Project Management Scientific Journal (https://damaacademia.com/pmsj/)
Volume 1, Issue 5, pp.11-40, May 2019
Published by: Dama Academic Scholarly & Scientific Research Society (www.damaacademia.com)

The Process Specification Language (PSL) for Project Management Applications

Jinxing Cheng
Department of Civil and Environmental engineering
Stanford University

Abstract
PSL was proposed by the National Institute of Standards and Technology (NIST) to exchange manufacturing process information. While most data exchange standards, such as STEP [49] and IAI's IFC [46], deal primarily with product data, PSL is designed specifically for process information [67, 80]. In this chapter, we explore the applicability of PSL for the exchange of project management data [25]. Following the discussion of the language is an elaboration of how to exchange information among project management applications using PSL. A distributed data integration framework is proposed and prototyped. Two illustrative example projects are employed to demonstrate that information can be successfully exchanged through the prototype system. Conflicts appear in a variety of forms, arise due to different reasons, and occur frequently in many construction projects. It takes a great deal of time for project personnel to resolve various conflicts. This chapter proposes a formal mechanism to detect conflicts of project information arising from different sources. The implemented prototype has been successfully tested on a few example projects. In addition to consistency checking, the potential application of PSL in constraint scheduling is also explored. Large, complex projects often involve many constraints. It usually takes a significant amount of time for schedulers to ensure that a schedule meets all constraints. This chapter proposes a method to express constraints in PSL and to check whether a project schedule meets constraints. An example is provided to demonstrate that PSL has the potential to ensure conformity of project schedules to scheduling constraints.
Keywords: *Process Specification Language (PSL), Project Management Applications*

1.0 INTRODUCTION

The Process Specification Language (PSL) has been designed to facilitate correct and complete exchange of process information among manufacturing systems [67, 81][*]. Included in these applications are scheduling, process modeling, process and production planning, simulation, project management, workflow, and business process reengineering. This chapter discusses how to exchange information among distributed project management tools using PSL. As will be discussed in Chapter 3, PSL is also adopted as the basic data exchange language for project management applications in the simulation access framework. 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 2.1.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.

[*] 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/}.

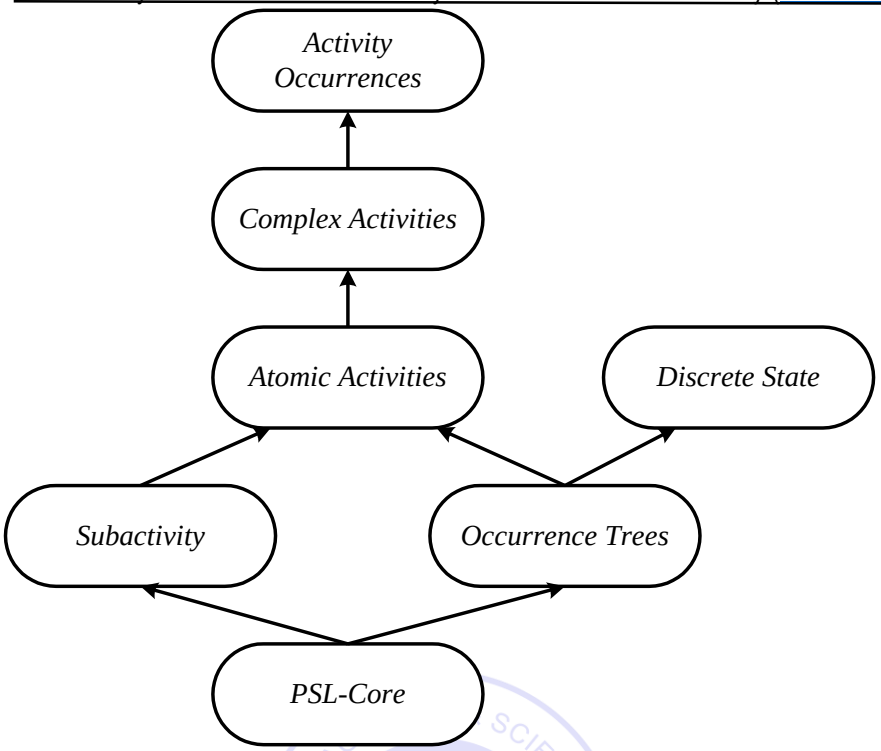

Figure **Error! No text of specified style in document.**.1: Core Theories of the PSL Ontology (from [67, 81])

PSL-Core is not strong enough to provide definitions of the many auxiliary notions that become necessary to describe all intuitions about manufacturing processes and project activities. 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 sequences 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 specify the conditions that must be satisfied by any object that is a resource for an activity).

Table **Error! No text of specified style in document.**.1: Definitional Extensions of PSL (from [67, 81])

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 has 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 2.1 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.

2.0 LITERATURE REVIEW

2.1 Using PSL to Exchange Information among Project Management Applications

To exchange information using PSL, wrappers for individual applications need to be implemented, so that they are PSL compliant. Section 2.2.1 first discusses semantic mapping between PSL and project management application concepts, an important step in wrapping applications. Section 2.2.2 then elaborates on how to develop wrappers for different project management tools.

2.2 Semantic Mapping between PSL and Project Management Application Concepts

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 [80]. 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, the Primavera Project Planner (P3), as the benchmark 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. First, 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. Second, 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; however, other terms, such as *after-start* and *after-start-delay*, are used in PSL to describe the same concepts. To exchange project scheduling information, we first need to map the concepts in different applications onto the 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 associated attributes, such as start date, duration, etc. Dependency relationships describe the constraints defining the order in which the activities must occur to complete the project [44]. There are four typical dependency relationships: Finish to Start, Finish to Finish, Start to Start, Start to Finish. Figure 2.2 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 2.2 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.

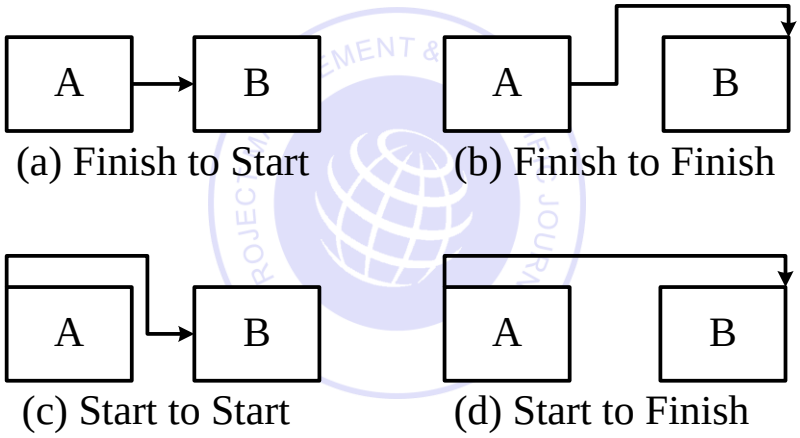

Figure **Error! No text of specified style in document.**.2: Dependency Relationships among Activities

Table **Error! No text of specified style in document.**.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. In particular, PSL has the capability to express uncertainty, conditioning, and universal and existential relations. 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 2.3 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 the 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 2.4 shows the ASCII outputs of the scheduling and resource information of the project plan from Primavera P3. For example, as shown in Figure 2.4, 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 2.5 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 square feet of *drywall*.

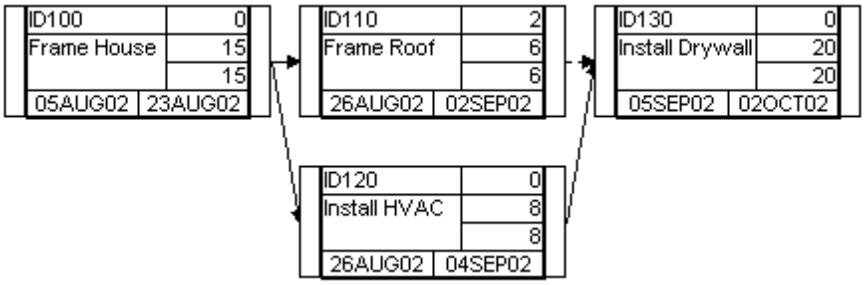

Figure **Error! No text of specified style in document.**.3: Example Dependency of a Scheduling Chart in Primavera P3

```
  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 **Error! No text of specified style in document.**.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 **Error! No text of specified style in document.**.5: PSL Expressions for the Example Chart in Primavera P3

2.3 Wrapping Project Management Applications

There are many commercial software tools as well as in-house computer programs that have been developed for project management. These tools use different internal representations and usually do not communicate to each other. To achieve interoperability, different approaches have been proposed over the past decades. Among them are

direct translation, neutral file-based integration, centralized database, and software interface. The pros and cons of these approaches are summarized in Table 2.3.

Table **Error! No text of specified style in document.**.3: Pros and Cons of Different Data Integration Approaches

Methods	Pros	Cons
Direct Translation	• Only one translator is needed to achieve integration if there are only two programs.	• It requires the cooperation of two software developers to achieve interoperability. • As the number of applications increase, the translators needed increase dramatically.
Neutral File Based Translation	• Only one translator is needed for each application. • Software developers focus on the translator of their own tools.	• It requires more work if there are only two tools involved.
Centralized Database	• This approach has the potential to ensure that all participants have the latest information.	• It could be difficult to define a common schema. • It requires extra programming effort to interact with database.
Software Interface	• This approach has more flexibility in automating the translating process.	• Programmers need to understand APIs of individual tools. • It may require significantly more programming effort. • It may not work when no API is provided.

To achieve interoperability among computer tools using PSL, it is necessary to develop wrappers for each individual tool. The specific implementation might be different depending on individual tools. There are two major considerations in selecting an appropriate approach to wrapping a legacy application:

The Input/Output methods the legacy application supports: Typical Input/Output (I/O) methods includes application programming interface (API), neutral files, proprietary files, databases, and interactive interfaces. Different legacy applications may support some or all of the methods. The wrapping approach is thus limited to the I/O methods that the legacy tool supports. For example, we cannot use the API approach to interact directly with a legacy tool if the tool does not provide programming interface.

Developer' requirement: When more than one Input/Output method is provided by a legacy tool, it is up to the software developers to choose a method based on their specific requirements. For example, file-based approaches require less programming effort than the API-based approach; however, they also provide less flexibility in automation. In this research, the goal is to automate the data integration process as much as possible; thus, the programming interface approach is employed whenever possible.

To exchange project scheduling information among different project management applications, we need to develop wrappers for each application. The PSL wrappers are used to retrieve and transfer information between the applications, to map between application concepts and PSL ontology, and to parse and generate PSL files. Figure 2.6 shows a variety of software applications that have been wrapped using PSL. Currently, these applications include the Primavera Project Planner (P3), Microsoft Project, Vite SimVision, 4D Viewer, AutoCAD Architectural Desktop, the GeneralCost Estimator, and Microsoft Excel. As depicted in Figure 2.6, wrappers for individual applications need different implementations. The implementations of a few wrappers are listed as follows:

For Vite SimVision, we use Java Database Connectivity (JDBC) to parse the relevant information stored in the Access database created by the tool, 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 this information to P3.

For Microsoft Project, VBA (Visual Basic for Application) as well as the Microsoft Project Object Model 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. For Microsoft Excel, VBA is employed as the programming language. In addition, the Microsoft Excel Object Model is utilized to retrieve and update project information. For AutoCAD Architectural Desktop (ADT), we use VBA as well as the AutoCAD ActiveX Object Model. The General Cost Estimator is a cost estimating tool running in Microsoft Excel. The wrapping process for the tool is similar to the process for Microsoft Excel, except that the data structures and functionalities provided by the estimator are also utilized.

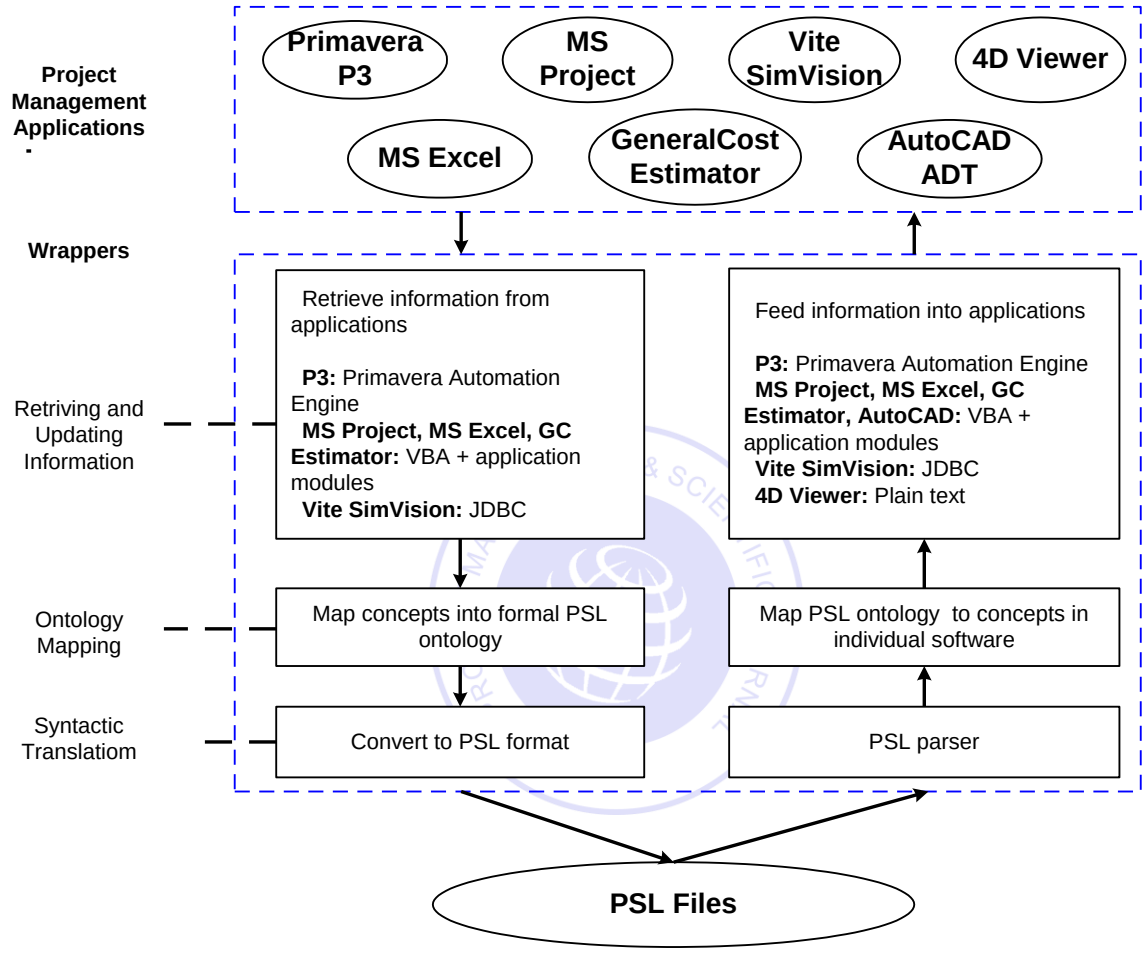

Figure **Error! No text of specified style in document.**.6: PSL Wrappers

As shown in Figure 2.7, the wrapping process is decomposed into three modules: I/O modules, mapping modules, and translation modules. By decomposing a wrapper into three modules, we standardize the wrapping process and improve the reusability of previous modules.

- A syntactic translation module include a PSL parser and generator, which are responsible for parsing and generating PSL files according to the PSL syntax, respectively. This module can be reused by wrappers for other project management tools with no or minimal changes.

- Semantic mapping depends not only on the PSL ontology but also on the concepts used in the individual applications. For many applications in the same domain, the concepts are similar. Thus, the semantic mapping module can also be reused by future PSL wrappers with appropriate changes in the terminology of application concepts.

- Input/Output (I/O) modules are responsible for retrieving and updating information in project management tools. The implementation of an I/O module depends on the I/O methods supported by the tool and its internal data representations. Thus, I/O modules cannot be reused if project management tools do not support the

same Input/Output methods. However, these modules can also be partly reused among a number of project management tools. For example, Microsoft Project, Microsoft Excel, AutoCAD ADT, and the GeneralCost Estimator all support VBA programming. The only difference is their individual Object Modules. Thus, a significant part of the I/O modules can also be reused among the wrappers for these applications.

As noted, modularizing the wrappers encourages shareability and reusability of the codes for different application tools. In summary, the syntactic translation modules have the greatest reusability, because they deal with PSL files in standard ontology. On the other hand, the I/O modules have to interact with different project management tools; thus, they are less likely to be reusable.

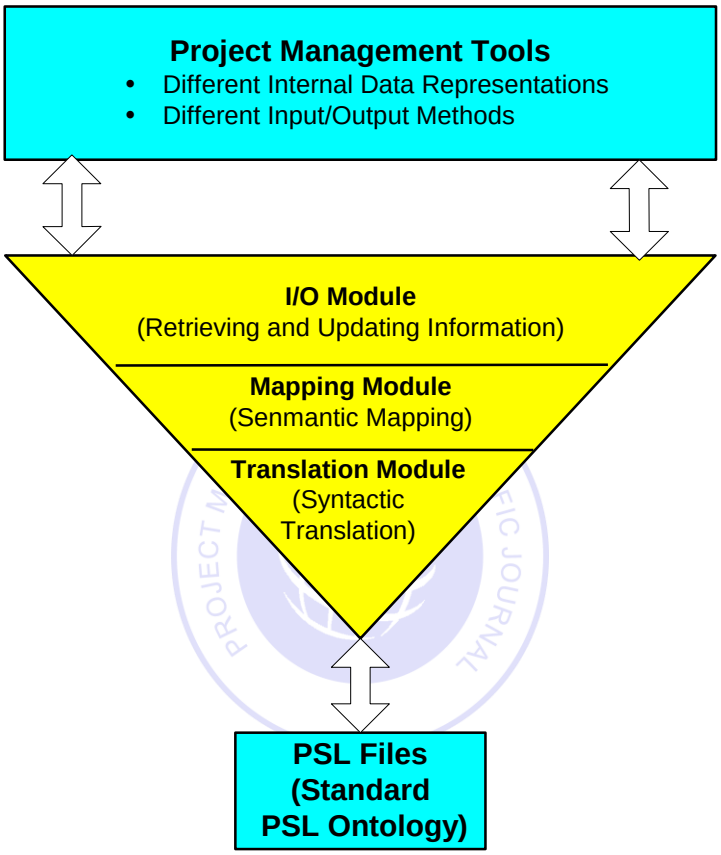

Figure **Error! No text of specified style in document.**.7: The Decomposition of PSL Wrappers

A PSL parser, as part of the wrappers, has 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 2.8, the 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.

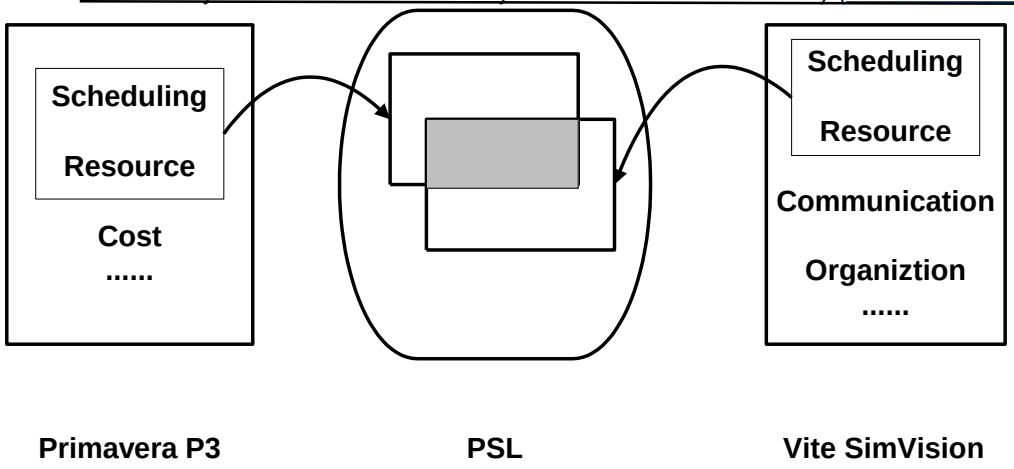

Figure **Error! No text of specified style in document.**.8: Exchange Information between Primavera P3 and Vite SimVision through PSL

2.4 Distributed Data Integration Infrastructure

Various architectures have been proposed to achieve software integration, such as localized integration, client-server integration and distributed integration. Localized integration involves integrating various software tools on one machine (e.g., a desktop PC). In a client-server environment, software integration is often accomplished using a project repository, which is either a neutral file or a database, residing on a central server, to which all applications communicate to exchange information. In a distributed environment, applications reside on different computers and are accessed over private or public, local or wide area network.

In a construction project, it is not unusual that project management tools are geographically distributed. The goal of a distributed integration infrastructure is to link application tools and to act collaboratively on a project. We have prototyped a distributed integration infrastructure using PSL as the information interchange standard among different project management tools [26]. As shown in Figure 2.9, a communication server is used to serve as the backbone of the system. The communication server is responsible for listening requests from the communication agents associated with different project management applications. When the server receives a request, it broadcasts the request to different communication agents. For example, the server may ask a communication agent to invoke a scheduling service or ask the agent to transfer the scheduling result to another simulation tool. The communication agents then pick up the request and process it. In addition, the Oracle database is used to store the project information, so that users can access and update project information through a Web browser.

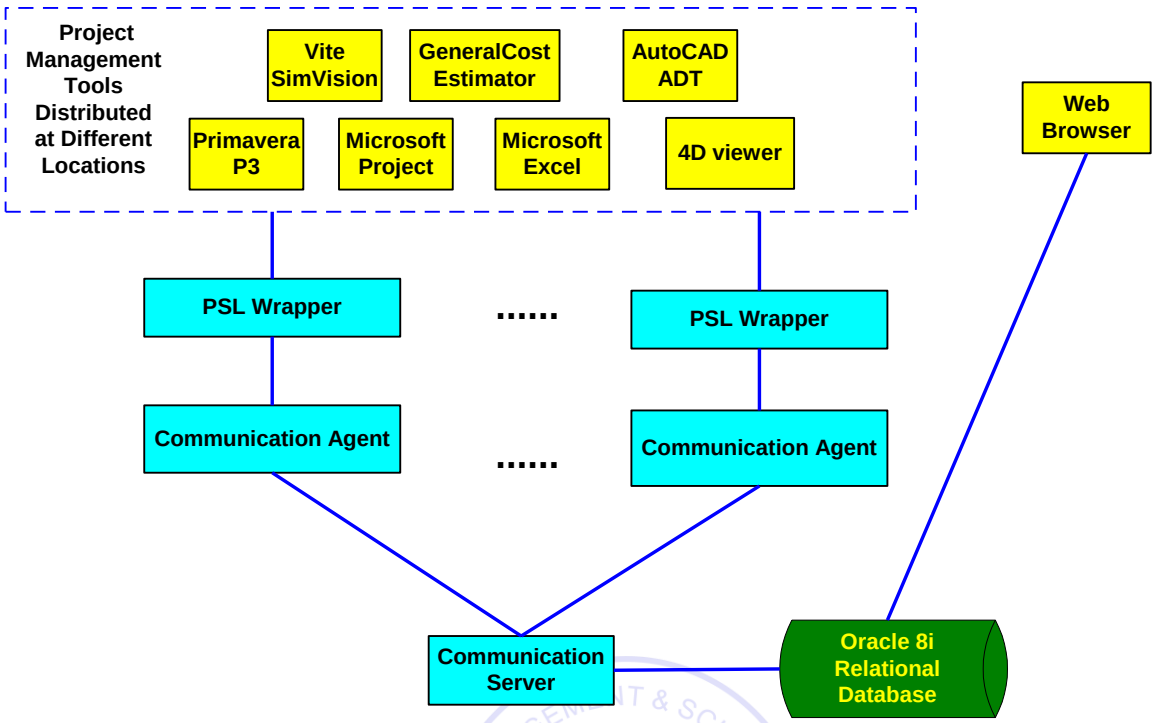

Figure **Error! No text of specified style in document.**.9: A Distributed Integration Infrastructure

2.5 Translation between PSL and Database

The Oracle database is used to store project information so that it is accessible by users through a Web browser. To use the database for data exchange, a database schema needs to be defined. Table 2.4 illustrates the database schema in the current implementation. The PROJECT table stores the overall information of projects. The ACTIVITY, SCHEDULE, and DEPENDENCY tables are used to describe activities and their relationships in the projects. The ACTOR table describes project participants, while the ASSIGNMENT table stores the assignment information of project activities. All cost information is stored in the COST table.

Table **Error! No text of specified style in document.**.4: A Database Schema for the Project Repository

Tables	**Attributes**
project	projectid, description, companyname, workday, workhour, startdate, finishdate, costid
activity	projectid, activityid, description, costid, scheduleid
actor	projectid, actorid, description, role, hourwage
cost	projectid, costid, actualcost, budgetedcost
schedule	projectid, scheduleid, startdate, finishdate, actualstart, actualfinish, earlystart, earlyfinish, latestart, latefinish, duration, actualduration, freefloat, totalfloat
dependency	projectid, dependencyid, activityid1, activityid2, relationship, driving, lag
assignment	projectid, assignmentid, activityid, actorid

Once the schema is defined, a Java program is developed and employed to translate the information between PSL and database. Figure 2.10 illustrates the translation process. JDBC is used to establish connection to the database,

and SQL statements are used to query project information in the database. The information is then written into PSL according to the syntax. In the reverse process, we can reuse the PSL parser that has already been developed to parse project information. SQL operations are constructed based on the information. The program finally connects to the database and writes the information into the database.

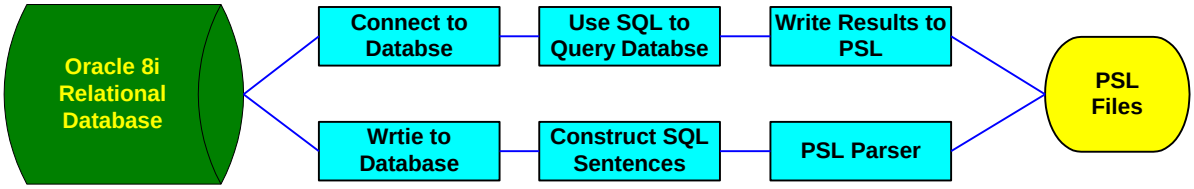

Figure **Error! No text of specified style in document.**.10: Translation between Database and PSL

2.6 Network Communication in the Distributed Integration Framework

The network communication mechanism in the distributed integration framework is illustrated in Figure 2.11. Java socket communication is used as the protocol between the communication server and agents. A communication agent consists of an event listener, an event dispatcher, and a data mapper. The messages in the system include control messages and data messages. Control messages, such as invocation and termination requests, are typically small in size. Data messages, such as the project scheduling information and organization information, however, are usually bigger in size. The event listener receives control messages, while the event dispatcher sends out control messages.

2.7 The data mapper is responsible for sending and receiving data messages.

Figure 2.12 shows a Java code segment of an event listener. The listener first defines two data streams: one input stream and one output stream. It then creates a Socket on a specific port. Finally, it keeps listening on the port to see if there are messages from the server.

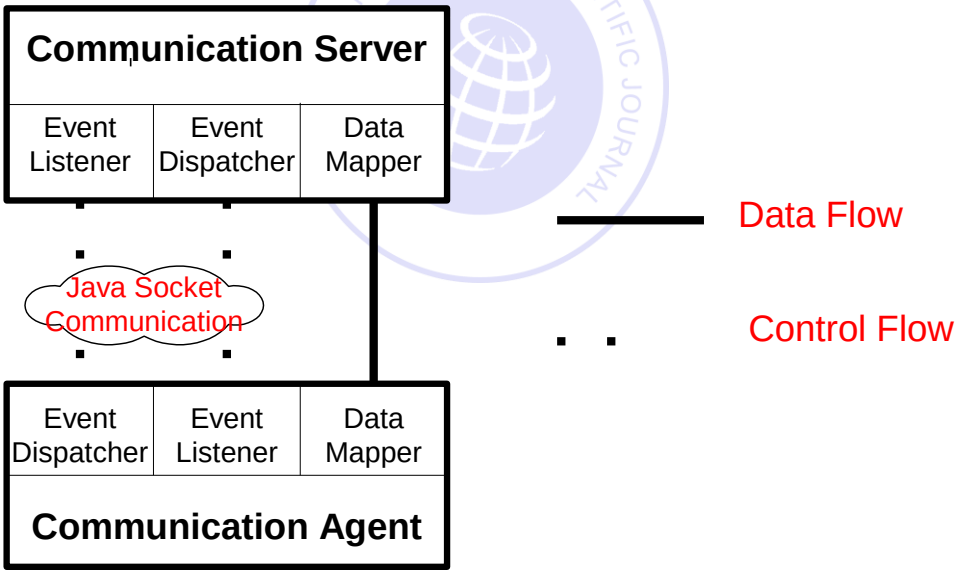

Figure **Error! No text of specified style in document.**.11: A Network Communication Framework

```
Public class ClientListener{
  protected DataInputStream i;  protected DataOutputStream o;
  public static void main (String args[]) throws IOException {
    Socket s = new Socket (args[0], Integer.parseInt (args[1]));
    ClientListener client = new ClientListener(" " + args[0] + ":"  + args[1], s.getInputStream (),
s.getOutputStream ());
    client.waitForEvent();
    s.close(); }
  public void waitForEvent () {
```

```
    try {   String line = i.readUTF ();}
    ……}
   ......
}
```

Figure **Error! No text of specified style in document.**.12: The Code Segment of an Event Listener

3.0 DEMONSTRATIONS OF DISTRIBUTED DATA INTEGRATION

In this section, two examples are used to demonstrate the integration of distributed project management tools. We have conducted a number of demonstrations between Glasgow Caledonian University in Scotland and Stanford University [27][*]. In the demonstrations, the Oracle database server, the 4D Viewer, and the Vite SimVison are located at Stanford University, while the scheduling services (e.g., Microsoft Project and the Primavera Project Planner) reside at Glasgow Caledonian University.

Example 1: A Chip Design Scenario

We select a sample project from the tutorial of Vite SimVision software to test PSL for the exchange of project scheduling information. A Vite SimVision 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 2.13, 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, the Primavera Project Planner (P3), and Microsoft Project. Figure 2.14 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.

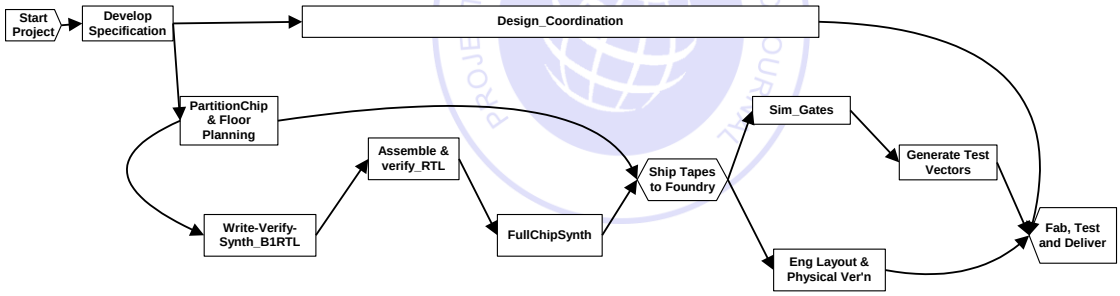

Figure **Error! No text of specified style in document.**.13: Original CPM Diagram in Vite SimVision

[*] The demonstrations were conducted in collaboration with Professor Bimal Kumar of Glasgow Caledonian University, UK.

```
(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 **Error! No text of specified style in document.**.14: Sample PSL File

Figures 2.15 to 2.17 illustrate the generated schedule in Vite SimVision, P3, and Microsoft Project. Figure 2.15 is the original Gantt chart of the sample project in Vite SimVision. Figures 2.16 and 2.17 show the regenerated project schedule in P3 and Microsoft Project, 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.

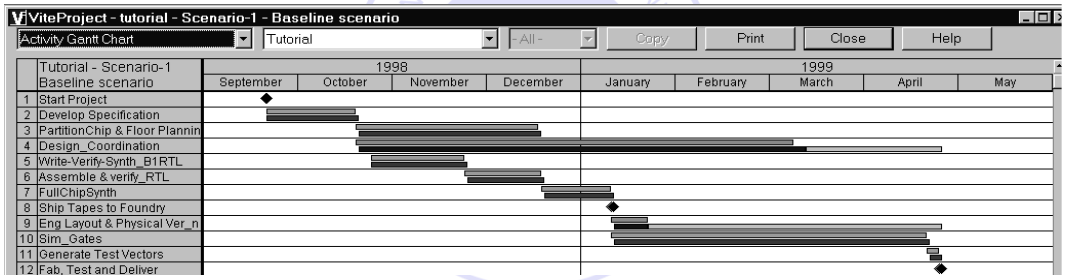

Figure **Error! No text of specified style in document.**.15: Original Gantt Chart in Vite SimVision

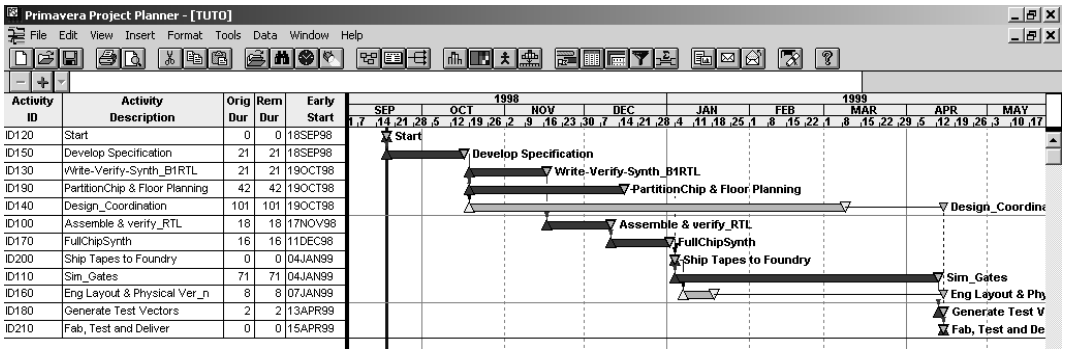

Figure **Error! No text of specified style in document.**.16: Regenerated Schedule in Primavera P3 using PSL

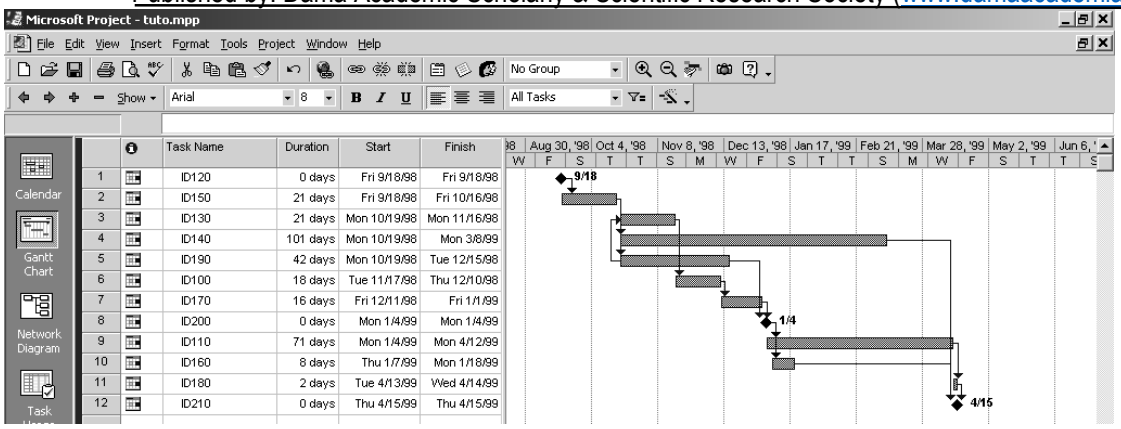

Figure **Error! No text of specified style in document.**.17: Regenerated Schedule in Microsoft Project using PSL

In this example scenario, the scheduling information from Vite SimVison 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 the Primavera Project Planner and Microsoft Project. The successful information exchange among these applications shows the potential of PSL as an interchange standard in construction project management.

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. PSL is employed 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 2.18 to 2.20 show selected results of this example demonstration. Figure 2.18 is the original Gantt chart of the ceiling project in the Primavera Project Planner (P3). Figure 2.19 shows a snapshot of the construction progress in 4D Viewer on March 25, 2001. The scheduling information originally in P3 is successfully transferred to Microsoft Project using PSL, as shown in Figure 2.20.

To further illustrate the information exchange process, we altered the duration of activity 18T1-33201 from 1 day to 40 days in Microsoft Project, as shown in Figure 2.21. The regenerated information is exchanged and displayed using P3 in Figure 2.22 and 4D Viewer in Figure 2.23. The successful information exchange on this project demonstrates the scalability, applicability, and robustness of PSL as an interchange standard.

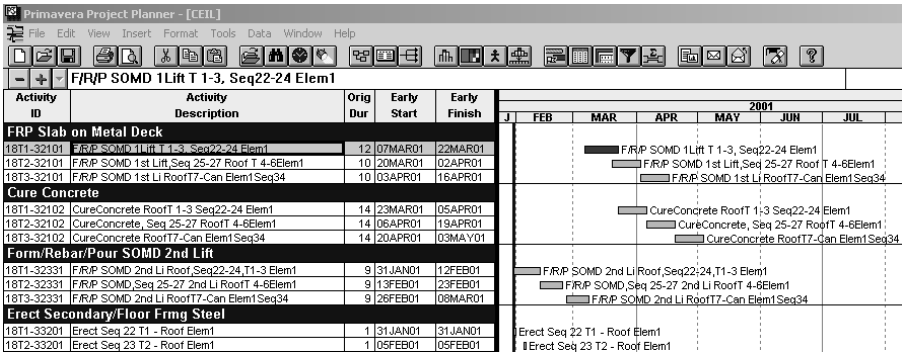

Figure **Error! No text of specified style in document.**.18: Original Schedule in Primavera P3

* The building model and the 4D viewer were provided by Professor Martin Fischer and his research group at Stanford University.

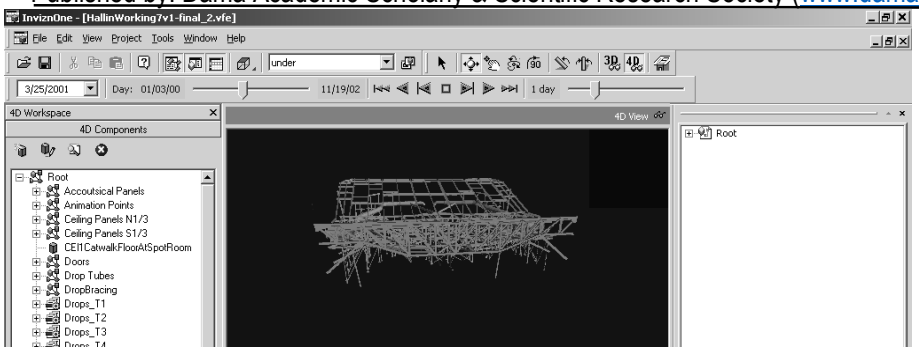

Figure **Error! No text of specified style in document.**.19: Model in 4D Viewer Taken on March 25, 2001

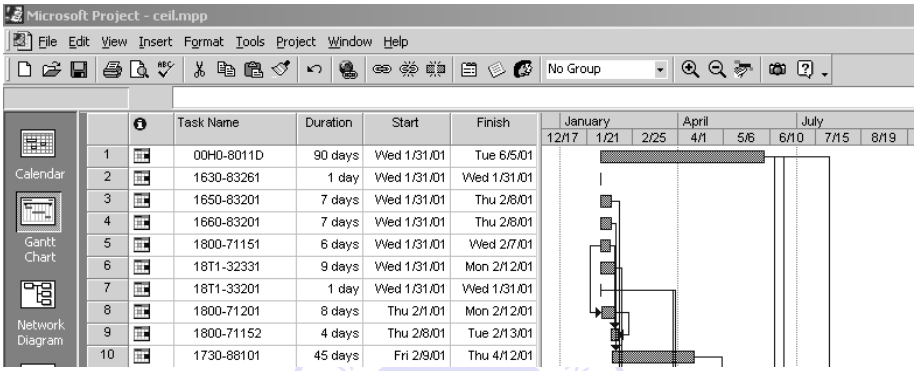

Figure **Error! No text of specified style in document.**.20: Regenerated Gantt Chart in Microsoft Project using PSL

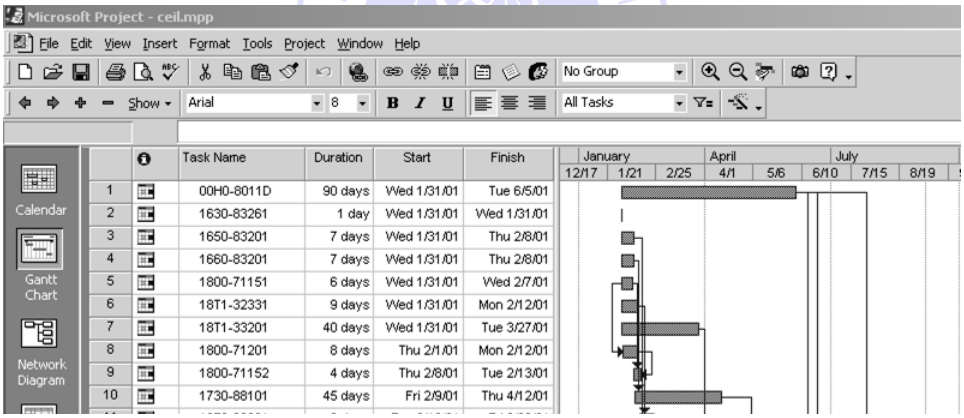

Figure **Error! No text of specified style in document.**.21: Updated Project Schedule in Microsoft Project

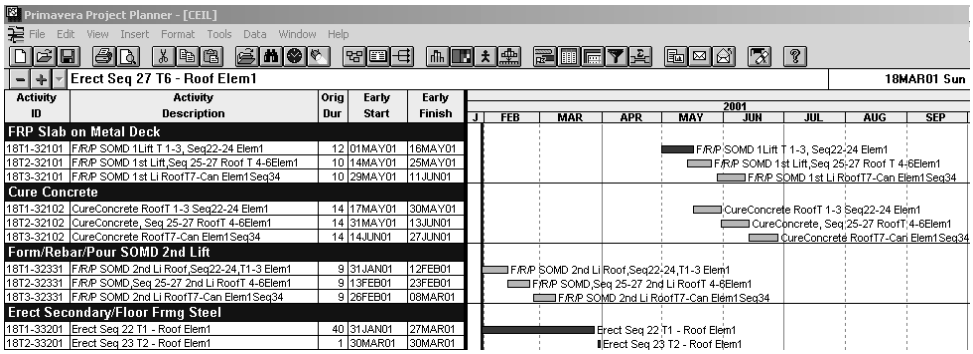

Figure **Error! No text of specified style in document.**.22: Updated Project Schedule in Primavera P3

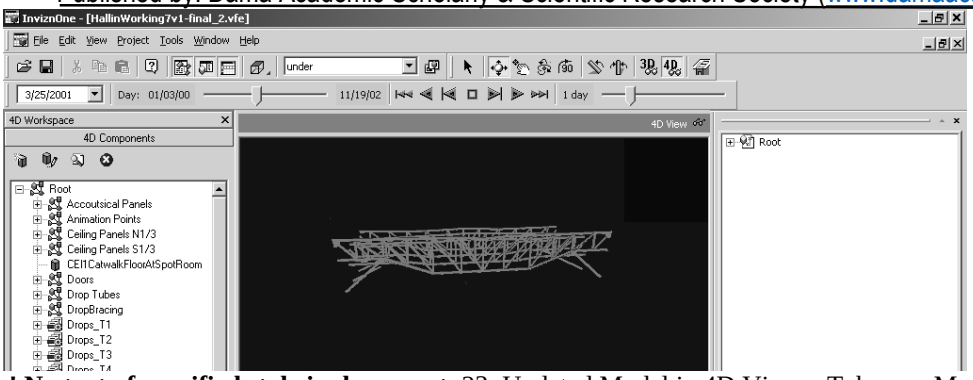

Figure **Error! No text of specified style in document.**.23: Updated Model in 4D Viewer Taken on March 25, 2001

4.0 PSL FOR CONSISTENCY CHECKING

4.1 Conflicts in Project Management

Conflicts are ubiquitous in everyday life. The Oxford English Dictionary defines conflict as "the clashing or variance of opposed principles, statements, arguments, etc." Brown [17] defines conflict as "a form of interaction among parties that differ in interests, perceptions, and preferences." Conflicts are commonplace on large construction projects, since each project by itself has its unique character and thus is a new venture. No two building products are exactly the same. To name a few, design specifications, site conditions, management teams, and financial situations vary from project to project. Experiences gained from past projects cannot be fully transferred to new ones. The information at hand is often conflicting, inconsistent, and incomplete. Project personnel spend a great deal of time handling various conflicts. It has been estimated that as much as 80% of their time is spent dealing with conflicts [84]. Conflicts arise due to many factors and can occur from time to time during the course of a construction project. The interaction of different intellects, beliefs, cultures, personalities, and educational backgrounds may all generate conflicts. Design changes, unexpected weather conditions, labor actions, and procurement delays are common bases for conflicts during various project stages. One method to classify conflicts is by the source of conflicts, as follows [85]:

- Conflict of interests, the discrepancy among involved parties on preferred outcomes.
- Conflict of values, the discrepancy in beliefs or ideologies.
- Conflict of opinions, the discrepancy in the best way to accomplish a shared goal.
- Conflict of information, the discrepancy in project information (e.g., scheduling, resources, physical conditions).

This research focuses on conflicts of information in project management. In a distributed engineering environment, such conflicts occur more often due to incremental changes and miscommunications. For example, a subcontractor may change its sub-schedule without realizing the potential impact on other project participants. Thus, conflicts occur among the schedules of different participants.

4.2 Review of Existing Approaches in Conflict Resolution

There are two major approaches to solving information inconsistency problems in project management. One method is to employ a centralized database, where all project information is stored. The other one is to use various heuristic approaches for specific domain problems. A centralized database enables all project participants to be in tune with the latest information; thus, version conflicts can be eliminated with this approach. However, this approach does not address any logic conflicts. For example, a centralized database can ensure that all participants have the latest information, but it cannot ensure that there are no internal conflicts in the information itself. Heuristic approaches are employed to solve many specific conflicts in project management. For example, Akinci et al. [2] developed a taxonomy to categorize and detect time-space conflicts. The drawback is that it is difficult to generalize such heuristic solutions to handle conflicts that are outside of the defined domain problem.

4.3 Consistency Checking using PSL

PSL can be used to check the consistency of project information from different sources. In particular, PSL can be used to detect 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 the 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 2001-11-15)*, 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 2.24 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 [65, 102]. Otter infers conclusions from given hypotheses 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.

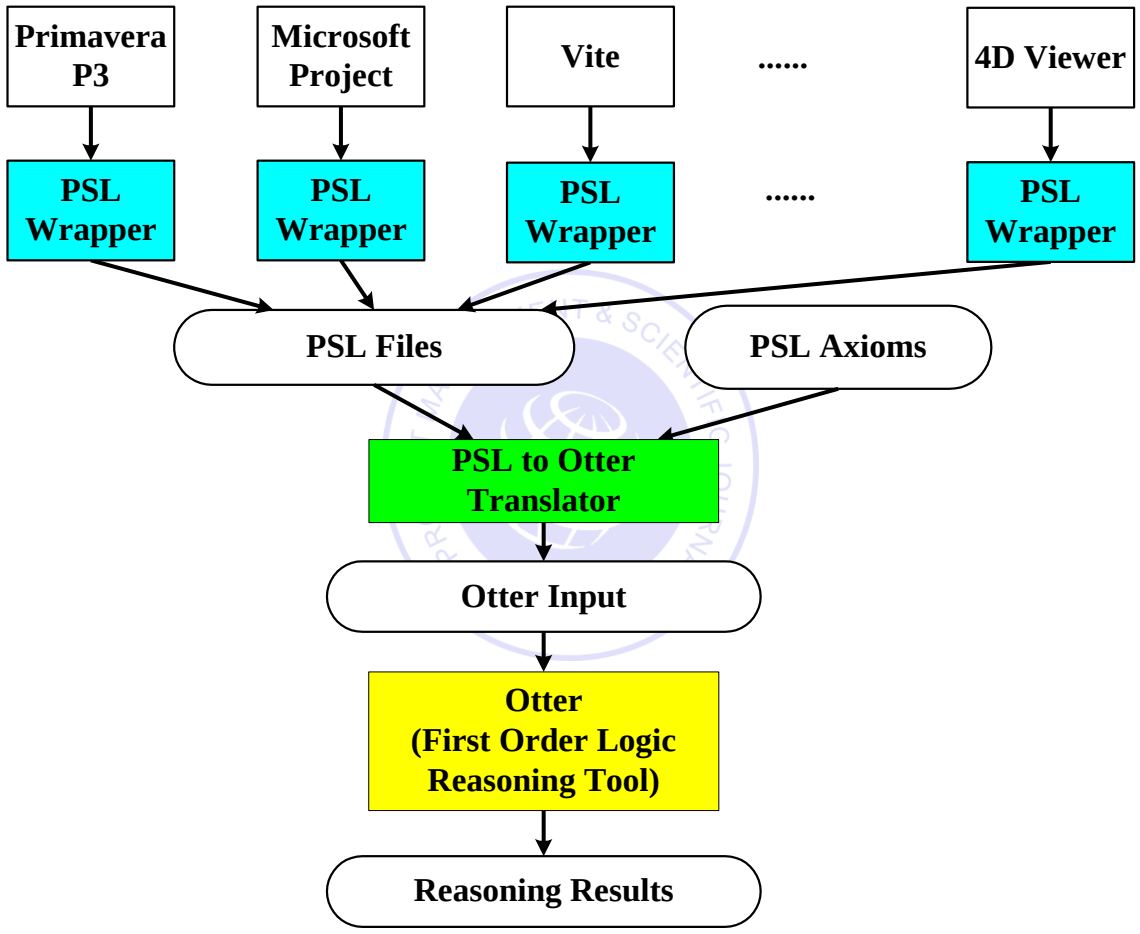

Figure **Error! No text of specified style in document.**.24: Consistency Checking using PSL

To utilize Otter, a translator has been developed to convert PSL files and PSL axioms into first-order logic sentences that Otter can understand. Both PSL and Otter inputs are based on first-order logic, and they have only minor differences in syntax. For example, in PSL a predicate and its variables are all within a pair of parentheses, while in Otter a predicate is followed by a pair of parentheses containing its variables. Figure 2.25 shows the translation from PSL files to Otter inputs.

(activity-occurrence 00H0-8011D) (beginof 00H0-8011D 2001-01-31) (duration-of 00H0-8011D 90)	activity_occurrence(00H0_8011D). beginof(00H0_8011D, 11384). duration_of(00H0_8011D, 90).

(freefloat 00H0-8011D 0) (totalfloat 00H0-8011D 29) (after-start 00H0-8011D 00H0-8011S CEIL) (after-start-delay 00H0-8011D 00H0-8011S CEIL 0.0) **PSL Expressions**	freefloat(00H0_8011D, 0). totalfloat(00H0_8011D, 29). after_start(00H0_8011D, 00H0_8011S, CEIL). after_start_delay(00H0_8011D, 00H0_8011S, CEIL, 0). **Otter Input**

Figure **Error! No text of specified style in document.**.25: Converting PSL Expressions into Otter Input

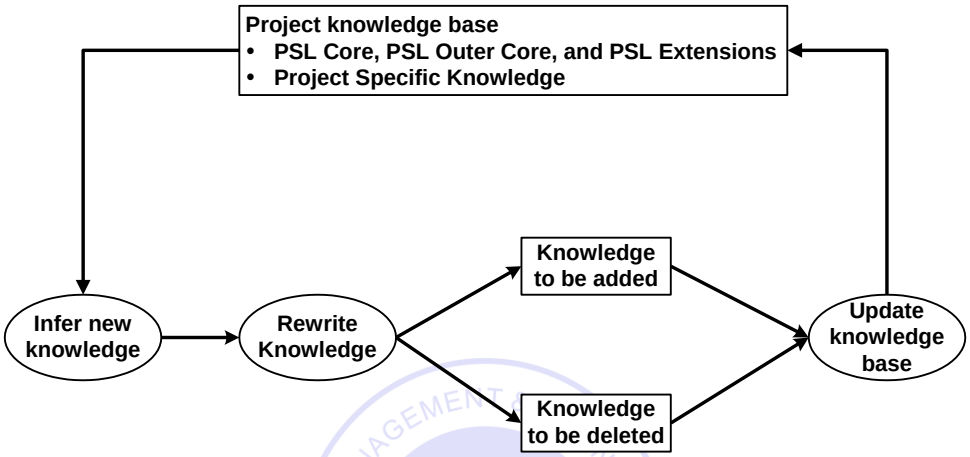

Figure **Error! No text of specified style in document.**.26: Simplified Reasoning Process in Otter

The reasoning process using Otter is summarized in Figure 2.26. Otter first infers new conclusions from the existing knowledge base. Otter then rewrites the new knowledge 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 further conclusions can be inferred. 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 projects 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 the demonstration section, Otter takes only seven seconds to complete the reasoning, as compared to several hours without partitioning.

4.4 Demonstration of the Consistency Checking Prototype

This section presents an example to demonstrate how PSL can be used for consistency checking. A chip design scenario is used to test the potential of PSL for consistency checking purpose. The example project includes the design and fabrication of a chip set for a new personal digital assistant (PDA) product. It 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. We assume that the two groups employ different application software and work on the schedule independently but collaboratively. In addition, we 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 2.27 shows the group 1's schedule in Primavera P3, and Figure 2.28 shows the group's 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 2.29 shows group 2's schedule in Microsoft Project, and Figure 2.30 shows the CPM diagram.

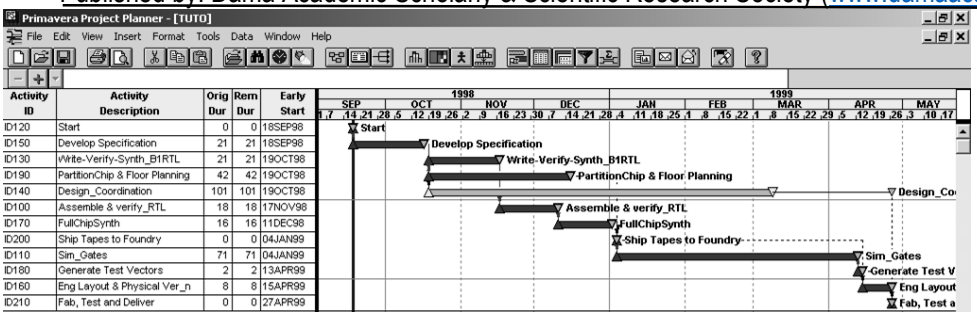

Figure **Error! No text of specified style in document.**.27: Group 1's Schedule in Primavera P3

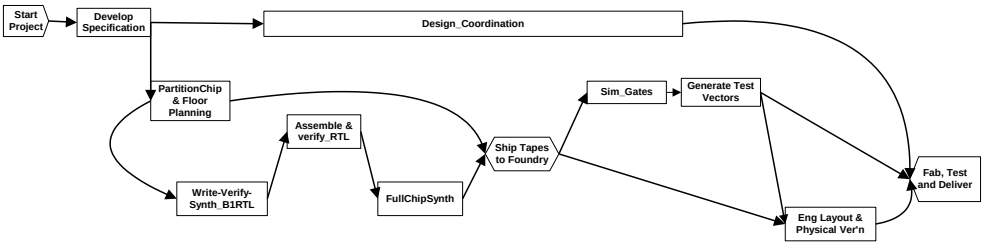

Figure **Error! No text of specified style in document.**.28: Group 1's CPM Diagram

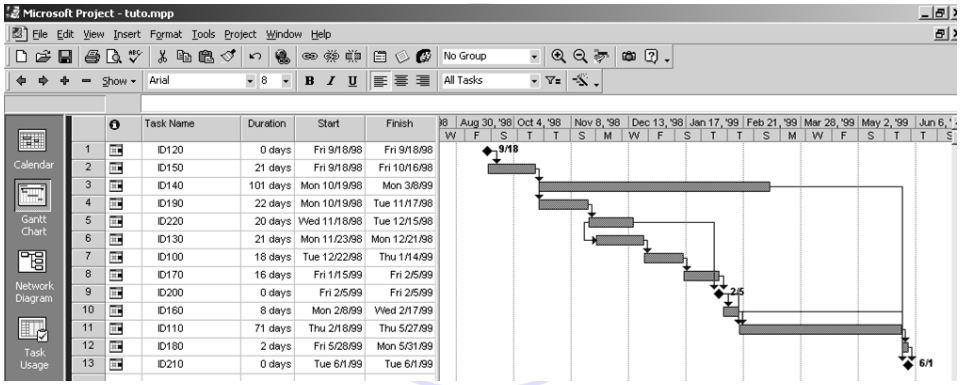

Figure **Error! No text of specified style in document.**.29: Group 2's Schedule in Microsoft Project

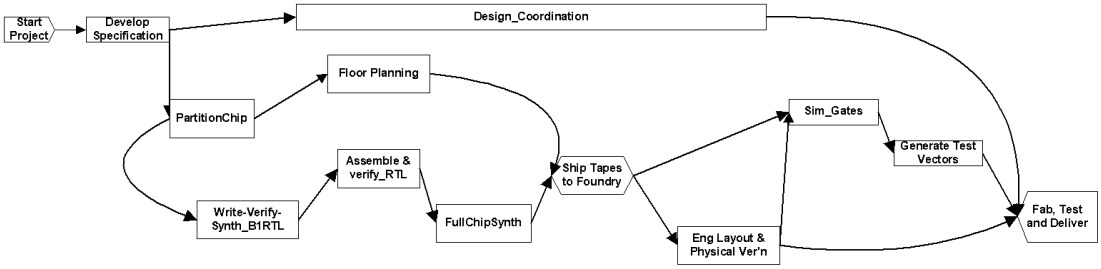

Figure **Error! No text of specified style in document.**.30: Group 2's CPM Diagram

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 2.31 shows the results obtained from the reasoning. In the last sentence, the "$F" indicates that 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 2.32. 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(ID160,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 **Error! No text of specified style in document.**.31: Reasoning Results in Cyclic Dependency Relationships

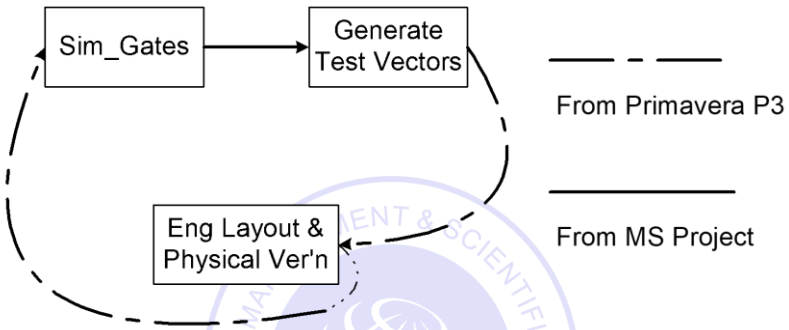

Figure **Error! No text of specified style in document.**.32: Cycle in Dependency Relationships

In addition to logic conflicts in the activity relationships, other conflicts (e.g., conflicts arising due to versioning problems) can also be detected. 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 those activities that have different start dates or durations in Primavera P3 and Microsoft Project.

Figure 2.33 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 2.33 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 2.27, 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 2.29 using Microsoft Project. The last logic sentence in Figure 2.33 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 inconsistencies in the project knowledge base. Following the proof process, we can trace the root of the conflicts, identify the causes, and help resolve the inconsistency problems in the 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 **Error! No text of specified style in document.**.33: Reasoning Results in Version Conflicts

5.0 SOUNDNESS AND COMPLETENESS

The soundness and completeness of the consistency checking prototype largely depends on the soundness and completeness of the employed logic reasoning tool. In this research, it depends on the soundness and completeness of Otter.

Soundness: The consistency checking prototype is sound if a conflict detected by the prototype is indeed a conflict; in other words, the reasoning process is valid. Otter has a very good record on soundness, but no part of it (approximately 28,000 lines of C code) has been formally verified [65]. Thus, any conflicts detected by the prototype may need to be checked manually or by another independent program.

Completeness: The consistency checking prototype is complete if it detects all conflicts in the knowledge base. Theoretically, the proof system of Otter is not complete [65]. In Otter, many strategies have been adopted to save time and memory. These strategies might be incomplete in theory; however, careful use of these strategies does not prevent Otter from finding proofs in practice [65]. To test the soundness and completeness of the consistency checking prototype, four example projects were employed, and various conflicts were created manually. The prototype was then tested on these examples to see if the system is sound and complete. In the demonstration, the following types of conflicts were manually created and mixed into individual example projects:

- Inconsistent dates (e.g., start dates and finish dates)
- Inconsistent task durations
- Cyclic dependency relationships (A cycle may involve two, three, or more activities.)
- Conflicts of resource competition among activities (e.g., two activities compete for one crane at the same time.)

Table 2.5 shows the test results of the prototype system on soundness and completeness. In all test cases, the system successfully detected all the conflicts and did not infer any false conflicts. These experiments illustrate that the theorem prover can be employed reliably for conflict detection and consistency checking applications.

Table **Error! No text of specified style in document.**.5: Soundness and Completeness of the Consistency Checking Prototype

Test	**Soundness**	**Completeness**
Project 1 (1 conflict)	✓	✓
Project 2 (3 conflicts)	✓	✓
Project 3 (8 conflicts)	✓	✓
Project 4 (12 conflicts)	✓	✓

5.1 Performance Analysis of Consistency Checking

Performance is a major issue for many logic reasoning systems. As the knowledge base increases, the required reasoning time usually increases dramatically, which frequently makes a logic reasoning system not useful for practical situations. We examine the performance issue by testing the consistency checking prototype on four

example projects with different sizes. The smallest project consists of 12 activities and 101 pieces of project specific facts, while the largest project has 460 activities and 5147 pieces of project specific facts*. The experiments were conducted on two computers. One is a Dell computer with 1 GB memory and a 2.4 GHZ Intel Pentium IV CPU, and the other is a Dell server with 4 GB memory and four 2.0 GHZ Intel Xeon CPUs.

Section 2.5.3 has briefly discussed the importance of partitioning the input. Basically, the input includes two parts: (1) axioms and definitions from the PSL language specification; and (2) project specific facts. These two parts can be put either into a single list as the input of the reasoning tool, or into two separate lists as the input. By partitioning the input into two separate lists, the reasoning among the PSL axioms and definitions themselves has been eliminated, thus significantly reducing the reasoning time. To illustrate the importance of partitioning, the consistency checking prototype was also tested on the two smaller projects on the Dell server without partitioning the input. The reasoning times on these projects are shown in Table 2.6. In all test cases, a cyclic dependency relationship involving three activities was created for each project. We recorded the time required by the consistency checking system to find all potential conflicts in each project, not the time to find the first conflict.

Table **Error! No text of specified style in document.**.6: Performance Results of the Consistency Checking Prototype

Project	**Project Size**	**Running Time**		
		Partitioning the knowledge base		No-partitioning (on Dell Server)
		Dell PC	Dell Server	
Project 1	12 Activities 101 Pieces of Facts	7 sec	7 sec	> 100 hours
Project 2	58 Activities 541 Pieces of Facts	11min	23 sec	> 100 hours
Project 3	191 Activities 2064 Pieces of Facts	12 min	30 sec	---
Project 4	460 Activities 5147 Pieces of Facts	14 min	1min 23 sec	---

The following observations have been made from the test results on the consistency checking prototype:

- It is difficult to predict the time required to detect conflicts in the knowledge base. However, in general, running time increases non-linearly as the number of facts increases in the project.
- Partitioning the input can dramatically reduce the reasoning time of the consistency checking prototype. Even for a simple project that can be completed in a few seconds with partitioning, the reasoning process fails to complete after more than 100 hours when the input is not partitioned.
- By partitioning the input into two lists, the execution time can be quite reasonable. As shown in Table 2.6, a fast computer with adequate memory can significantly reduce the time of consistency checking even for relatively large projects.

These experiments illustrate the importance of partitioning the knowledge base when applying the theorem prover. Furthermore, the performance of the logic-based reasoning tool is acceptable for practical use in moderate size projects.

* In the experiments with partitioning, only axioms and definitions related with project scheduling are selected from the PSL specification and loaded into the logic reasoning tool; thus, only around 120 pieces of knowledge from the PSL core or extensions are used with partitioning instead of thousands of facts.

5.2 PSL for Constraint Scheduling

Project scheduling is frequently subject to many constraints, such as the targeted completion dates, budget limits, and physical conditions. To develop a good schedule, project personnel need to understand different constraints involved in scheduling. A classification can help resolve the constraint problems and develop appropriate schedules.

5.3 Scheduling Constraint Categorization and Expression

Since the 1970s the Critical Path Method (CPM) has been widely used for project scheduling in the construction industry. Shortly after the adoption of the CPM method, researchers started to realize the need to classify scheduling constraints. For example, Antill and Woodhead [6] classified constraints according to origins, such as physical, hazard, safety, and equipment constraints. Echeverry et al. [33] classified constraints according to physical component relationships, trade interaction, path interference, and code regulations. In this work, the constraints are grouped into three categories: single-task constraints, local constraints between two consecutive tasks, and global constraints that involve tasks. This classification is based on the way such that constraints can be easily expressed in PSL. Single-task constraints are those involving only one task. For example, the duration of a task should be limited to a number of days (e.g., 20 days). Another typical single-task constraint is that a task may have exactly one responsible contractor (e.g., a general contractor or a sub-contractor). In addition, schedulers often need to ensure that there are no isolated tasks in the schedule. In other words, except for the start and finish tasks, each activity needs to be connected to at least one predecessor and one successor. One essential step to expressing these constraints in PSL is the semantic mapping between the constraints and the corresponding PSL terms. The following examples illustrate the mapping.

- The duration of an activity should be limited to N days.

(for all ?a ((<= (duration-of a) N)))

- Except the start activity, each activity should have a predecessor.

(forall ?a (=> (<> a start)
(exist ?a1 (or (before-start ?a1 ?a)
(before-finish ?a1 ?a))))

- Except the finish activity, each activity should have a successor.

(forall ?a (=> (<> a finish)
(exist ?a1 (or (after-start ?a ?a1)
(after-finish ?a ?a1))))

- Each task should have exactly one individual who is directly responsible for it.

(forall ?a (exist ?c (assign a c)))
(forall ?a (=> (assign a c1) (assign a c2) (= c1 c2)))

Local constraints involve two neighboring tasks (e.g., constraints determining the dependency relationships), such as.

- Physical component relationships (support, connect, cover, enclose, protect, etc.)
- Trade interaction (serviced by, damaged by, workspace, resource, etc.)
- Path interference
- Code regulation (inspection, testing, safety, etc.)

Here resource is used as an example to illustrate local constraints. In the example, activity *a1* and activity *a2* require 3 and 5 units of resources respectively. There are only 6 units of resources available; thus, activities *a1* and *a2* cannot be performed in parallel (e.g., activity *a2* may have to start after the completion of activity *a1*.). The constraints can be expressed in PSL as follows:

(resource r)
(resource_point 6)
(demand r a1 3)
(demand r a2 5)
(after-start a1 a2)

Global constraints refer to constraints involving multiple tasks (more than two) and even for the whole project, such as.

- Project costs and durations
- Space constraints of the construction site
- Critical resource constraints (e.g., cranes)
- External constraints (e.g., regulatory permissions, utility access, and unexpected site/weather conditions)

Again, we use an example to illustrate how to express global constraints in PSL. Suppose that on any given date a sub-contractor may not work on more than one task, this constraint can be expressed as follows:

```
(forall ?sub  ?a1 ?a2  ?d
  ( (activity ?a1)
    (activity ?a2)
    (actor ?sub)
    (assigned ?a1 ?sub)
    (assigned ?a2 ?sub)
    (isbetween ?d (beginof ?a1) (endof ?a1))
    (isbetween ?d (beginof ?a2) (endof ?a2))
    (= ?a1 ?a2) ) )
```

5.4 PSL for Constraint Scheduling

To ensure that a schedule meets multiple constraints can be difficult, especially for large, complex schedules with thousands of tasks and constraints. For example, in the San Francisco De Young Museum project, there are over 4,000 tasks and 40 subcontractors. In this project, it took over nine months for the project engineer to develop a schedule that met the various constraints*.

PSL can be used to check whether a schedule meets certain constraints due to its logic structure. Obviously, a PSL wrapper needs to be developed to express scheduling information in PSL, as discussed earlier [25]. In addition, constraints need to be encoded in PSL. A logic-reasoning tool can then be employed to detect potential conflicts between the schedule and the constraints. The following example illustrates the use of PSL in checking whether a schedule meets constraints.

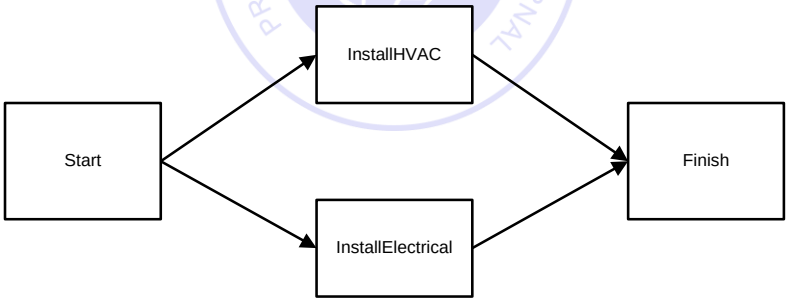

Figure **Error! No text of specified style in document.**.34: An Example Schedule

```
(and (activity-occurrence ID100)
    (doc ID120 "Start")
    ......)
(and (activity-occurrence ID120)
    (doc ID120 "InstallHVAC")
    (beginof ID120 08/05/2002)
    (duration-of ID120 18)
    (after-start ID120 ID160 proj)
    (after-start-delay ID120 ID160 proj 0)
    (demand tool ID120 80)
```

```
(and (activity-occurrence ID140)
    (doc ID140 "InstallElectrical")
    (beginof ID140 08/05/2002)
    (duration-of ID140 15)
    (after-start ID140 ID160 proj)
    (after-start-delay ID140 ID160 proj 0)
    (demand tool ID130 40)
     ......)
(and (activity-occurrence ID180)
    (doc ID120 "Finsih")
```

* This information was obtained from the interview with Mr. Jeff Budke of Swinerton Builders, a project engineer in the San Francisco De Young Museum project.

.....) .....)	.....

Figure **Error! No text of specified style in document.**.35: The Scheduling Information Expressed in PSL

Let us assume that there are four tasks in a simplified schedule: one start task, one finish task, and two tasks in between, as shown in Figure 2.34. Initially, task ID120 ("InstallHVAC") and task ID140 ("InstallElectrical") are scheduled to conduct in parallel, and they require 80 and 40 units of tools, respectively.
The scheduling information can be expressed in PSL expressions, as shown in Figure 2.35. The schedule itself is valid, and there are no internal conflicts. However, if scheduling constraints exist, further analysis is then needed. For example, assume that there are only 100 units of tools available for use, which could be expressed in the following PSL expressions:

(and (resource tool)
(available tool 100))

Potential conflicts may then arise due to resource limitation. With the help of a logic reasoning engine, we can check whether the schedule satisfies the resource constraints. In this example, the available resources are not adequate to conduct tasks *ID120* and *ID140* in parallel, and a conflict is detected. Here *P5* refers to an axiom about the PSL relation *agg_demand*; basically, the aggregate demand for the resource equals the sum of resources required by each activity. *P8* is an axiom dictating that the aggregate demand for resources should not exceed the available quantity. The following shows the reasoning process:

P1: (demand tool ID120 80) {Assertion}
P2: (demand tool ID140 40) {Assertion}
P3: (<> ID120 ID140) {Assertion}
P4: (agg_demand ?r N) {Assertion}
P5: (?r ?a1 ?a2 ?n1 ?n2 ?n (demand ?r ?a1 ?n1)
(demand ?r ?a2 ?n2)
(<> ?a1 ?a2)
(agg_demand ?r ?n)
(= ?n (+ ?n1 ?n2)) {Axiom}
P6: (= N 120)) {P1, P2, P3, P4, P5}
P7: (available tool 100) {Assertion}
P8: (?r ?n1 ?n2 ((agg_demand ?r ?n1)
(available ?r ?n2)
(<= ?n1 ?n2))) {Axiom}
P9: (<= N 100) {P4, P7, P8}
P10: (<= 120 100) {P9, P6}
P11: False {P10}

Adjustments are needed if a schedule fails to meet all constraints. In this example, we can either add more resources or conduct tasks *ID120* and *ID140* in a sequential order. Each option will lead to a schedule satisfying the resource constraint. It should be noted that while the reasoning system is able to check conformity, it does not automatically produce suggestions on alternatives. Instead, project personnel have to develop alternatives and adjust the schedule when resource constraints are detected by the reasoning system.

6.0 CONCLUSION

Data integration has always been a challenge for the engineering and construction industry, where volumes of information are frequently generated by different software applications. Although the Process Specification Language (PSL) was designed with the aim of exchanging process information among manufacturing applications, our research shows that PSL can also be applied to the engineering and construction industry. The mapping between PSL and application concepts is first discussed, followed by the discussion of how to wrap different project management tools.

While each application needs a separate wrapper, the modularity of the wrappers allows share-ability and reusability of the modules among different project management tools. A distributed integration framework is proposed to integrate typical project management applications. The framework has been successfully demonstrated with two example projects. The distributed framework and the data exchange provide ubiquitous access to the project data

from different tools in geographically dispersed locations. PSL is employed for information exchange among project management tools in the simulation access framework to be discussed in the next chapter.

As an ontology standard based on first-order logic, PSL has applications beyond data exchange. This chapter has illustrated the potential applications of PSL in two areas: consistency checking and constraint scheduling. A formal mechanism is proposed to detect conflicts of project information arising from different sources. An example demonstration is provided to illustrate the potential of PSL in consistency checking. In addition, a method is presented to ensure conformity of project schedules to scheduling constraints using PSL.

Reference

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. "The Lorel Query Language for Semistructured Data," *International Journal on Digital Libraries*, 1(1):68-88, 1997.
[2] B. Akinci, M. Fischer, R. Levitt, and R. Carlson. "Formalization and Automation of Time-Space Conflict Analysis," *Journal of Computing in Civil Engineering*, 16(2):124-134, 2002.
[3] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. *BPEL4WS Specification: Business Process Execution Language for Web Services Version 1.1*, http://www-106.ibm.com/developerworks/library/ws-bpel/, 2003.
[4] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. "Natural Language Interfaces to Databases -- An Introduction," *Journal of Natural Language Engineering*, 1(1):29-81, 1995.
[5] A. Ankolekar, M. Burstein, J. R. Hobbs, O. Lassila, D. L. Martin, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, and H. Zeng. "DAML-S: Semantic Markup for Web Services," *Proceedings of the International Semantic Web Working Symposium*, Stanford, CA, 2001.
[6] J. Antill and R. Woodhead. *Critical Path Methods in Construction Practice*, 2nd Ed., John Wiley & Sons, Inc., New York, 1970.
[7] N. Apte and T. Mehta. *UDDI: Building Registry-based Web Services Solutions*, Pearson Education, 2003.
[8] A. Arkin. *Business Process Modeling Language*, Business Process Management Initiative, 2002.
[9] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*, Addison Wesley Ltd, 1999.
[10] A. D. Birrell and B. J. Nelson. "Implementing Remote Procedure Calls," *ACM Transactions on Computer Systems*, 2(1):39-59, 1984.
[11] B. C. Bjork. "Basic Structure of a Proposed Building Product Model," *CAD*, 21(2):71-78, 1989.
[12] A. Bosworth. "Developing Web Services," *Proceedings of the International Conference on Data Engineering*, Heidelberg, Germany, pp. 477-481, 2001.
[13] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte, and D. Winer. *Simple Object Access Protocol (SOAP)*, W3C Note, http://www.w3.org/TR/SOAP, 2000.
[14] E. J. Breck, J. D. Burger, D. House, M. Light, and I. Mani. "Question answering from large document collections," *Proceedings of AAAI Fall Symposium on Question Answering Systems*, North Falmouth, MA, pp. 26-31, 1999.
[15] E. J. Breck, J. D. Burger, L. Ferro, L. Hirschman, D. House, M. Light, and I. Mani. "How to Evaluate Your Question Answering System Every Day and Still Get Real Work Done," *Proceedings of LREC-2000, the 2nd International Conference on Language Resources and Evaluation*, Athens, Greece, 2000.
[16] P. Brothner. "Kawa: Compiling Scheme to Java," *Proceedings of Lisp Users Conference*, Berkeley, CA, 1998.
[17] L. D. Brown. *Managing Conflict Among Groups*, Prentice-Hall, Inc., 1995.
[18] M. Burner. "The Deliberate Revolution Transforming Integration With XML Web Services?," *ACM Queue*, 1(1):28-37, 2003.
[19] J. Burstein, K. Kukich, S. Wolff, C. Lu, and M. Chodorow. "Computer Analysis of Essays," *Proceedings of NCME Symposium on Automated Scoring*, 1998.
[20] C. Callison-Burch and P. Shilane. *A Natural Language Question and Answer System*, Stanford University, Stanford, CA, 2000.
[21] S. Chandrasekaran, G. Silver, J. Miller, J. Cardoso, and A. Sheth. "Web Service Technologies and their Synergy with Simulation," *Proceedings of the 2002 Winter Simulation Conference (WSC'02)*, San Diego, CA, pp. 606- 615, 2002.
[22] J. Cheng, B. Kumar, and K. H. Law. "A Question Answering System for Project Management Applications," *Advanced Engineering Informatics*, 16(4):277-289, 2002.
[23] J. Cheng and K. H. Law. "Using Process Specification Language for Project Information Exchange," *Proceedings of the 3rd International Conference on Concurrent Engineering in Construction*, Berkeley, CA, pp. 63-74, 2002.

[24] J. Cheng, P. Trivedi, and K. H. Law. "Ontology Mapping Between PSL and XML-Based Standards For Project Scheduling," *Proceedings of the 3rd International Conference on Concurrent Engineering in Construction*, Berkeley, CA, pp. 143-156, 2002.
[25] J. Cheng, M. Gruninger, R. D. Sriram, and K. H. Law. "Process Specification Language for Project Information Exchange," *International Journal of Information Technology in Architecture, Engineering and Construction*, 1(4):307-328, 2003.
[26] J. Cheng, K. H. Law, and B. Kumar. "Integrating Project Management Applications as Web Services," *Proceedings of the 2nd International Conference on Innovation in Architecture, Engineering and Construction*, Loughborough University, UK, 2003.
[27] J. Cheng, K. H. Law, and B. Kumar. "Online Collaboration of Project Management Applications," *Proceedings of the 11th International Workshop of the EG-ICE*, Weimar, Germany, 2004.
[28] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. *WSDL. Web Services Description Language*, W3C Note, World Wide Web Consortium, http://www.w3.org/TR/wsdl, 2000.
[29] A. Diekema, X. Liu, J. Chen, H. Wang, N. McCracken, O. Yilmazel, and E. D. Liddy. "Question Answering : CNLP at the TREC-9 Question Answering Track," *Proceedings of Proceedings of the 9th Text REtrieval Conference (TREC-9)*, National Institute of Standards and Technology, Gaithersburg, MD, pp. 501-510, 2000.
[30] A. M. Dubois and F. Parand. "COMBINE Integrated Data Model," *Proceedings of CIBSE National Conference*, Manchester, UK, pp. 96-108, 1993.
[31] C. L. Dym and R. E. Levitt. *Knowledge-Based Systems In Engineering*, McGraw-Hill, Inc., 1991.
[32] C. M. Eastman. *Building Product Models: Computer Environments Supporting Design and Construction*, CRC Press, 1999.
[33] D. Echeverry, W. Ibbs, and S. Kim. "Sequence Knowledge for Construction Scheduling," *Journal of Construction Engineering and Management*, 117(1):118-130, 1991.
[34] B. Eisenberg, A. Grangard, and D. Nickull. *ebXML Technical Architecture Specification v1.0.4*, Organization for the Advancement of Structured Information Standards, http://www.ebxml.org/specs/ebTA.pdf, 2001.
[35] Fatdog. *XQEngine Introductory Tutorial*, Fatdog Software, http://www.fatdog.com/tutorial.html, 2002.
[36] C. Fellbaum and G. Miller. *WordNet: An Electronic Lexical Database (Language, Speech, and Communication)*, MIT Press, 1998.
[37] J. Fowler. *STEP for Data Management, Exchange and Sharing*, Technology Appraisals Ltd., UK, 1995.
[38] G. Frank. "A General Interface for Interaction of Special-Purpose Reasoners within a Modular Reasoning System," *Proceedings of Question Answering Systems, Papers from the 1999 AAAI Fall Symposium*, North Falmouth, Massachusetts, pp. 57-62, 1999.
[39] T. Froese, M. Fischer, F. Grobler, J. Ritzenthaler, K. Yu, S. Sutherland, S. Staub, B. Akinci, R. Akbas, B. Koo, A. Barron, and J. Kunz. "Industry Foundation Classes for Project Management-A Trial Implementation," *Electronic Journal of Information Technology in construction*, 4:17-36, 1999.
[40] GAO. *Computer Aided Building Design*, GAO, 1978.
[41] F. K. Garas and I. Hunter. "CIMSteel (Computer Integrated Manufacturing in Constructional Steelwork) - Delivering the Promise," *Structural Engineering*, 76(3):43-45, 1998.
[42] H. Garcia-Molina, J. D. Ullman, and J. D. Widon. *Database Systems: The Complete Book*, Prentice Hall, 2001.
[43] M. R. Genesereth and R. Fikes. *Knowledge Interchange Format Reference Manual - Version 3*, Report No. CSD-Logic-92-1, Department of Computer Science, Stanford University, Stanford, CA, 1992.
[44] E. F. Gould. *Managing the Construction Process: Estimating, Scheduling, and Project Control*, Prentice Hall, 2002.
[45] E. Hovy and C. Y. Lin. *Automated Text Summarization in SUMMARIST*, MIT Press, 1999.
[46] IAI. *Industry Foundation Classes*, International Alliance for Interoperability, Washington, DC, 1997.
[47] IAI. *AecXML*, International Alliance for Interoperability, http://www.aecxml.org, 2002.
[48] IGES. *Initial Graphics Exchange Standard (IGES), Version 5.1*, National Bureau of Standards, Gaithersburg, MD, 1991.
[49] ISO. *Industrial Automation Systems-Product Data Representation and Exchange*, International Organization for Standardization, 1989.
[50] ISO. *EXPRESS Language Reference Manual: External Representation of Product Definition Data*, ISO DIS10303 Part 11, 1991.

[51] P. Jacobs and L. Rau. "SCISOR: Extracting Information from On-line News," *Communications of the ACM*, 33(11):88-97, 1990.
[52] V. Karamcheti and A. A. Chien. "A Comparison of Architectural Support for Messaging in the TMC CM-5 and the Cray T3D," *Proceedings of ISCA 95*, Santa Margherita, Italy, pp. 298-307, 1995.
[53] D. Klein and C. Manning. "Parsing with Treebank Grammars : Empirical Bounds, Theoretical Models, and the Structure of the Penn Treebank," *Proceedings of the 39th Annual Meeting of the ACL*, Toulouse, France, pp. 330-337, 2001.
[54] K. H. Law and M. K. Jouaneh. "Data Modeling for Building Design," *Proceedings of the 4th Conference on Computing in Civil Engineering*, ASCE, pp. 21-36, 1986.
[55] K. H. Law, D. L. Spooner, and M. K. Jouaneh. "Abstraction Database Concepts for Engineering Modeling," *Engineering with Computers*, 2(2):79-84, 1987.
[56] K. H. Law, T. Barsalou, and G. Wiederhold. "Management of Complex Structural Engineering Objects in a Relational Framework," *Engineering with Computers*, 6:81-92, 1990.
[57] S. M. Lewandowski. "Framework for Component-Based Client/Server Computing," *ACM Computing Surveys*, 30(1):3-27, 1998.
[58] F. Leymann. *Web Services Flow Language (WSFL 1.0)*, IBM Corporation, http://www4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf, 2001.
[59] T. Liebich. *XML Schema Language Binding of EXPRESS for ifcXML*, International Alliance for Interoperability, 2001.
[60] D. Liu, K. H. Law, and G. Wiederhold. "Data-flow Distribution in FICAS Service Composition Infrastructure," *Proceedings of the 15th International Conference on Parallel and Distributed Computing Systems*, Louisville, KY, 2002.
[61] D. Liu, J. Peng, K. H. Law, G. Wiederhold, and R. D. Sriram. "Composition of Autonomous Services with Distributed Data Flows and Computations," *submitted to ACM Transactions on Internet Technology*, http://mediator.stanford.edu/papers/FICAS.pdf, 2003.
[62] D. W. Liu. *A Distributed Data Flow Model For Composing Software Services*, Ph.D. Thesis, Department of Electrical Engineering, Stanford University, 2003.
[63] M. P. Marcus, B. Santorini, and M. A. Marcinkiewica. "Building a large annotated corpus of English: the Penn Treebank," *Computational Linguistics*, 19(2):310-330, 1993.
[64] R. Mayer and J. Linden. *Using Iges, Dxf and Cals for Cad/Cam Data Transfer: The Complete Guide to Data Transfer Standards*, Management Roundtable, 1992.
[65] W. W. McCune. *Otter 3.0 Reference Manual and Guide*, Report No. ANL-94/6, Mathematics and Computer Science Division, Argonne National Laboratory, 1994.
[66] K. McKinney and M. Fischer. "Generating, Evaluating and Visualizing Construction Schedules with CAD Tools," *Automation in Construction*, 7(6):433-447, 1998.
[67] C. Menzel and M. Gruninger. "A Formal Foundation for Process Modeling," *Proceedings of Formal Ontology in Information Systems*, Ogunquit, Maine, pp. 256-269, 2001.
[68] P. Naur and J. Backus. "Report on the Algorithmic Language ALGOL 60," *Communications of the ACM*, 3(5):299-314, 1960.
[69] NRC. *The 1984 Workshop on Advanced Technology for Building Design and Engineering*, Building Research Board, NRC, 1985.
[70] NRC. *The 1985 Workshop on Advanced Technology for Building Design and Engineering*, Building Research Board, NRC, 1986.
[71] NRC. *The 1986 Workshop on Integrated Data Base Development for the Building Industry*, Building Research Board, NRC, 1987.
[72] S. K. Park, K. I. Lim, and E. D. Kim. "A STEP-based Integrated Structural Design System for Steel Framed Buildings," *Proceedings of the 8th International ASCE Conference on Computing and Building Engineering*, Stanford, CA, pp. 788-795, 2000.
[73] F. Pereira. *Logic for Natural Language Analysis*, Technical Note 275, SRI International, 1983.
[74] E. Pitt and K. McNiff. *java.rmi: The Remote Method Invocation Guide*, Addison Wesley, 2001.
[75] A. Pope. *The CORBA Reference Guide: Understanding the Common Object-Request Broker Architecture*, Addison Wesley, 1998.
[76] J. Robie. *XQL Tutorial*, Software AG, http://ibiblio.org/xql/xql-tutorial.html, 1999.
[77] E. Roman, S. W. Ambler, and T. Jewell. *Mastering Enterprise JavaBeans*, 2nd Ed., John Wiley & Sons, 2001.
[78] J. Roy and A. Ramanujan. "Understanding Web Services," *IT Professional*, 3(6):69-73, 2001.

[79] N. Sample, D. Beringer, L. Melloul, and G. Wiederhold. "CLAM: Composition Language for Autonomous Megamodules," *Proceedings of the 3rd International Conference on Coordination Models and Languages*, Amsterdam, Netherland, pp. 291-306, 1999.
[80] C. Schlenoff, M. Ciocoiu, D. Libes, and M. Gruninger. "Process Specification Language: Results of the First Pilot Implementation," *Proceedings of the International Mechanical Engineering Congress and Exposition*, Nashville, Tennessee, pp. 529-539, 1999.
[81] C. Schlenoff, M. Gruninger, and M. Ciocoiu. "The Essence of the Process Specification Language," *Transactions of the Society for Computer Simulation*, 16(4):204-216, 1999.
[82] I. Schröder. *Ingo's Collection Of POS Taggers*, http://nats-www.informatik.uni-hamburg.de/~ingo/icopost/, 2002.
[83] J. Siméon. *Galax Implementation of XQuery*, XQuery Implementation Panel, XML 2001, Orlando, http://db.bell-labs.com/galax/, 2002.
[84] A. Singh and D. A. Vlatas. "Using Conflict Management for Better Decision Making," *Journal of Management in Engineering*, 7(1):70-82, 1989.
[85] A. Singh and H. M. Johnson. "Conflict Management Diagnosis at Project Management Organizations," *Journal of Management in Engineering*, 14(5):48-63, 1998.
[86] Sourceforge. *Sourceforge Xquench Project*, Open Source Development Network, http://sourceforge.net/projects/xquench/, 2002.
[87] STEP. *ST-Developer*, STEP Tools Inc., Rensselaer Technology Park, Troy, NY, 1997.
[88] T. L. Thai. *Learning DCOM*, O'Reilly & Associates, 1999.
[89] S. Thatte. *XLANG: Web Services For Business Process Design*, Microsoft Corporation, 2001.
[90] D. Vanier. "Product Modeling: Helping Life Cycle Analysis of Roofing Systems," *Proceedings of the Life Cycle of Construction IT Innovations*, Stockhelm, Sweden, pp. 423-235, 1998.
[91] M. Vasconcellos and M. Leon. "SPANAM and ENGSPAN: Machine translation at the Pan American Health Organisation," *Computation Linguistics*, 11(2-3):122-136, 1985.
[92] Vite. *SimVision Help*, Vite SimVision Help Manual, Vite Corporation, 2002.
[93] W3C. *XML-QL: A Query Language for XML*, World Wide Web Consortium, http://www.w3.org/TR/NOTE-xml-ql/, 1998.
[94] W3C. *XML Path Language (XPath) Version 1.0*, World Wide Web Consortium, Recommendation 16, 1999.
[95] W3C. *XQuery 1.0: An XML Query Language*, W3C Working Draft 20, 2001.
[96] Webcor. *Ronal McDonal Housing Expansion Quarterly Project Review*, Webcor Builders, April 2003.
[97] WebGain. *Java Compiler Compiler (JavaCC) - The Java Parser Generator*, http://www.webgain.com/products/java_cc/, 2001.
[98] G. Wiederhold. *Strategic Uses of Information Technologies*, Presentation at Stanford Graduate School of Business, Stanford, CA, 1996.
[99] G. Wiederhold, R. Jiang, and H. Garcia-Molina. "An Interface Language for Projecting Alternatives in Decision-Making," *Proceedings of AFCEA Database Colloquium*, San Diego, CA, 1998.
[100] G. Wiederhold and H. Garcia-Molina. "SimQL: an Interface for Integrating Access to Simulations into Information Systems," *Proceedings of DARPA-JFACC Symposium*, San Diego, pp. 259-262, 1999.
[101] W. A. Woods. "Progress in Natural Language Understanding: An Application to Lunar Geology," *Proceedings of AFIPS Conference*, pp. 441-450, 1973.
[102] L. Wos and G. W. Pieper. *A Fascinating Country in the World of Computing: Your Guide to Automated Reasoning*, World Scientific Publishing Company, Singapore, 2000.
[103] P. Wyckoff, S. W. McLaughry, T. J. Lehman, and D. A. Ford. "TSpaces," *IBM Systems Journal*, 37(3):454-474, http://www.almaden.ibm.com/cs/TSpaces, 1998.
[104] M. J. Young. *Step by Step XML*, Microsoft Press, 2001.
[105] R. Zajac. "Towards Ontological Question Answering," *Proceedings of ACL Open Domain Question Answering Workshop*, Toulouse, 2001.