The Glass Line

Specification

[Note: This is from the original paper on the glass line. I have added some comments in brackets about how csci201-csci200 is going to do things. For example, [Prof. W: blah, blah.] is such a comment.]

In the kitting cell an agent-oriented approach led to a successful cell control implementation. The glass line was already operational when a similar analysis was sought. An offline model was built with three goals:

1. to enhance throughput by improving the conveyance algorithms;
2. to eventually supply a full control program for the line; and
3. to show that part tracking is feasible and could reduce many of the manual responsibilities. [Prof. W: We are going to assume and do this.]

The glass line, shown schematically in Figure 9.4, turns raw glass panels into windows and windshields for automotive vehicles. Glass-finishing processes are linked together by a segmented conveyor system. Some of the processing is done automatically right on the conveyor, but most happens as operators move the glass between the conveyor and their work centers.

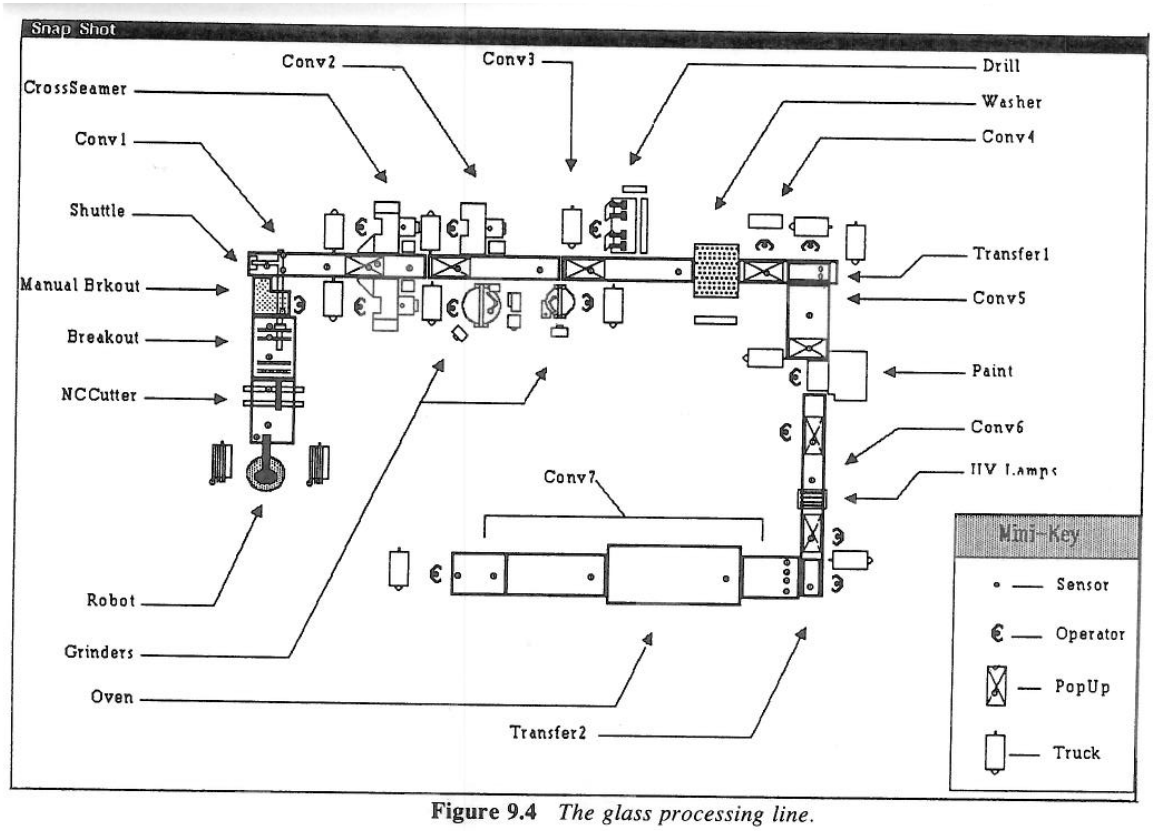

Figure 9.4 *The glass processing line.*

The line is in a laboratory developing glass manufacturing technology. New glass processes are continually being tried and evaluated. The line is continually being reconfigured to accommodate them. However, the line is also a working prototype with

daily production quotas and an intention to be a model for all the company's glass lines. The desire to continually improve throughput is strong. Though the glass-processing is the primary focus, the line management is almost as important. It is here that the agent-oriented analysis was expected to yield its main results.

The glass line's basic cycle has four parts:

1. A robot picks up a glass panel from either of its input bins and puts it on the first conveyor (it's called the NC Cut Conveyor; that name is not on the diagram). From there the glass moves into position where an NC Cutter machine cuts a template into it. The specific NC part-program is dialed in manually [Prof. W: in our system a bar-code reader will read part and dial in the appropriate program.]. The operations are keyed by sensors on the conveyor. After cutting, the glass is heat-treated (not named in the diagram, but shown right after the NC Cutter), and the template is broken out of the glass. The shuttle automatically moves the template onto conveyor 1. If the breakout was unsuccessful, an operator can manually finish the process.
2. At the top, four conveyors move the template through various phases of edge-treatment. Not all operations are necessary on all templates. Each of those conveyors has a pop-up table (the boxed X) to raise the glass up. The pop-up will, by default, raise the glass unless the operator signals otherwise by pushing a bypass button. The operator then slides the glass off the pop-up into his work-area for specialized processing. When the operator is done, he slides the glass back onto a raised pop-up (if the pop-up is not raised he can push a button to raise it directly) and pushes a button to lower it. The conveyor can be moving while the pop-up is raised. [Prof. W: we will have most of this under part-type control, i.e., the system will know what type is approaching and whether the process (and its pop-up) should be activated.]
3. On the right side, conveyors 5, 6, and 7 move the template where it is manually painted and then treated with ultra-violet light. The same pop-up mechanism is used here when manual processing is required.
4. Finally, on the bottom the paint is baked on in a special oven. The glass template is manually removed al the end.

The line is controlled by a factory-hardened computer that is wired to the devices and contains all the logic--agent and otherwise.

Each conveyor has combinations of the following controls:

- start/stop conveyor,
- raise/lower pop-up, which can be enacted by operator push-buttons,
- pop-up bypass switch, enacted by an operator push-button.

the bypass switch means the operator doesn't want the glass to be raised

There are the following glass detection sensors:

- for each pop-up, there is one sensor preceding it and one sensor on it.
- each conveyor has an entry and exit sensor,
- if a pop-up is at the beginning of a conveyor, then its pop-up sensor doubles as the entry sensor,

- if a pop-up is at the end of the conveyor, then its pop-up sensor doubles as the exit sensor; [Prof W. If a pop-up ends one conveyor, and another pop-up starts the next conveyor, then the ending pop-up acts both as an exit sensor and a pre-pop-up sensor. Right?]

Of course, the placement of sensors and pop-ups varies from conveyor to conveyor.

For our purposes, the various off-conveyor glass processing machines have a start-process(part-type) control. The machine signals us when it is done, so that the operator/robot can move it back to the pop-up.

System analysis and design rationale

The main goal of the system, to keep glass flowing efficiently around the line, is tempered by the danger of glass panels crashing into one another or into raised pop-ups. This kind of continuous flow system is difficult to manage using standard methods. Like the kitting cell with its many devices and hundreds of potential interactions, the glass line has many conveyors to coordinate with information coming from dozens of sensors. The continual reconfiguration caused by the glass-processing experiments only added insecurity to the factory floor management.

old way that was easy to code but not efficient

The original technical response was to make conservative control decisions. For example, on conveyor 1 only one part was allowed to approach its pop-up, even though there was room for several; a new part could be loaded onto the conveyor only as a part was leaving the pop-up.

In that original implementation, a part could approach only if the pop-up was down. Otherwise, the conveyor would stop and wait for the pop-up to lower. When it had, the conveyor would move the part off the pop-up as the next part approached. When the trailing edge of the processed part triggered the pop-up sensor, a timer would start. Its expiration signaled that the new part had arrived and the pop-up could be raised.

This solution required a pre-designed flow of parts on the conveyor. The approach times were hard coded into the algorithm and the operators had to know the loading rules. Though simple, it was inefficient. By ignoring most of the available sensor information, the implementation was unreliable as well. Operator and timing errors were undetectable. Unfortunately, the coding simplicity won out over robustness.

The conceptual difficulty of managing so many connecting conveyors, each with slightly different layouts and slightly different entry/exit requirements was daunting. The inherent volatility of the line's overall configuration just added to the problem. Yet it is precisely this type of problem that yields to agent-oriented methods.

Part recipes

Interestingly, in the original implementation most of the line decisions depend on human recognition of what was going on. Nothing in the line's automation knew what kind of parts were flowing through. An operator had to dial in the NC part program number for

the glass parts being loaded by the robot. Each operator at a glass-processing station had to declare using the bypass buttons, whether or not he wanted the part that was approaching the pop-up. The parts were moving around blindly, totally dependent on the operators to manage the interface between stations. To simplify things, templates were done in large batches; different types of templates were not mixed together on the line.

In our agent-oriented model, where parts are essentially handed off between agents, part type information should be handed off as well. The agents could make decisions on their own, making it easier to mix parts on the conveyor. On entry to the system a bar code reader could tell the system what kind of part was to be produced from that piece of raw glass..

The system has part recipes that declares what must be done to each type of part. As parts move from conveyor zone to zone, the part type information is passed between the agents along with updates about the state of the part. This scheme makes more automation possible. For example, the NCCutter part program number can be downloaded directly by the agent instead of it being dialed in by an operator. Similarly, pop-up agents can tell if the part is to be processed at its station; the operator need not push the bypass buttons to give the system that information.

~~Timestamp information was stored as well. These statistics were important for the process engineers to study, but they were also used by other analysis packages. The statistics were fed into a time-compression discrete event simulator (not to be confused with the agent simulations discussed above). Suddenly, it was possible to study the different patterns relating to the part mix, changeovers, start-ups and shut-down conditions, etc. The new data was fed into a time-compression discrete-event simulator in order to generate throughput statistics over long periods of time~~. [Prof. W: We won't be worrying about this.]

Conclusion

The kitting cell and the glass line benefitted from a prototyping mentality made possible by agent-oriented techniques. Both had processing models that were fairly easy to state and understand, yet were difficult to implement as control solutions using classical techniques.

The basic problem is two-fold: (a) systems are hard to understand until they are implemented and (b) once systems are implemented, they are hard to fix or improve. The former implies a need for rapid prototypes, the latter a need for an implementation that is flexible, understandable, and changeable. These case studies showed how both problems were addressed using agent-oriented techniques.

The kitting cell showed how the prototype led to understanding about the device interactions and suggested improvements in the cell's mechanisms. The prototype evolved with those changes; without the encapsulation offered by the agent methodology,

the software could never have kept up with the improvements. The glass line model did exactly the same thing, but here the line was already implemented and operating poorly.

Prototyping and understanding are brothers under the skin. The kind of models we built here created a foundation for understanding and improvement. This kind of analysis has typically been done using only time-compression discrete-event simulations. However, those models are never elaborate enough. Things turn up during implementations that were missed during the simulation.

The agent implementations are concrete models intended to control the target system. The systems can be prototyped quickly, analyzed, and improved. Then, if the implementation is used for control, integration is straightforward, as it was in the kitting cell. Or, if the model is used to improve a process, some of the algorithms in the prototypes can be reimplemented in the actual system, as they were in the glass line. Agent-oriented methodologies have a natural place in manufacturing; successful case studies will demystify their use.

References

ANSI (1989) Manufacturing Message Specification (MMS). EIA Standard 511, American National Standards Institute.

General Motors (1984) MAP Specification 2,J. Appendix 6 (MMFS).

Savoir (1988) FLEXIS product overview. Hayward, CA.

Wilczynski, D. (1988) A common device control architecture - The Savoir Agent. Proceedings of Autofact '88. November, Chicago, IL.

Wilczynski, D. (1990) Creating a control system application - From concept through production. Proceedings 0f IPC '90, April, Society of Manufacturing Engineers. Dearborn, MI.