Using the Process Flow Diagram Object to Communicate Information

Frank DiIorio
ASG, Inc.

Background

- Large data volumes - problematic and in vogue
- Options for analysis are increasing as costs are driven down
- "Off the shelf" products include INSIGHT, LAB, JMP. And AF can drive them.

This Paper

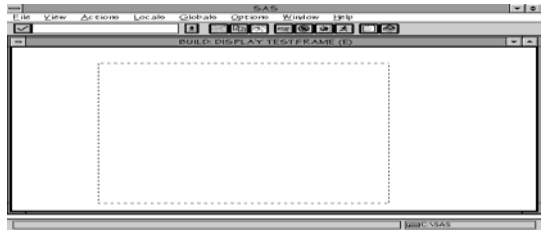
- Describes a customized, graphical solution to an engineering problem
- Presents the problem ...
- discusses alternative solutions, ...
- and outlines the solution

Initial FRAME Screen

Make a New Region

Select PFD Object

Size the Object



Set Object Attributes

PFD, Ready to Draw

The Setting

- Mitsubishi Semiconductor America, Inc (MSAI). Wafer fab facility in Durham NC
- "Near real time" data capture and analysis
- Complex, and extensible
- Request from Photo Etching: completions by unit by time, in graphical format
- Identify both completions and idle time (gaps between completions)

The Application

- Display completions and gaps on the same screen
- Point and click access to data
- Ideally:
  - hard-copy reports from underlying data
  - screen prints
  - user-selectable date
  - user-selectable gap size

The Application (cont.)

- Other design issues
  - Constant screen size
  - Display period always 24 hours, always 12 Photo Units (don't let them grow !)
  - Multiple days displayed in a single invocation of the program
  - Click on a completion to get info
- Engineers gave us carte blanche re: implementation and "look and feel"

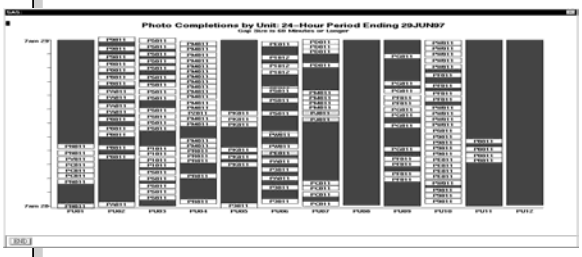
Solution Strategy

- First, "rigged" GHCART (since automatically hot-spotted)
- Then, ANNOTATE and DSGI (hot-spotting not readily implemented. Old, nasty technology)
- Process Flow Diagram object (new to AF in Version 6.11). Nodes are automatically hotspotted. Real cool technology. Won't die in Ver 7 (like ANNOTATE).

Introduction to the PFD Object

- Basically, a crude flow-charting tool
- An interactive (or data-driven!) drawing environment
- All widgets are named, and can be manipulated by SCL (_get_info_ and _set_info_ methods)
- Think of the Photo Unit requirement as a drawing, not a flow chart, and PFD makes a lot of sense. Each completion or gap is a node (rectangle)

Sample Screen Built with PFD



Using the PFD

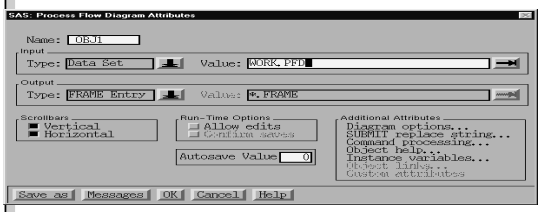
- Can be done manually (drag objects into drawing area). Inappropriate for current app.
- Build diagram via input SAS dataset
- Each object has numerous characteristics. These include:
  - type (arrow, line, rectangle)
  - position
  - color
  - associated text
  - a _TYPE_ variable value
- "deconstruct" a representative diagram's dataset to understand the structure

Building the PFD

Appearance of the initial screen

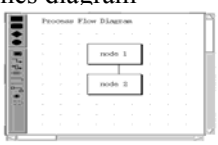
Object Attribute screen

Right click from anywhere on the screen



Building the App

- Create a bare-bones diagram,
- save it to an output SAS dataset,
- deconstruct to understand dataset structure (meaning of _TYPE_, font values, etc.)
- Bare-bones diagram



Building the App (cont.)

- The dataset used to create the previous page's diagram

Items of Interest

- Must have a Title (_TYPE_ = 22)
- _LABEL_ uniquely identifies a node
- x1-y1, x2-y2 are node coordinates
- _TEXT1_ contains optional text displayed with the object
- _ICOLOR_ is fill color of object
- _FONT_ is _TEXT1_'s font
- Once these and some other features are understood, we can build the app.

Building the App.

- Recall ... layout was fixed WRT # of units (along X axis) and time (24 hours along the Y axis)
- Thus ... create a "template" for fixed or slowly changing elements (title, axis lines and labels). This is a permanent SAS dataset. Edit with FSVIEW, then preview. Iterate until appearance ok.
- Then, create event/gap dataset from the production database. This is a WORK (temporary) dataset.

Building the App. (cont.)

- Event/gap dataset issues:
  - Representing an event (rectangle, not a line, to allow addition of text and to enable easy pointing)
  - Node color (white, yellow, red) Green when selected.
  - Identifying gaps
  - Exert control when building the dataset (_label_ values DATAxxx and GAPxxx enable direct access when displaying info for a selected node)
  - Combine template and event datasets and you're good to go

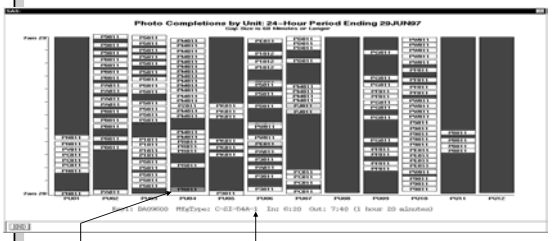
Running the App.

- Combine the template and event/gap datasets and you're good to go
- Invoke via a driver screen (separate FRAME since can't _REPOPULATE_ the object to display a new day's data)
- Driver screen. User selects a date, clicks on "Create Diagram" If data exists, next screen is displayed.

Running the App. (cont.)

- User clicks on an event or gap
- ... _GET_INFO_ method retrieves _LABEL_ value (DATAxxx or GAPxxx)
- ... this value is used in a WHERE clause in the production data subset to retrieve the observation
- ... then FETCH the data to make it available to the app
- ... construct and display a message at the bottom of the screen

Running the App. (cont.)



Message area (part of the PFD)

User clicked on this completion

Aesthetic Considerations

- Stack by Photo Unit to give quick view of completion distribution. Color code "reworks" (yellow, rather than white) and gaps (in red)
- Ideally, node width should reflect time required for completion.
- "Lean" display (underlying grid distracts visually and interferes with node selection)
- Recognize text-based reports are useful supplements (all numbers displayed simultaneously)

Enhancements

- Node width = f(time in unit)
- Printing (remarkably problematic, but possible). Print command prints only PFD widgets, **not** the entire screen!
- Deal better with overlapping nodes (sometimes won't [de]select)
- Display speed is quick, and **not** an issue(!)

In closing ...

- This is a popular, visually appealing tool used by wafer fab technicians and engineers (functional, yet entirely mouse-driven)
- Note importance of digging into documentation to uncover obscure- yet-legitimate aspects of objects (and procedures). Input datasets to create diagrams is an example of this.
- Get design ideas from people like Edward Tufte!
- Thanks SI tech support, esp'ly Art Alexander!