GPS System Simulation Methodology

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

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 `intrp` (followed by `resume` to continue)

GPS Steady-State Power System Models

<table>
<thead>
<tr>
<th>Basic component models</th>
<th>Basic thermionic models</th>
</tr>
</thead>
<tbody>
<tr>
<td>gas - gas flow initiator</td>
<td>reac - reactor model</td>
</tr>
<tr>
<td>sp - gas flow splitter</td>
<td>ti - thermionic converter</td>
</tr>
<tr>
<td>mx - gas flow mixer</td>
<td>rad - thermal radiator</td>
</tr>
<tr>
<td>ht - gas flow heater/cooler</td>
<td>sp - power flow splitter</td>
</tr>
<tr>
<td>hx - gas flow heat exchanger</td>
<td>res - electrical resistor</td>
</tr>
<tr>
<td>cp - compressor</td>
<td>bc - boost converter</td>
</tr>
<tr>
<td>gl - gas turbine</td>
<td>bus - electrical bus</td>
</tr>
<tr>
<td>pump - pump</td>
<td>mass - mass calculations</td>
</tr>
<tr>
<td>df - diffuser</td>
<td></td>
</tr>
<tr>
<td>nz - nozzle</td>
<td></td>
</tr>
<tr>
<td>power - calculate system powers</td>
<td>More sophisticated models</td>
</tr>
<tr>
<td></td>
<td>therm - thermal flow initiator</td>
</tr>
<tr>
<td></td>
<td>hprad - heat pipe radiator</td>
</tr>
<tr>
<td></td>
<td>tds - thermionic diode subsystem</td>
</tr>
<tr>
<td></td>
<td>shx - simple, multinode heat exchanger</td>
</tr>
<tr>
<td></td>
<td>nhx - multinode, general purpose HT model</td>
</tr>
</tbody>
</table>
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 an interface definition file (IDF).
- GPS uses IDF 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