First Quarterly Progress Report
Winter Quarter, 2000

Project Title: Process Specification and Simulation
Date: December 30, 2000
Principal Investigator: Kincho H. Law, Stanford University
Period Covered: October 1, 2000 – December 30, 2000

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

As noted in the proposal, our first task is to evaluate PSL as process specification interchange standard using Vite as a benchmark application. As for this initial investigation, we have built a sample demonstration using PSL as an interchange format to exchange information between Primavera's P3 and Vite. (Primavera is a project scheduling software widely used in the construction industry.) Presently, the demonstration application includes two translators: Vite to PSL(KIF) translator and PSL(KIF) to P3 translator.

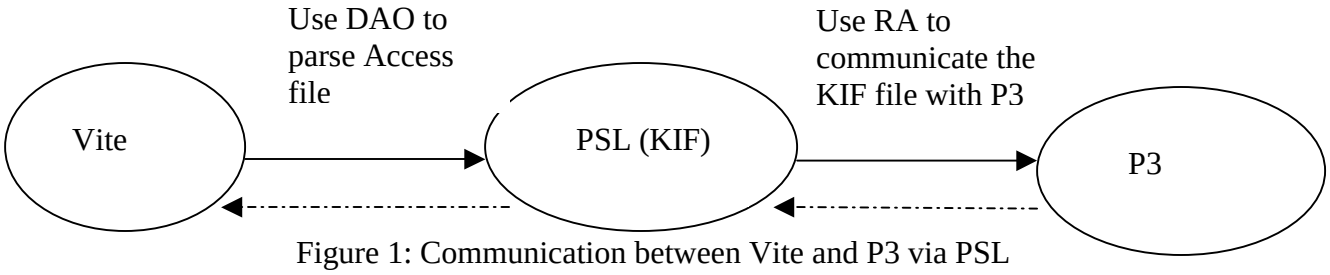

Figure 1: Communication between Vite and P3 via PSL

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. A Vite project is composed of a traditional CPM diagram, additional links showing failure dependence and reciprocal information dependence, a management structure diagram, and responsibility links between the management structure and activities. In this example scenario, we focus on interchanging activity information between Vite and P3. A typical CPM activity diagram is shown in Figure 2. Activities are named and given durations, and groups of activities are linked together using relationships such as finish-start, start-start, or finish-finish. Milestones may be included in the activity diagram, such as "ship tapes to foundry" in the example project. Begin and end milestones are used to denote the start and finish of the project ("Start Project" and "Fab, Test and Deliver.")

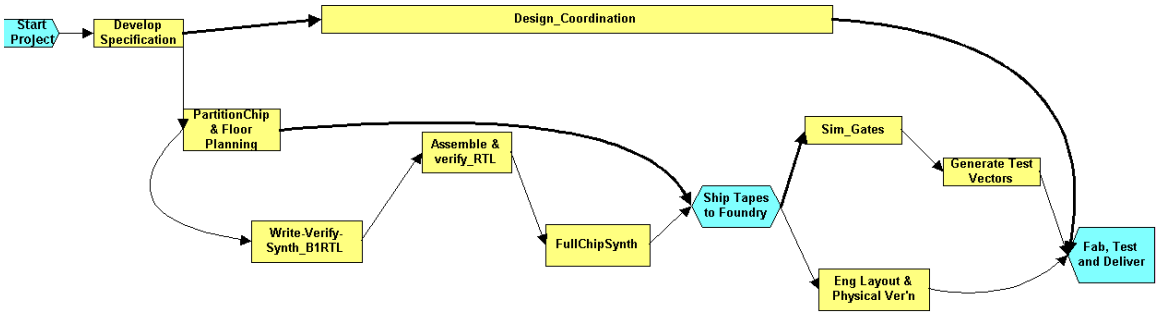

Figure 2: Traditional CPM diagram in Vite

Currently, Vite does not provide any API. The simulation results of Vite are stored in an Access database file format. Vite exports project information to an Access database, which contains the following relational tables: ViteActivities, ViteActivityStatistics, ViteActorCrafts, ViteActors, ViteActorStatistics, ViteActorTrace, ViteAssignments, ViteDependencies, ViteMeetings, ViteProjects, ViteProjectStatistics, ViteProjectSuccessors, ViteReciprocals, ViteScenariors, ViteScenarioStatistics, ViteSuccessors, ViteSupervisions, and ViteTeams. The only interesting tables in the current example scenario are ViteActivityStatistics, ViteScenariors and ViteSuccessors. From the ViteScenariors table we can calculate the total number of scenarios in the simulation results. From ViteActivityStatistics we can obtain activity information of the final scenario. The relationships between activities (such as finish-start relationship) are contained in ViteSuccessors table.

For the Vite to PSL(KIF) translator, We map the concepts in Vite into the formal ontology described in PSL, that explicitly and unambiguously defines all terms introduced within the language. Then we parse relevant information stored in the Access database using DAO (Data Access Object), translate the information into PSL(KIF) according to a set of rules, and create a PSL(KIF) file.

For the PSL(KIF) to P3 translator, we use RA (Primavera Automation Engine) to communicate with P3. 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 RA to parse the activity information stored in PSL(KIF) formate and to communicate with P3. Finally, P3 re-constructs the project schedule, based on the PSL(KIF) formatted information.

The sample demo is illustrated as shown in Figures 3 and 4. Figure 3 shows the Ghant chart for the Vite project activtities shown in Figure 2. The PSL(KIF) file is generated as shown in Appendix A. Finally, the scheduling information as produced by P3 is shown in Figure 4.

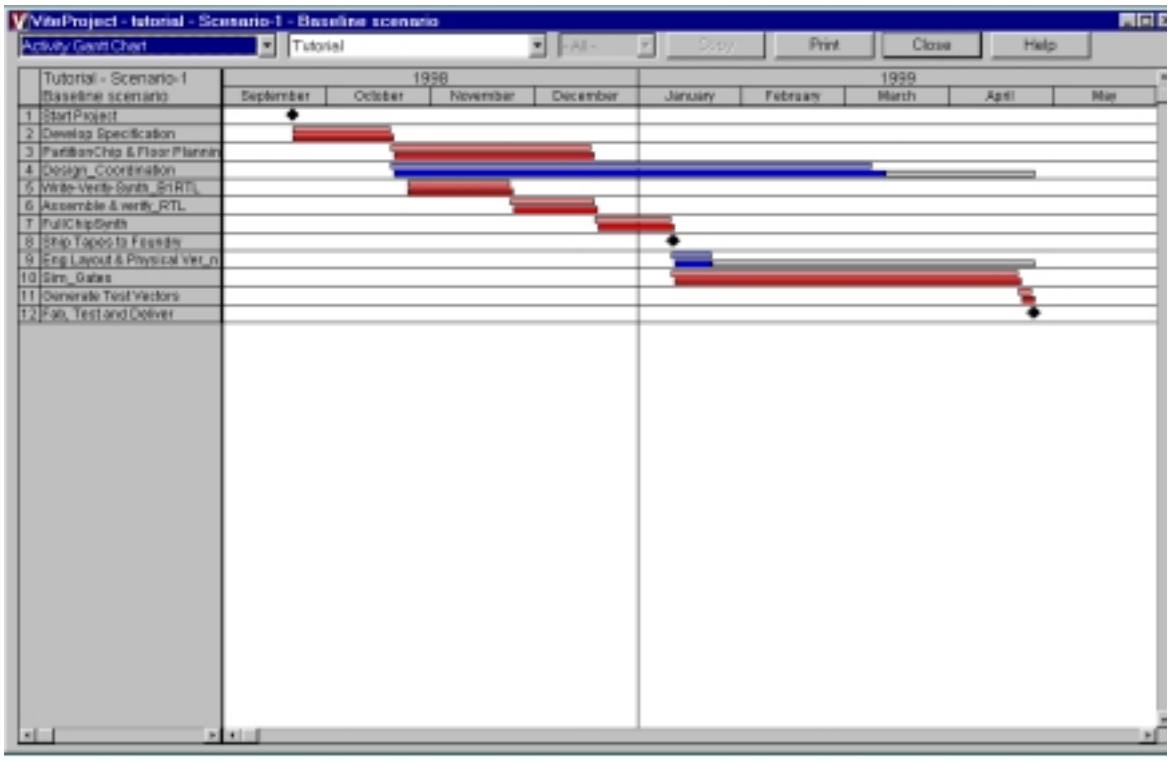

Figure 3: Vite’s Activity Gant Chart

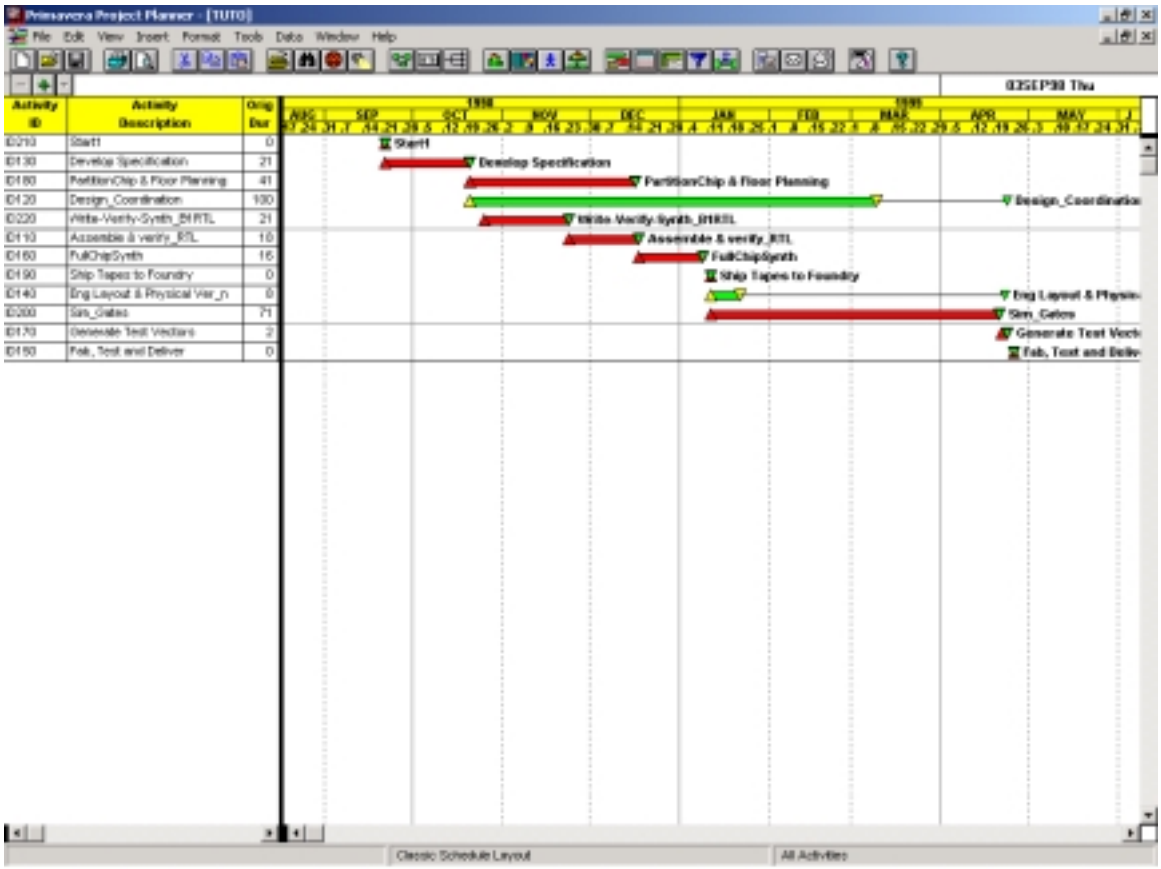

Figure 4: Schedule chart re-produced by P3

For this simple demonstration example, it appears that PSL works relative well as an interchange language for exchanging activity information. However, we do see some inconsistency between the activity chart in Vite and p3 activity schedule chart. This is primarily due to the different ontology used in Vite and P3. In Vite, there are only three terms to describe the time of an activity: CPMStartTime, CPMDuration and CPMFloat. P3, however, uses more terms to describe the time of an activity such as: EarlyStart, LateStart, EarlyFinish, LateFinish, OriginalDuration, RemainingDuration, and etc. On the other hand, there is a lot of information in Vite such as management structure and failure dependency, which are not needed in P3. Furthermore, the relationship between activities in P3 is much more complex than in Vite. Nevertheless, this example shows that it is relatively simple to transfer information from Vite to P3 using PSL. A lot of information in Vite such as management structure and failure dependency is not needed in P3.

Future Tasks

Our next step is to complete the translation of project information from P3 to Vite. In addition, we plan to investigate using PSL to exchange information from other programs to Vite. The process will involve more than activity scheduling information to include other information such as management structure and failure dependency. Using this new

integration demo, we could evaluate the potential of PSL as a process interchange language more thoroughly.

Currently PSL is focused on modeling activities' duration and ordering relationship. In order to model more complex ideas introduced in Vite, we categorize five extensions that may be needed for PSL. These five extensions include activity failure and communication dependence, management structure, activity attributes, actor attributes, and project attributes. The first two extensions would be useful standard extensions to the PSL language. The other extensions would be required to fully implement Vite in PSL, but are rather Vite specific and may therefore be primarily useful for further testing of PSL's flexibility in terms of expansion capabilities.

Finally, as noted in the proposal we plan to investigate the potential of SimQL. SimQL is a new and promising simulation access language to wrap existing simulation programs. We have done some initial investigation on SimQL and is in the process of re-compiling the software. Our plan is to use SimQL to wrap some typical software in construction industry such as P3 and Vite. The application of SimQL in conjunction with PSL could be very useful. In essence, Vite is a program that could be wrapped using SimQL, provided all the necessary input data could be gathered in advance. Using SimQL to wrap some sample programs in construction industry such as P3 and Vite, we could understand more about the applicability of SimQL in practical engineering problems. Also if we could use SimQL to wrap P3 or Vite successfully, it could further evaluate the potential of PSL as an interchange language.

Appendix A

This appendix includes PSL(KIF) file generated by the Vite to PSL(KIF) translator. It serves only for the purpose to demonstrate the possibility to translate activity information using PSL. Future tasks will further enhance the translator and other development.

```
(and (doc Tutorial "Tutorial")
        (subactivity AssembleverifyRTL Tutorial)
        (subactivity DesignCoordination Tutorial)
        (subactivity DevelopSpecification Tutorial)
        (subactivity EngLayoutPhysicalVern Tutorial)
        (subactivity FabTestandDeliver Tutorial)
        (subactivity FullChipSynth Tutorial)
        (subactivity GenerateTestVectors Tutorial)
        (subactivity PartitionChipFloorPlanning Tutorial)
        (subactivity ShipTapestoFoundry Tutorial)
        (subactivity SimGates Tutorial)
        (subactivity Start1 Tutorial)
        (subactivity WriteVerifySynthB1RTL Tutorial)
        )

(and (doc AssembleverifyRTL "Assemble & verify_RTL")
        (beginof AssembleverifyRTL 22045)
        (duration AssembleverifyRTL 8640)
        (follows AssembleverifyRTL FullChipSynth Tutorial)
        )

(and (doc DesignCoordination "Design_Coordination")
        (beginof DesignCoordination 10285)
        (duration DesignCoordination 48000)
        (follows DesignCoordination FabTestandDeliver Tutorial)
        )

(and (doc DevelopSpecification "Develop Specification")
        (beginof DevelopSpecification 0)
        (duration DevelopSpecification 10285)
        (follows DevelopSpecification DesignCoordination Tutorial)
        (follows DevelopSpecification PartitionChipFloorPlanning Tutorial)
        )

(and (doc EngLayoutPhysicalVern "Eng Layout & Physical Ver_n")
        (beginof EngLayoutPhysicalVern 38685)
        (duration EngLayoutPhysicalVern 4000)
        (follows EngLayoutPhysicalVern FabTestandDeliver Tutorial)
        )

(and (doc FabTestandDeliver "Fab, Test and Deliver")
        (beginof FabTestandDeliver 74170)
        (duration FabTestandDeliver 0)
        )
```

```
(and (doc FullChipSynth "FullChipSynth")
        (beginof FullChipSynth 30685)
        (duration FullChipSynth 8000)
        (follows FullChipSynth ShipTapestoFoundry Tutorial)
        )

(and (doc GenerateTestVectors "Generate Test Vectors")
        (beginof GenerateTestVectors 72970)
        (duration GenerateTestVectors 1200)
        (follows GenerateTestVectors FabTestandDeliver Tutorial)
        )

(and (doc PartitionChipFloorPlanning "PartitionChip & Floor Planning")
        (beginof PartitionChipFloorPlanning 10285)
        (duration PartitionChipFloorPlanning 20114)
        (follows PartitionChipFloorPlanning ShipTapestoFoundry Tutorial)
        (follows PartitionChipFloorPlanning WriteVerifySynthB1RTL Tutorial)
        )

(and (doc ShipTapestoFoundry "Ship Tapes to Foundry")
        (beginof ShipTapestoFoundry 38685)
        (duration ShipTapestoFoundry 0)
        (follows ShipTapestoFoundry EngLayoutPhysicalVern Tutorial)
        (follows ShipTapestoFoundry SimGates Tutorial)
        )

(and (doc SimGates "Sim_Gates")
        (beginof SimGates 38685)
        (duration SimGates 34285)
        (follows SimGates GenerateTestVectors Tutorial)
        )

(and (doc Start1 "Start1")
        (beginof Start1 0)
        (duration Start1 0)
        (follows Start1 DevelopSpecification Tutorial)
        )

(and (doc WriteVerifySynthB1RTL "Write-Verify-Synth_B1RTL")
        (beginof WriteVerifySynthB1RTL 11725)
        (duration WriteVerifySynthB1RTL 10320)
        (follows WriteVerifySynthB1RTL AssembleverifyRTL  Tutorial)
        )
```