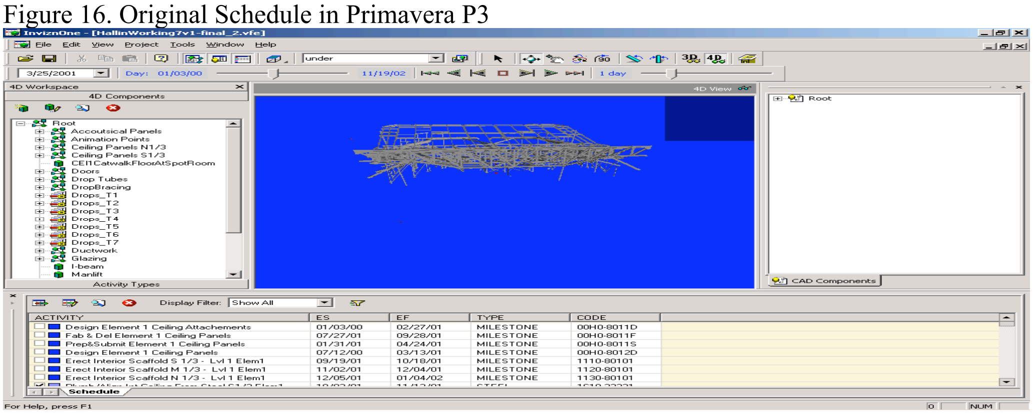
Figure 17. Model in 4D Viewer Taken on March 25, 2001

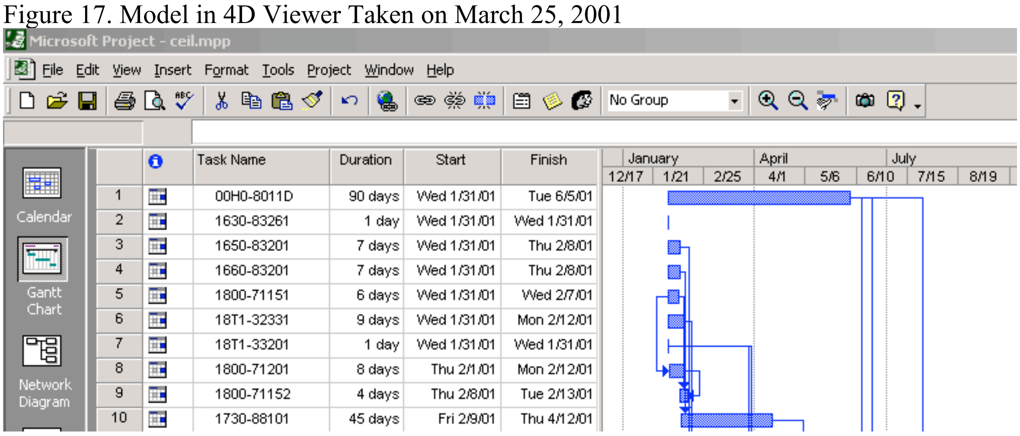

Figure 18. Regenerated Gantt Chart in Microsoft Project using PSL

To further illustrate the information exchange process, we altered the duration of activity 18T1-33201 from 1 day to 40 days in Microsoft Project^{TM}, as shown in Figure 19. The regenerated information is exchanged and displayed using Primavera Project Planner^{TM} in Figure 20 and 4D Viewer in Figure 21. The successful information exchange on this project demonstrates the scalability, applicability, and robustness of PSL as an interchange standard.

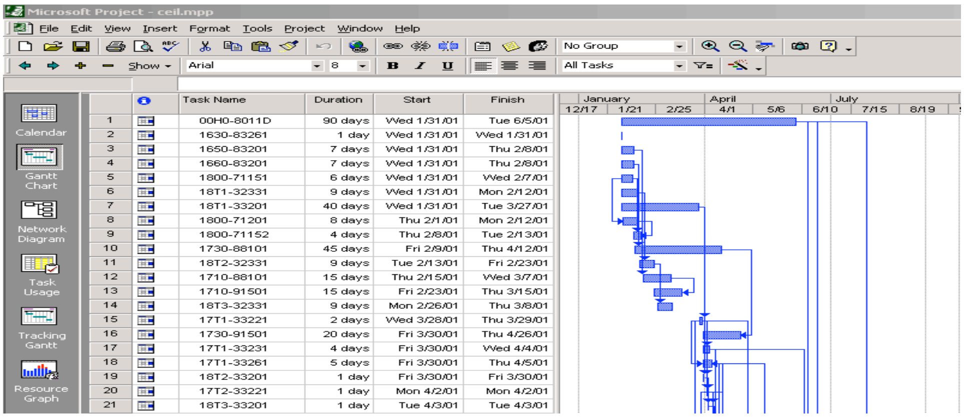

Figure 19. Updated Project Schedule in Microsoft Project

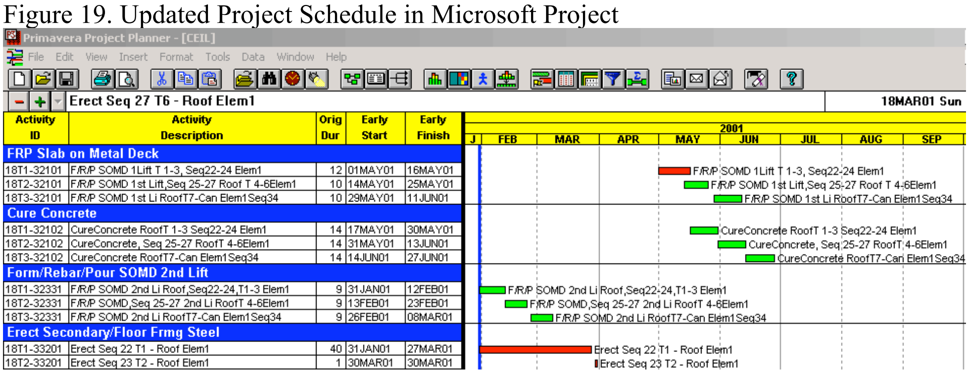

Figure 20. Updated Project Schedule in Primavera P3

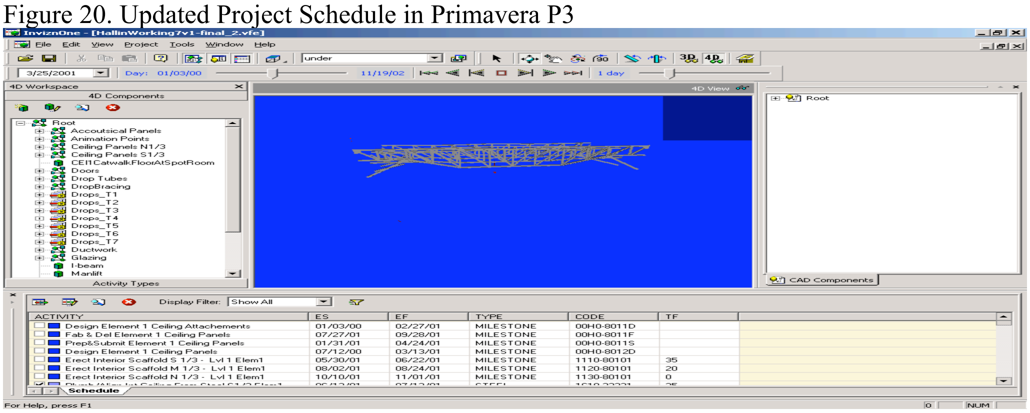

Figure 21. Updated Model in 4D Viewer Taken on March 25, 2001

6.2 Consistency Checking of project schedules

To test the use of PSL for consistency checking purpose, we use the same chip design scenario, as shown in Figure 11. For this example, which includes the design and fabrication of a chip set for a new personal digital assistant (PDA) product, the project involves managing design tasks as well as the foundry's layout, testing, and manufacturing tasks. Here we assume that there are two groups working on the project: one primarily responsible for the foundry's layout, and the other primarily responsible for testing and manufacturing tasks. Assuming that the two groups employ different application software, they work on the schedule independently but collaboratively. In addition, let's assume that group 1 uses Primavera P3 to create the detailed schedule. Moreover, in this group's schedule the "Eng Layout & Physical Ver'n" task is assumed to start after the "General Test Vector" task. Figure 22 shows the group 1's schedule in Primavera P3, and Figure 23 shows the CPM diagram.

For group 2, Microsoft Project™ is employed as the project management tool. Furthermore, the task "PartitionChip & Floor Planning" is split into two tasks: task "PartitionChip" and task "Floor Planning." In addition, in the schedule, group 2 assumes that the task "Sim_Gates" should follow the task "Eng Layout & Physical Ver'n." Figure 24 shows the group 2's schedule in Microsoft Project™, and Figure 25 shows the CPM diagram.

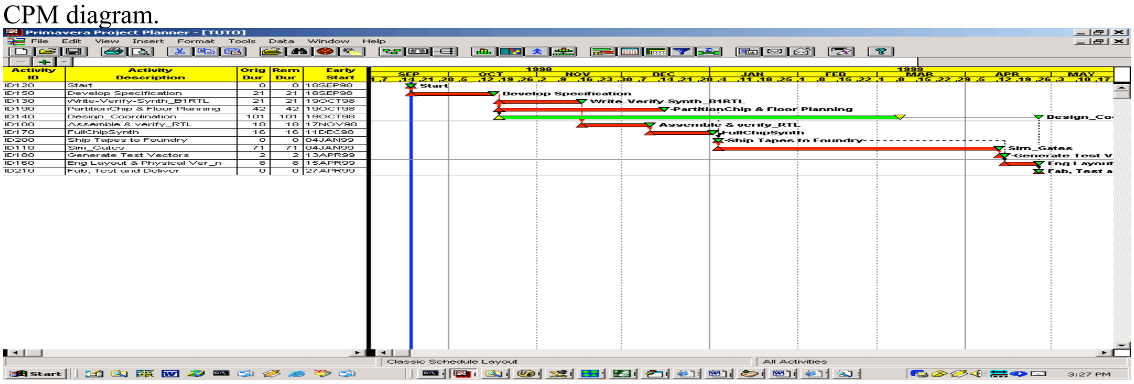

Figure 22. Group 1's Schedule in Primavera P3

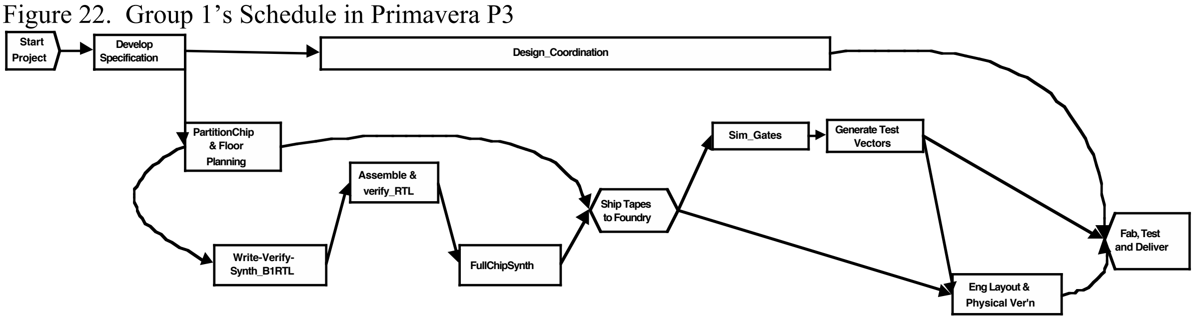

Figure 23. Group 1's CPM Diagram

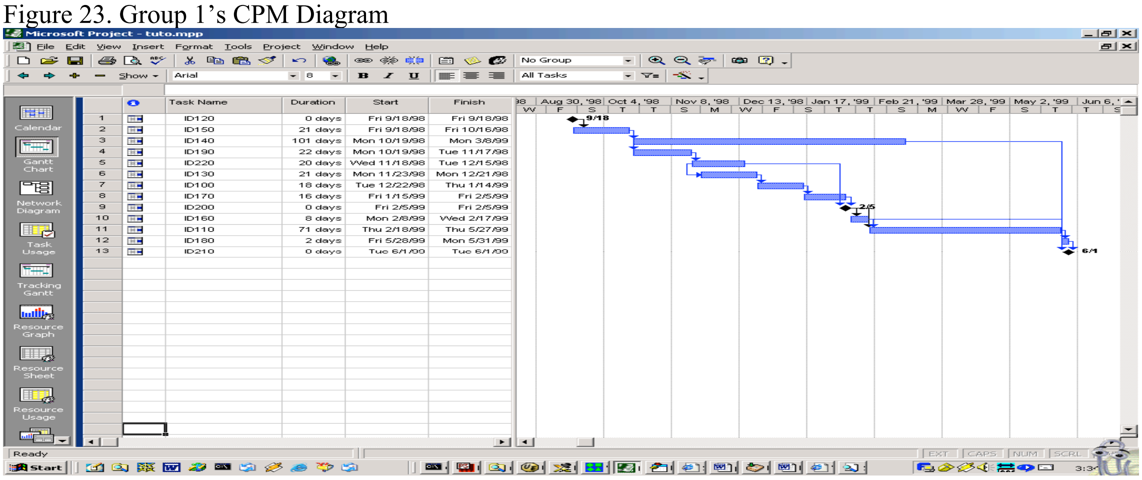

Figure 24. Group 2's Schedule in Microsoft Project

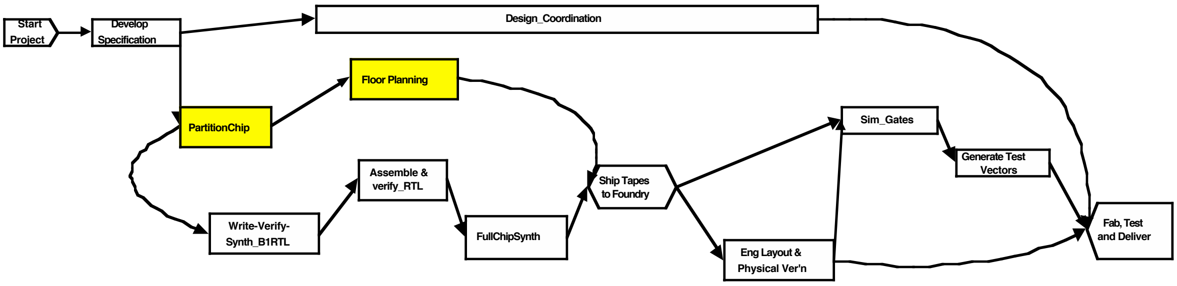

Figure 25. Group 2's CPM Diagram

To check for inconsistencies in the two schedules, we first use PSL wrappers to retrieve project information from Primavera P3 and Microsoft Project™. We then store the information in PSL files, convert the PSL files into Otter format, and link the project information with Otter. Finally, Otter is employed to reason about the project knowledge base and to detect conflicts. Figure 26 shows the results obtained from the reasoning. In the last sentence, the "$F" indicates a conflict has been found; the sentence numbers 333 and 47 can be used to traced the sources of conflicts. In particular, the sentence *after_start(ID110,ID180,TUTO)* specifies that ID110 ("Sim_Gates") should finish before ID180 ("Generate Test Vectors") starts. Similarly, *after_start(ID180,ID160,TUTO)* indicates that ID180 completes before ID160 ("Eng Layout & Physical Ver'n") starts, while *after_start(ID160,ID110,TUTO)* indicates that ID160 completes before ID110 starts. The conflict detected is graphically depicted in Figure 27. A cyclic dependency relationship in the project schedule is detected because the task "Sim_Gates" needs to start after the task "Eng Layout & Physical Ver'n" is completed, while at the same time the activity "Eng Layout & Physical Ver'n" needs to start after the activity "Sim_Gates" finishes.

```
44 [] -after_start(x100,x101,x102)| -
after_start(x101,x103,x102)|after_start(x100,x103,x102).
47 [] -after_start(x111,x112,x113)| -
after_start(x112,x111,x113). 85 []
after_start(ID110,ID180,TUTO). 136 []
after_start(ID180,ID160,TUTO). 252 [] after_start(I
D160,ID110,TUTO). 310 [hyper,136,44,85]
after_start(ID110,ID160,TUTO). 333 [hyper,310,44,252]
after_start(ID160,ID160,TUTO). 361 [hyper,333,47,333] $F.
```

Figure 26. Reasoning Results in Cyclic Dependency Relationships

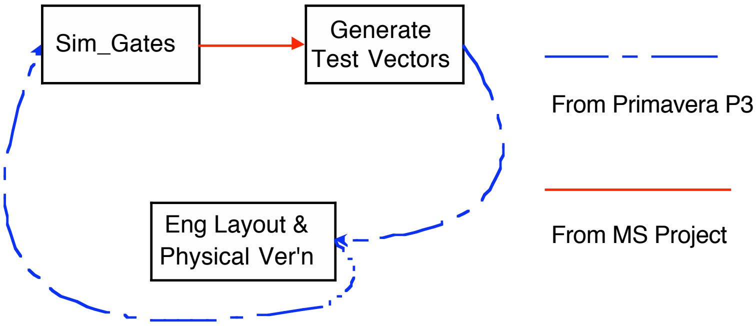

Figure 27. Cycle in Dependency Relationships

In addition to detecting logic conflicts in the activity relationships, we can also detect other conflicts that may arise due to versioning problems. For example, the same activity may have different start dates or durations in Primavera P3 and Microsoft Project™. To find these conflicts, we can simply add the following axioms into the knowledge base.

(forall ?a ?t1 ?t2 (=> (beginof ?a ?t1) (beginof ?a ?t2) (= ?t1 ?t2))

(forall ?a ?d1 ?d2 (=> (duration-of ?a ?d1) (duration-of ?a ?d2) (= ?d1 ?d2))

The first axiom specifies that the start date of an activity is unique. In other words, if an activity has two start dates, these two start dates must be equal. Similarly, the second axiom specifies that the duration of an activity is unique. These axioms will guarantee that an activity has a unique start date or duration. With these axioms added into the project knowledge base, Otter can detect the activities that have different start dates or durations in Primavera P3 and Microsoft Project™.

```
59 [] -beginof(x162,x163)| -beginof(x162,x164)|x163==x164.
161 [] beginof(ID210,10738). 273 [] beginof(ID210,10773).
323 [hyper,273,59,161,demod,propositional] $F.
```

Figure 28.

Reasoning Results in Version Conflicts

Figure 28 shows the sample conflict of the start dates of the activity ID210 ("Fab, Test and Deliver") detected by the reasoning tool. The first logic sentence in Figure 28 indicates that an activity must have a unique start date. Since Otter cannot directly operate on dates, we assume 01/01/1970 as the base date and use the Java class *Calendar* to convert the dates into numeric values. The second logic sentence *beginof(ID210,10738)* specifies that the activity ID210 starts at 10738 that is equivalent to 04/27/1999, as shown in Figure 22, which displays the project schedule using Primavera P3. Similarly, in the logic sentence *beginof(ID210,10773)*, the numeric value 10773 corresponds to the date 06/01/1999, which is the start date of the activity ID210 from the schedule shown in Figure 24 using Microsoft Project™. The last logic sentence in Figure 28 concludes that the activity ID210 has different start dates in the schedules from Primavera P3 and Microsoft Project, thus causing inconsistency.

The above examples show that PSL can be used to detect inconsistency in the project knowledge base. Following the proof process, we can trace for the root of the conflicts, identify the causes, and help resolve the inconsistency problems in the project.

7 Conclusions

In an engineering project, project team members may use many software applications. Exchanging project information among different software applications poses an impediment to collaboration. Maintaining the consistency of the project information from various sources presents an even bigger challenge. Although PSL was originally designed for manufacturing process information, our research shows that PSL can be used for construction project management applications. In this study, we have developed PSL wrappers and successfully exchanged project scheduling information among software applications, such as Primavera Project Planner™, Microsoft Project™, Vite SimVision™, and 4D Viewer. Moreover, we have explored the potential use of logic-based PSL for conflict resolution and consistency checking of a project schedule. Our research shows that PSL, an emerging interchange standard for manufacturing applications, not only shows promise in this role, but also has the potential to resolve conflicts and check consistency.

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 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 4D Viewer and the 4D model of the Mortenson Ceiling Project are provided by Professor Martin Fischer and his research group at Stanford University. The authors are grateful to Mr. Peter Denno of NIST for his valuable inputs on this paper. No approval or endorsement of any commercial product by the National Institute of Standards and Technology or by Stanford University is intended or implied.

References

Akinci, B., Fischer, M., Levitt, R., and Carlson, R. (2002). "Formalization and Automation of Time-Space Conflict Analysis." *Journal of Computing in Civil Engineering*, Vol. 16, No. 2, pp. 124-134.

Anumba, C.J., Baldwin, A.N., Bouchlaghem, N.M., Prasad, B., Cutting-Decelle, A.F., Dufau, J., and Mommessin, M. (2000). "Integrating Concurrent Engineering Concepts in a Steelwork Construction Project." *Concurrent Engineering: Research and Applications*, Vol. 8, No. 3, pp. 199-212.

Arkin, A. (2002), "Business Process Modeling Language." Draft of the BPML Specification, BMPL Working Group.

Fischer, L. (editor) (2002), *Workflow Handbook 2002*. Future Strategies.

Garas, F. K., and Hunter, I. (1998). "CIMSteel (Computer Integrated Manufacturing in Constructional Steelwork) - Delivering the Promise." *Structural Engineer*, Vol. 76, No. 3, pp. 43-45.

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

Guarino, N. (1997). "Understanding, Building and Using Ontologies." *Int. J. of Human-Computer Studies*, Vol. 46, No. 2-3, pp. 293-301.

Hassanian M, Froese T., and Vanier D. (2000). "IFC-based Data Model for Integrated Maintenance Management." *Proceedings of the Eighth International ASCE Conference on Computing and Building Engineering*, Vol. 1, pp. 796-803, Stanford, CA.

IAI (1997). "Industry Foundation Classes." Specification Volumes 1-4, International Alliance for Interoperability, Washington, DC.

IAI (2002). "AecXML." International Alliance for Interoperability, <http://www.aecxml.org>.

ISO (1994). "Product Data Representation and Exchange: Part 1: Overview and Fundamental Principles." No. 10303-1, International Organization for Standardization.

ISO (2003). "Industrial Automation System and Integration -- Process Specification Language." No. 18629-11, International Organization for Standardization.

McCune, W.W. (1994). "Otter 3.0 Reference Manual and Guide." Mathematics and Computer Science Division, Argonne National Laboratory, Report No. ANL-94/6.

McKinney, K., and Fischer, M. (1998). "Generating, Evaluating and Visualizing Construction Schedules with 4D-CAD Tools." *Automation in Construction*, Vol. 7, No. 6, pp. 433-447.

Menzel, C., and Gruninger, M. (2001), "A formal foundation for process modeling." *Proceedings of Formal Ontology in Information Systems*, Ogunquit, Maine, pp. 256-269.

Penã-Mora, F., and Dwivedi, G.H. (2002). "Multiple Device Collaborative and Real Time Analysis System for Project Management in Civil Engineering." *Journal of Computing in Civil Engineering*, Vol. 16, No. 1, pp. 23-38.

Schlenoff, C., Ciocoiu, M., Libes, D., and Gruninger, M. (1999a). "Process Specification Language: Results of the First Pilot Implementation." *Proceedings of the International Mechanical Engineering Congress and Exposition*, Vol. 10, pp. 529-539, Nashville, Tennessee.

Schlenoff, C., Gruninger, M., and Ciocoiu, M. (1999b). "The essence of the Process Specification Language." *Transactions of the Society for Computer Simulation*, Vol. 16, No. 4, pp. 204-216.

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

Vanier, D. (1998). "Product Modeling: Helping Life Cycle Analysis of Roofing Systems." *The Life Cycle of Construction IT Innovations*, Stockholm, Sweden, pp. 423-235.

Wos, L., and Pieper, G.W. (2000). *A Fascinating Country in the World of Computing: Your Guide to Automated Reasoning*. World Scientific Publishing Company, Singapore.