Web Interface to Materials Simulations

WIGLAF

“Web Interface Generator and Legacy Application Façade” Portal

Development Team
- Akos J. Czikmantory (JPL - Wiglaf)
- Hook Hua (JPL - Wiglaf)
- Seungwon Lee (JPL - EZTB)
- James Lu (JPL - Wiglaf)
- Gian Franco Sacco (JPL - EZTB)
- Paul vonAlmen (JPL - EZTB)
- Zhen “Joseph” Xu (JPL/Caltech - Wiglaf)

WIGLAF Concept
- Tom Cwick (JPL)
- Gerhard Klimeck (Purdue University/JPL)

Funding
- DARPA-PROM
- JPL SRRF
- JPL ESTO-CT

Applied Cluster Computing Technologies Group
Earth Science Data Systems Section
NASA Jet Propulsion Laboratory
Agenda

- Background
- Motivation
- Approach
- Application Integration
- Example Applications
- Conclude with Demo
The Need for Computational Resources

- **Scientific Data Processing**
  - Large datasets
  - Fast computational results

- **Physics-based Models**
  - High-fidelity simulations
  - Long runtimes
  - Large memory/disk usage

- **Design Environments**
  - Optimization
  - Genetic Algorithms
Commodity Clusters

- Beowulf cluster
  - High-performance parallel computing
- Commodity PCs
- Commodity networking infrastructure
- Open source software
- Extensible as needed
- Cost effective
Motivation for WIGLAF

- Legacy cluster applications are difficult to use
  - Consoled-based
  - Knowledge of specific environment needed
    - Command line arguments
    - Dependencies
  - Disparate input formats
  - Output management
- Cluster knowledge needed
  - Requires management of resources on nodes
  - Interaction with MPI, MPICH, LAM, PVM, Myrinet, etc.
Solution

- Create web-enabled middleware framework to facilitate ease-of-use
- **Web Interface Generator and Legacy Application Façade (WIGLAF)**
  - Web portal application
  - Network enabled Java applets

**Benefit**

- Application developers
  - Need not create custom GUIs
  - Need not modify existing applications
- Users
  - Need not know about execution environment
  - Need not install new software
Requirements

- Generates GUI front ends
  - Dynamic
- Multi-user support
  - Security model
- Multi-application support
  - Existing applications
  - General, not too specific
- Manages application inputs
  - XML representation
  - Creation
  - Editing
  - Validation
  - Defaults
- Remote application I/O handling
  - Real-time output monitoring
    - Textual output
    - 2D graphical output
  - Drive STDIN
- Cluster resource management
  - Queuing
  - Optimize resource utilization
- Extensible framework
  - Design patterns
  - Software architecture
  - Open-source
  - Open-standards
Native Input Representation

- Optimized for scientific & numerically oriented applications
  - Hierarchical structure
  - Vectors and Matrices
- Input Deck Model
  - Containers
    - Holds parameters
    - Holds other containers
  - Parameters
    - Stores values of data
    - 18 types implemented
- Solution: use XML AND Schema
  - Hierarchical
  - Content and Structural
    - constraints
    - validation
  - Represent flat-files (base case)
Generated GUIs

- Tree-based navigation of containers and parameters
- Containers
  - Define layout grid (graph paper)
- 18 parameter data types
  - Optimized UI component of each type
  - Positioned on graph paper layout
- Maximize component reusability
  - Usage in standalone application
  - Usage in web applet
From Schema to Application Input

Option 1: Style-Sheet Transforms
- No modification to applications

Option 2: Direct XML Binding
- Natively read & write XML
Application Integration

- Shell scripting
  - Unix command line commands
  - Connect XSLT Transformation and application invocation
    - Application native input
- Python scripting
  - Interpreted, interactive, object-oriented language
  - Use Python for “main” of application
  - Python delegates to native Fortran/C/C++ for high-performance computing routines
  - Leverages existing native scientific code
- Existing Python-binding automation tools
  - High-performance
    - Bindings have little overhead
  - SWIG, Boost.Python, F2PY and others
WIGLAF

Client Browsers

Dynamic Shell scripts

Cluster

Execution

Applications

WIGLAF

Client Browsers

Dynamic Python scripts

Cluster

Python bindings

Fortran C C++

Shared libraries

Globus Toolkit

Grid Services

PyGlobus

Python frontend

Cluster

Python bindings

Fortran C C++

Shared libraries

Grid Resources
Example Applications

- **Caltech CMDF ITAP-MD**
  - Tersoff NVE calculation
    - No changes to application
    - Used XSLT for XML to native input file format
    - Shell scripting to drive application

- **JPL EZTB materials simulation package**
  - Tight binding electronic structures calculation
    - Refactored software into standalone modules
    - Exposed module interfaces to Python using SWIG
    - Created C++ data structures representing XML input data hierarchy
      - Exposed to Python using SWIG
    - Python XML binding to populate C++ data structures
    - Python scripting for “main” driver
Integrating CMDF and EZTB

- Application schemas merged
  - Combined input parameters into a logical application description
- XSLT to separate XML input files
- Used Shell script for glue
  - sequential run of ITAP-MD and EZTB
- Interface data structure between JPL and Caltech software
  - EZTB reads output data from ITAP-MD for electronic structure calculation
Conclusion

- Demo
- Contact info
  Joey Czikmantory
  joeycz@jpl.nasa.gov
  (818) 393-7083

- More info

- Group webpage
  http://hpc.jpl.nasa.gov