GPS System Simulation Methodology

Thomas F. Ewing
Argonne National Laboratory
Engineering Physics Division

Talk Outline

Background
GPS Methodology Overview
Graphical User Interface
Current models
Application to Space Nuclear Power/Propulsion
Interfacing requirements
History

• SALT (system analysis language translator) - Early 80's
  - PL/I code for IBM mainframes
  - Moved to multiple platforms and languages (C, C++)
  - Batch oriented - translate, compile, run
  - Used model and property libraries
  - Optimizations and system analysis

Applied to
- Open-cycle and liquid-metal MHD systems
- Fuel cells
- Ocean thermal energy conversion
- Municipal solid waste processing
- Fusion
- Breeder reactors
- Geothermal and solar energy systems

Next Generation Implementation - GPS

• Designed for modern workstation environments
• Developed in C++, moved to C for greater portability
• Steady-state & dynamic model libraries concept of SALT, but accessed as class objects
• Complete, extensible, object-oriented control language with numerous procedures for optimizations, equations solving, system constraints, parametric analysis
• Language interpreted, but uses compiled, fully optimized models and math procedures ==>
  - Fast prototyping cycles
  - On-the-fly creation of interaction with simulations
  - Simulation systems can be interrupted, queried and changed, then resumed
Simulation/Modeling Approach

PROCEDURES
- Equation Solvers
- Component Models
- OPTIMIZERS
- Plotters
- Properties
- LIBRARIES

LIBRARIES
- Handling arbitrary configurations of components and interconnecting information flows
- Uses object technology: component models and procedures attached to client objects (exploits software re-use)

ARGONNE

GPS Operators
- 86 built-in operators
- I/O functions (fopen, printf, scanf, sprintf)
- Math functions (atan2, pow, exp, max, ln, log10)
- Numerical procedures (vary, cons, icons, mini, diff)
- Looping and flow control
  cond [...] if
  cond [...] [...] ifelse
  start inc bound [...] for
  count [...] repeat
  [...] loop
  [cond] [...] while
**Miscellaneous Operators**

- Allocate new model class instance - `cdef`  
  `/pump1 { pump: /param1 12.0 /param2 0.495 } cdef`  
- Set a debug level (0 thru 5) - `debug`  
- Run gps simulation from a input file - `run`  
  "input.fil" run  
- Interrupt simulation to permit queries/interactions  
  `sintrp` (followed by `resume` to continue)
GPSTool - Graphical User Interface

Parameter panels provide access to system variables and model default values.

- Supports drop-and-drop loads of input files or templates.
- Graphics options include generation of plots and system diagram. System diagram also permits mouse interactions to select components for insertion, deletion, rearrangement, parameter changes, etc.

GPSTool panel provides point-and-click creation and control of simulations. Using object technology, GPSTool assembles simulations from standard software parts: model libraries, databases, and procedure libraries.

Control language governing simulation and generated by GPSTool appears here: may be edited or manually input by power users.

Phillips Lab Simulation Strategy

Network Comm
GUI
Utilities
Executive
Model Interface

Data
Models
Application Drivers

Standalone Apps

System Level
Lower Fidelity
Component Level
Higher Fidelity

NP-TIM-92
Advantages as Integrating Environment

- Consistent user interface to models
- Diverse models can be combined for use in arbitrarily complex systems
- Suite of gps system analysis capabilities (sweeps, optimizations) and numerical methods/properties available to models
- Interface definitions external to models =>
  - can adapt models developed independent of gps
  - can use proprietary models available only as object code
  - models used with gps can still be run in native mode
Interfacing Considerations

- Component models can be Fortran, C, or other Sun languages which generate linkable object code
- Standalone codes must be structured as subroutines with argument list of variables/parameters that must be known to GPS system
- Use of Fortran common blocks prevents (presently) having multiple instances of that model in a system
- Because models may be cycled through numerous convergence iterations with perturbed input flows
  Models must be true functions of their inputs
  Models must be reasonably robust
  I/O routines should be moved outside computation routines

Converting a standalone code

- Two step process:
  Convert code to one or more subroutines
  Create a interface definition file (IDF)
- GPS uses IDEF to generate small C code to handle interfaces
- Model can still be run independently of gps (standalone) by writing a main program to call subroutine
Interface Specification File Format

Interface specifications external to models
- User-prepared ASCII file used by GPS preprocessor to generate C stub code to handle gps interfacing
  - Model name
  - Variable types and initial values (arguments + gps I/O)
  - Entry procedures (name, arguments if Fortran routine, in and out flow variables)
  - Print variables (used as default gps output)

Example Conversion

Fortran Standalone code - TDS
- 8400 lines of Fortran code (includes TECMDL)
- Required 32 line interface definition file
- Conversion completed in < 2 hrs.
- Same model now runs standalone (called from main) or in gps environment
- Both open (once through) and closed systems have been run in gps
- Have successfully run problems with 250,000 nonlinear constraints in nested loops