Participants were:

Albert Choy (Moderator)
Ken Radlick
Rick Lowe
Ali Hariri
Brad Merrill
Douglas Turk (Recorder)

Our group participated in Sunday's full day Design Fest. [Recorder's note: since Design Fest was created as a learning exercise, these rather verbose notes are presented in a format intended to help the reader gain a greater appreciation for the discussions and interactions as we experienced them.]

The Software Stability Model for Dump Truck Scheduling document provided a thorough analysis of the problem domain -- complete with detailed diagrams, tables, use cases, a traditional class model, and sequence diagrams. The authors had addressed several design issues with a stability model, state diagrams, collaborations and even interfaces for each type. After reviewing the document, our initial reaction was that our work had already been done.

We reviewed the following list of suggested deliverables and tried to determine which parts were doable in the allotted time.

- Assumptions
- Class Model – Scheduling model
- Class interactions
- Interfaces
- Lessons learned
- Conceptual model (optional)
- Persistent data model (optional) – Defined as interfaces
- Error handling (optional)
- Class detail (optional)

We then chose to focus our efforts on the scheduling model, class interactions, interfaces, and error handling.

After a second reading of the supplied documentation, Albert suggested we describe each of the objects to ensure they make sense. Ken felt that the document showed how the system works and simply wanted a generalized scheduling system. All agreed to focus on the scheduling component (or scheduling engine) of the system. The documentation provided no clear-cut direction on how or when schedules should be generated or updated. One option would have been to calculate and produce 20 years (the time limit on exploiting the ore) worth of schedules up front. The comment was made that even if the schedules were produced on a daily basis, they would be obsolete by lunch break.

Ken proposed a model of the system that would schedule, monitor, and adjust the schedules dynamically.

We dug in by starting to list assumptions that will be highlighted throughout this document as they are introduced.

> *Assumption – We would model a generic scheduling solution.*

> *Assumption – The solution would support multiple models.*

Ali pointed out that the Stability model (supplied) already provides a more generic solution. After the group went off on several tangents, Albert suggested that we schedule our time in an effort to pace ourselves. Working backwards, we decided to allocate our final half-hour to "Lessons Learned" and the half-hour before that should be used to clean up our presentation.

Next, our attention turned to unanticipated negative scheduling events and how often the schedule should be run. We explored ideal vs. real world scheduling.

> *Assumption – The system should model both ideal and real world situations.*

Ken posed the question of what goes into a schedule and how do we determine one. Rick then differentiated the process of scheduling for queues vs. scheduling routes. This led to the introduction of the constraint of scheduling for routes where one happens to be and was followed with a lengthy discussion of how changes (ie. number of trucks) should be handled.

> *Assumption – Cannot change the number of trucks in a defined period where the default period is one blasting cycle and the minimum period is 13 minutes (the minimum route duration).*

Ken took the lead, mapping out his the first conceptual model of the scheduling engine (shown below).

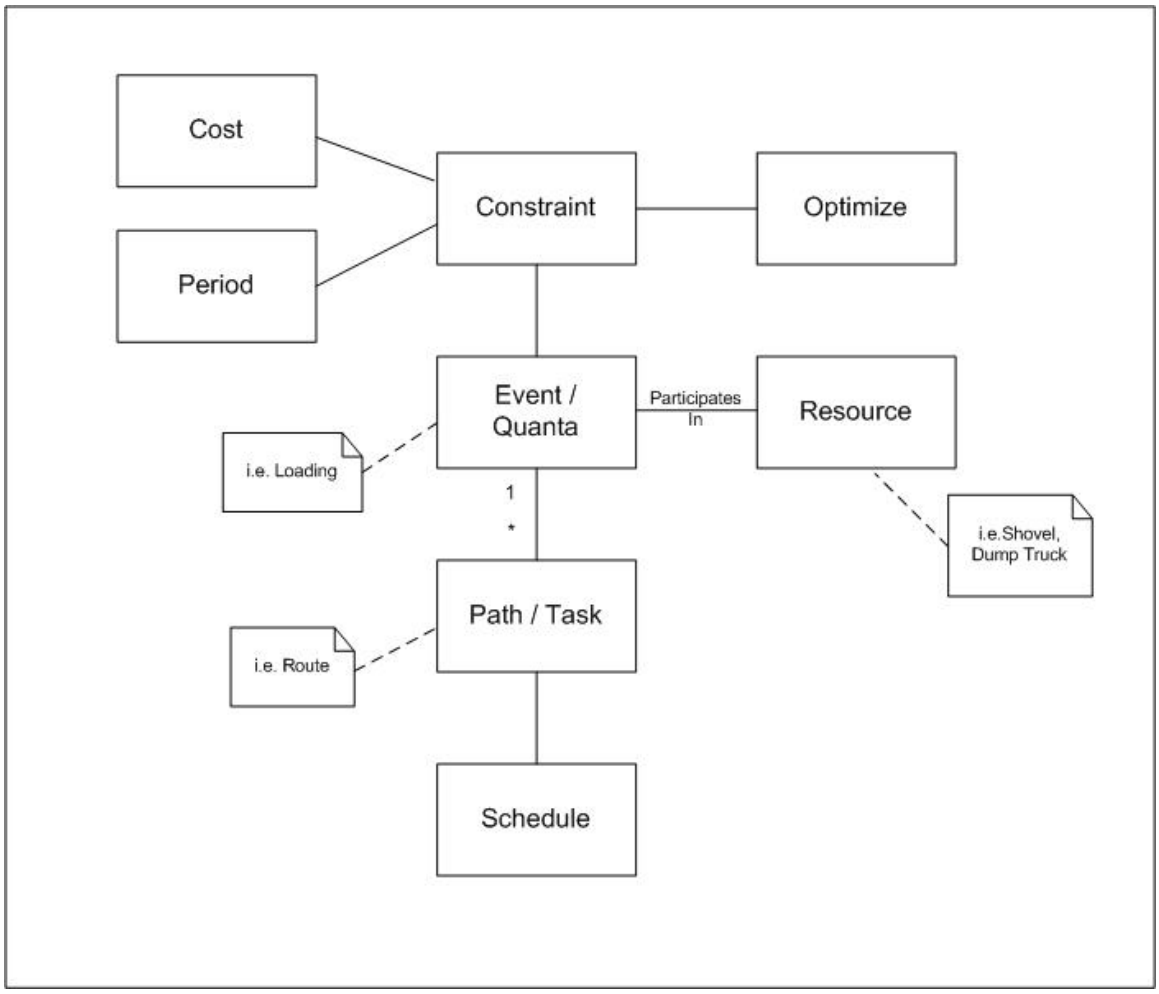

Figure 1 – Conceptual model

Assumption – No queuing at the waste dump.

Assumption – No queuing at the Mineral processing facility.

Brad proposed an alternate model using a rule base to generate a state machine. The diagram below depicts the status of the trucks along the route or at each location.

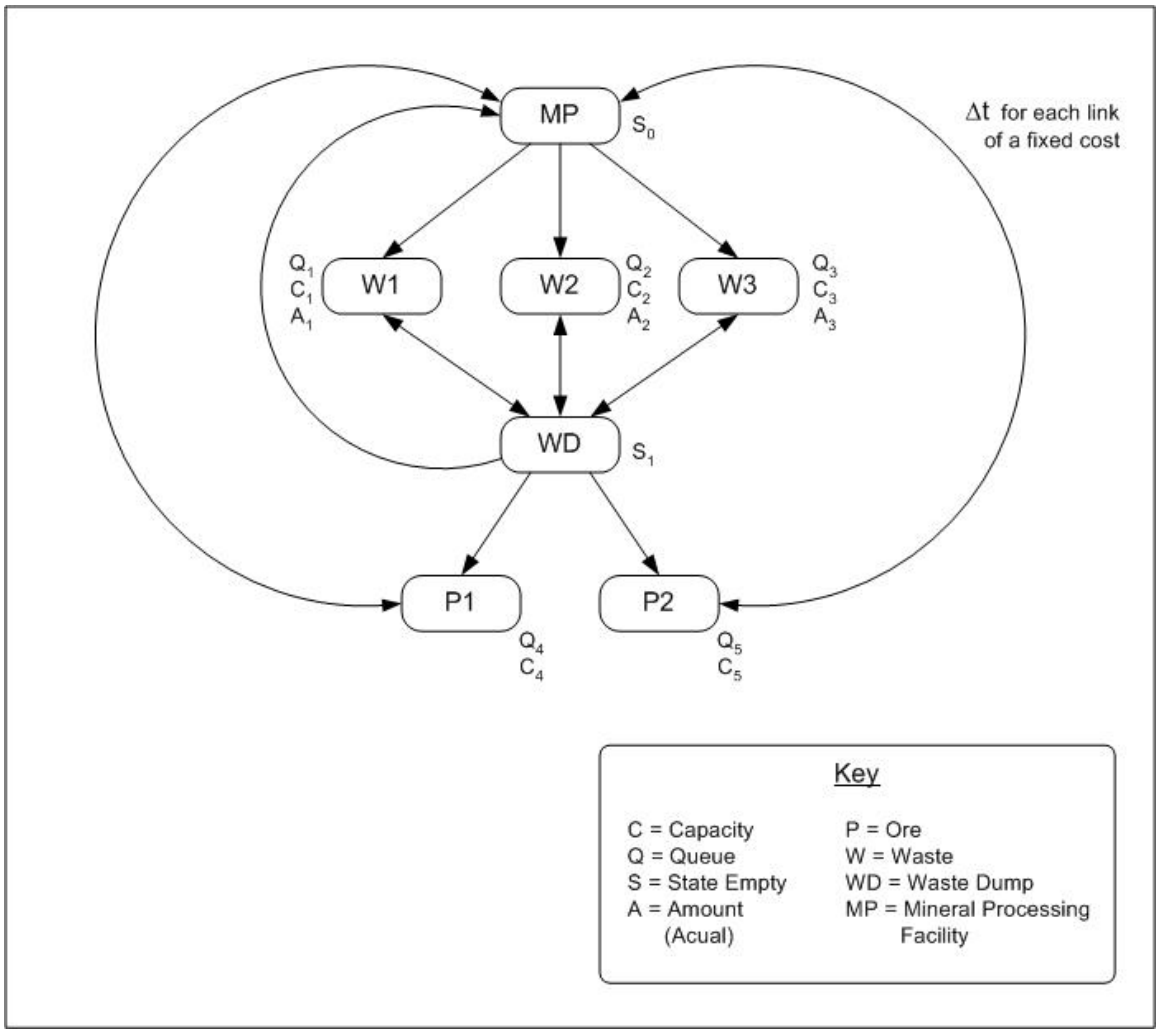

Figure 2 – Truck status

Assumption – Working day would be 24 hours.

We were reminded that one of our goals was to determine the optimal number of trucks. Rick did some baseline estimates and determined that on occasion, workdays would be shortened due to quotas.

Assumption – Obey the quotas for each truck and shovel.

Mid-afternoon, shortly after Albert reminded us of the time and suggested we move on to a class model, our domain expert (problem co-author Majid Nabavi) arrived. As in real-world development, early communications with the user is paramount. We were reminded of this as several of our early assumptions were corrected or clarified. We learned the drivers and shovel operators only worked a single shift each day; the shift was usually 8-12 hours long; one shift was reserved for maintenance; and the trucks (fueled prior to the workday) held enough fuel to last the entire day.

Majid suggested we keep our solution flexible and reminded us that if more than one Mineral Processing Facility existed, each facility handled a different type of ore. During a status review for Majid's benefit, we asked if he had a feel for the maximum times for each truck route. Perplexed at first, he realized our error and pointed out the times in the tables were an average. The (min) designation in truck route table stood for minutes rather than minimum as we had assumed. Again, we benefited from the value of having a domain expert readily available.

At the late hour, we re-assessed our task. Brad suggested we deliver a bunch of classes, while Ken favored a Use Case to guide deliverables. (Note: we also realized that Majid's copy of the Software Stability Model for Dump Truck Scheduling contained a Use Case diagram that was absent from our working copies.) We decided our solution should let the user (i) set up rules; (ii) set up states; and (iii) run the scheduler. We would design an "in-house tool to create/develop scheduling elements used to drive a scheduling engine." We needed to develop a class model to "support this thing" (the generalized scheduling engine). This is represented in the model below:

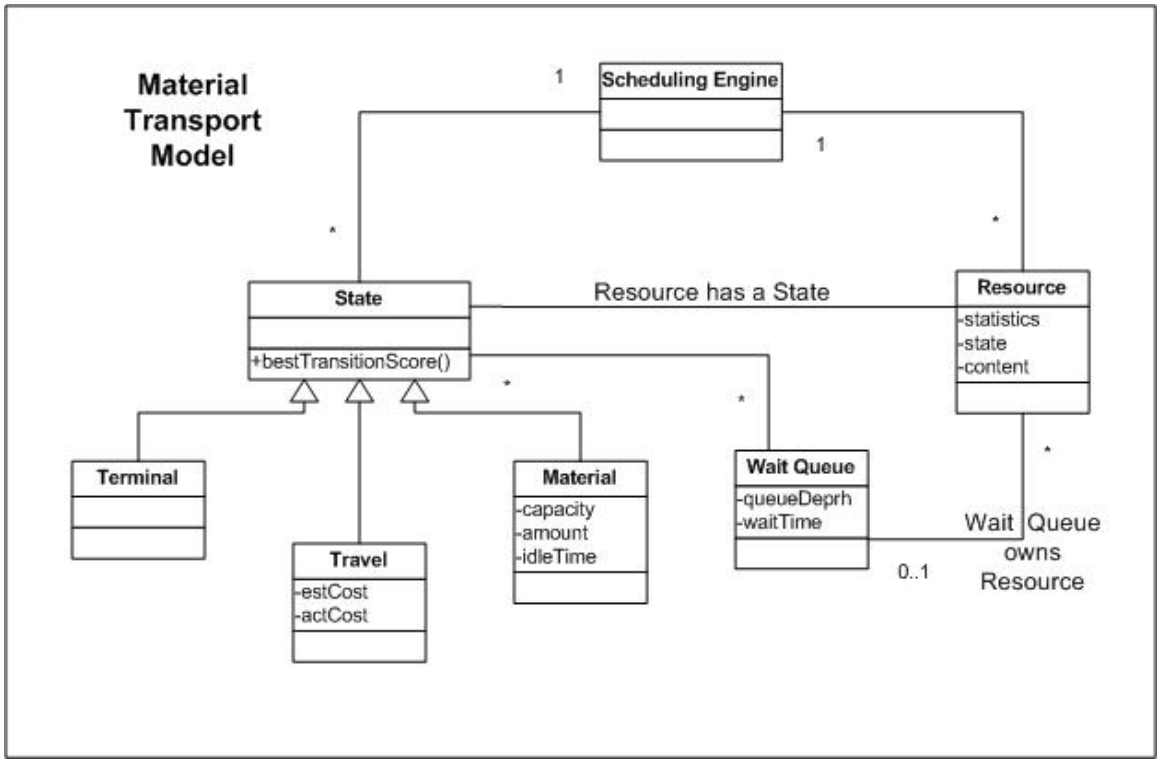

Figure 3 – Class Model

We did not get to design a tool to generate the rules. Ken noted that the supporting document gave the impression that immediate feedback was a requirement. We later learned from Majid that he only expected the system to generate daily schedules at the

beginning of each shift. We shared our early concerns that in a real world situation, such schedules would be unusable after a few slippages (probably by lunch at the latest).

At the end of the day, the two key points that this group learned were:

- Even really big, well-documented specifications can lead you astray.

- We had an expectation disconnect – The customer had expected single schedules per day rather than an adaptive system.

On another note: As the day wore on and our creative juices started blending with travel fatigue, the group seriously considered re-modeling the system in SML (Stupefied Modeling Language). The SML extends other popular OO methods with the following relationships:

"Has a"
"Should have"
"Never should have"
"Might have"
"Wish we had"