Lessons Learned From the MAGIC Program’s GUI Implementation

Capt Steve Lindsay
Satellite Autonomy Program Lead
Air Force Research Laboratory
lindsays@plk.af.mil
505/853-4140 (DSN 263)
Overview

• Background on MAGIC
• Lessons Learned
  – GUI implementation - language choice
  – Decoupling the GUI from its sibling components
  – Inter-process communication
  – Importance of iterative development and rapid prototyping
• Conclusions
Background on MAGIC

• Experimental satellite telemetry analysis system developed in 1994
  – Expert System (ExSys, CLIPS)
  – GUI (MS Visual C++, Tcl/TK)
  – Database (MS SQL Server)
  – Central Controller (Borland C++)
  – Front End Decommutator
• All components varying degrees of COTS
• Open architecture
• Total system cost: $300K
Lesson #1

- GUI implementation language **matters**
- MS Visual C++
  - Steep learning curve
  - Executable contains a great deal of overhead
- Tcl/Tk
  - Minimal learning curve
  - Less than optimal run-time performance
Lesson #1 Results

• Examine all the issues when choosing the GUI language implementation
  – Cost, performance, in-house expertise, turnover
• Perform a thorough survey to see what’s available
• Don’t take the decision lightly
Lesson #2

• Decouple the GUI from its sibling components
• Ours was coined the “Dumb GUI”
  – You tell it what to display and it displays it
  – Allows the user to initiate actions, but passes those actions to a different component
  – Its only intelligence was based on its internal world
    • Memory management
    • Caching commands
Lesson #2 Results

- Decoupled components lead to greater flexibility
  - We had the luxury to choose between GUI COTS products
  - Encourages specialized areas of expertise
- Beware: you can still create a coupling to the underlying implementation
  - GUI programmer requested his actions be in Tcl/TK syntax
Lesson #3

- Efficiently communicate data between the processes of an open architecture
  - MAGIC first used dynamic data exchange (DDE)
    - The OS kept striking down the connections
    - Debugging required all applications to execute
  - We switched to file-based inter-process communication (IPC)
    - Simplified debugging and stand-alone testing
    - Allowed for automated recovery when the OS gave us problems
    - “.INI” files simplified configuration management
Lesson #3 Results

• Based on your circumstance prepared to change your IPC mode at any time
  – We were counting on OS-supplied IPC
    • The OS consistently interfered
    • They ultimately decided to stop supporting it

• When adding functionality, file-based IPC complexity explodes

• Today’s commercial middleware choices look promising
  – Depends on the size of your application
Lesson #4 and Results

• Use iterative development and rapid prototyping
  – Monthly, we showed the users an updated GUI with minimal underlying functionality
  – Advantages were dramatic
    • They received a useful product
    • We incorporated requirements that had changed since development began
    • They felt a high degree of ownership in the software

• Results: why do anything else?
Conclusions

• Carefully choose the GUI implementation language, but give yourself the flexibility to change later

• Keep the GUI dumb

• Use commercial IPC if the project size warrants it, and “tried and true” IPC otherwise

• Incorporate iterative development