PSL Quarterly Progress Report

Project Title: Process Specification and Simulation
Date: September 30, 2001
Principal Investigator: Kincho H. Law, Stanford University
Period Covered: July 1, 2001 – September 30, 2001

Project Objectives

The proposed project is intended to be a feasibility study to evaluate the process specification language (PSL) and a simulation query language (SimQL) with application to project and workflow management. This proposed joint research effort includes:

- to evaluate the potential of the Process Specification Language (PSL) for the planning and modeling activities in a project
- to evaluate the adequacy of the Process Specification Language (PSL) as an interchange definition language to support process-oriented simulation
- to evaluate the applicability of SimQL, a simulation query language, for practical engineering problems
- to develop an integration framework using PSL and SimQL for process-oriented simulation

Our long-term goal is to develop a distributed network-based framework to integrate process specification and modeling and virtual simulations of project activities. To facilitate this research, we use Vite, which is originally developed at Stanford’s Center for Integrated Facility Engineering as a benchmarking application for the evaluation of PSL and SimQL.

Progress and Results

Our first goal in this project is to evaluate PSL as process specification interchange standard using Vite as a benchmark application. Vite is a project and organization modeling system designed to assist in developing organizational structures and identifying potential problems with project cost, time, or quality. It takes traditionally qualitative organizational management theory and builds a model that incorporates rough quantitative measures. As for this investigation, we have built a sample demonstration using PSL as an interchange format to exchange information among Primavera’s P3, Microsoft Project 2000, 4DViewer and Vite. Primavera’s P3 and Microsoft Project 2000 are project scheduling software widely used in the construction industry. 4D Viewer is an application developed by 4D research group at Stanford University, which could dynamically display the construction progress. The translation process among Primavera P3, Microsoft Project and Vite using PSL is summarized as shown in Figure 1.
As shown in figure 1, we have also built translator between PSL and Oracle Database, so that project information could be viewed and updated on any web browser and palm. Beyond that, we link PSL file with logic reasoning tool so that we can reason on PSL knowledge. In summary, three basic tasks have been performed during this report period.

- We select some widely used applications in construction industry, and build PSL wrapper for each of those applications. So that those applications are PSL compliant and could interoperate with each other.

- We also build translator between PSL and oracle database. So that project manager could view and update project information on site using web browser or palm. Here we test our prototype system on Mortenson Ceiling Project, which is a portion of the construction of Walt Disney Concert Hall.

- We have begun an initial investigation on the reasoning power of PSL.
**PSL Wrapper**

To exchange project information among different construction applications, we need to build a wrapper for each application, which could retrieve the information out from application and convert it into PSL format, and could also parse the information out from PSL file and feed it back into application.

To exchange project information, first we need to map the concepts in different applications into PSL ontology. The table below (Table 1) shows some terms in P3 and PSL, which are related with activity relationship.

<table>
<thead>
<tr>
<th>Concepts in P3</th>
<th>PSL Ontology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successor</td>
<td>Successor</td>
</tr>
<tr>
<td>Predecessor</td>
<td>Follows</td>
</tr>
<tr>
<td></td>
<td>after-start</td>
</tr>
<tr>
<td></td>
<td>after-start-delay</td>
</tr>
<tr>
<td></td>
<td>…………….</td>
</tr>
</tbody>
</table>

For example, activity B is the successor of activity A in construction project P in P3. The time lag is 3 days, and relationship type is FinishToStart, which is illustrated in figure 8.

If we translate the successor concepts into PSL ontology, we will have:
- (activity-occurrence A)
- (activity-occurrence B)
- (subactivity-occurrence A P)
- (subactivity-occurrence B P)
- (after-start A B P)
- (after-start-delay A B 3)

The basic process of PSL based project information exchange could be illustrated in picture 3.
Information Exchange Among Vite, P3 and MS Project

We select Mortenson Ceiling Project to demonstrate the translation. Mortenson Ceiling Project is a portion of the construction of the Walt Disney Concert Hall, built by Mortenson Construction, and designed by Frank O.Gehry & Associates. The curvilinear ceiling for a major concert hall is designed as light gage steel, hung from the trusses, as shown in figure 4.
In our prototype system, we use oracle database as our backbone system. Through oracle database, on-site personnel can view and update project information using a PDA or a desktop browser (Figure 9). Applications such as P3, Project, 4DViewer and Vite could also interoperate with each other through a process interchange standard (PSL).
Figure 6 is the 4D Model of Mortenson Ceiling Project taken on 3/25/2001 from 4D Viewer. Figure 7 and 8 show the Original Gantt Chart of Mortenson Ceiling Project in Primavera P3 and Microsoft Project. On site personnel can view and update project information through palm or web browser, as shown in Figure 9. Figure 10 and 12 show the modified Gantt Chart in Primavera P3 and Microsoft Project 2000. Figure 11 show the modified 4D Model of Mortenson Ceiling Project taken on 3/25/2001 from 4D Viewer. As shown in Figure 13, project manager can view the updated project information on web browser or any other e-machine.
Figure 7. Original Gantt chart in Primavera P3

Figure 8. Original Gantt Chart in MS Project (Generated from PSL file)
History

SCHEDULE

Review the schedule and make appropriate updates by changing the value in duration:

<table>
<thead>
<tr>
<th>SCHEDULEID</th>
<th>STARTDATE</th>
<th>DURATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>18T1-33201</td>
<td>01-31-2001</td>
<td>40</td>
</tr>
</tbody>
</table>

Update

18T1-33241

Figure 9. View and Update Project Information on Palm

Figure 10. Modified Gantt Chart in Primavera P3
Figure 11. Modified 4D Model Taken on 3/25/2001 From 4Dviewer

Figure 12. Modified Gantt Chart in MS Project (Generated from PSL file)
Figure 13. View Modified Project Information on Web Browser

PSL Reasoning

The underlying grammar used for PSL is based on KIF (Knowledge Interchange Format). KIF is a formal language based on first-order logic. All the knowledge about one project could be stored in PSL file, regarding whether it is from P3, Microsoft Project, Vite or 4D Viewer. Since all the terms in PSL are formally defined using first order logic, we can use some reasoning tool to reasoning on the knowledge base. From the reasoning, we can:

- Detect conflict in the knowledge database. It could be some simple conflict that could be easily detected by human being or some software. It could also be some deep logic conflict in the knowledge database that is hard to detect, since the project knowledge could come from heterogeneous resources.
• Identify the source of conflict and maintain the consistency of knowledge database. After we detected the conflict, we can trace back to the source of conflict and even solve the confliction.

• Infer some interesting conclusions, which might be not obvious. When project information comes from heterogeneous sources, some important project characteristics might not be obvious. However, reasoning tool can infer interesting conclusions from existing project information and help project manager to understand the project better.

The picture below (Figure 14) illustrates the basic reasoning process. First we infer some new conclusions from existing knowledge base. For the new knowledge, we will rewrite it and check whether it is subsumed by existing knowledge. If not, we could add the new knowledge to the knowledge base. Otherwise, we just delete it. Usually, the reasoning will stop either it find some conflict or no more conclusion could be inferred.

Figure 14. Reasoning on PSL knowledge
The knowledge base where the reasoning tool is going to reason on includes two main parts:

- Axioms and definitions from PSL Core, PSL outer core, PSL Extensions and Vite Extensions
- Facts of individual project form heterogeneous resources

The reasoning among the axioms and definitions is not our interest. That reasoning not only produces a lot of uninteresting results but also significantly slows the reasoning process. Instead, we focus on the reasoning among the knowledge of specific project, and the reasoning between specific project knowledge and those axioms and definitions.

At the first step, we use Otter to reason on some sample project. Otter (Organized Techniques for Theorem-proving and Effective Research) is a resolution-style theorem-proving program for first order logic with equality. Otter includes the inference rules such as binary resolution, hyperresolution, UR_resolution, and binary paramodulation. It takes two types of input: logic clause or first order logic sentences. Since PSL is based on first order logic, it is very easy to convert PSL file into first order logic sentences, which Otter could read. Here is an example that we use Otter to detect inconsistency of PSL knowledge:

16 [] -activity_occurrence(x11)|occurrence_of(x11,$f1(x11))
19 [] -occurrence_of(x19,x20)|-duration(x19,x21)| -beginof(x19,x22)|endof(x19,x21+x22)
39 [] -after_start(x71,x72,x73)| -occurrence_of(x71,x74)| -occurrence_of(x72,x75)| -endof(x71,x76)|-beginof(x72,x77)|x76<=x77
51 [] activity_occurrence(AssembleverifyRTL).
52 [] beginof(AssembleverifyRTL,42300).
53 [] duration(AssembleverifyRTL,8640).
54 [] after_start(AssembleverifyRTL,FullChipSynth,Tutorial).
76 [] activity_occurrence(FullChipSynth).
77 [] beginof(FullChipSynth,30940).
115 [hyper,51,16] occurrence_of(AssembleverifyRTL,$f1(AssembleverifyRTL)).
125 [hyper,76,16] occurrence_of(FullChipSynth,$f1(FullChipSynth)).
139 [hyper,115,19,52,demod] endof(AssembleverifyRTL,50940).
144 [hyper,125,39,54,115,139,77,demod,propositional] SF.

From the example above, we can see that there are some confictions in the project knowledge base. Following the proof process, we can trace back to the root of confliction, identify and solve the inconsistency problem in the project.