A Framework-Based Approach to Science Software Development

Stephen H. Watson
Steven A. Larson
Kaly Rengarajan

20 March 2000
Topics

- TES Project Description
- TES Science Data Processing System
- Framework Requirements
- Framework Components
- Current Status
- Conclusion
TES Project Description

- Tropospheric Emission Spectrometer (TES)
  - Fourier Transform Spectrometer
  - One of 4 Instruments on-board EOS-CHEM Platform
  - Launch Scheduled for 12/02
TES Project Description (cont.)

- Science Goals
  - Further understanding of long-term variations of minor gases in the troposphere
  - Understand resulting effects on climate and the biosphere.
  - Provide global maps of tropospheric ozone and its photochemical precursors.
  - Produce database of the three-dimensional distribution of gases important to tropospheric chemistry, troposphere-biosphere interactions, and troposphere-stratosphere exchange
TES Science Data Processing System
Framework Requirements

- Protect science algorithms from environment changes
- Implement common functionality
- Encapsulate utilities and 3rd-party packages
- Minimize run-time overhead due to large data volumes (>60 GB/day)
- Highest level of reliability
System Diagram

- Data Objects
- Log
- Process Control
- Algorithm/IF
- Foundation
- Parameters
- Exceptions

JPL

NASA
Current Packages

Environment
TAI
FileIO
DataTypes
Exception
parameters
Log
Utilities

All packages may depend on Log, Exception and Utilities packages
Environment and Algorithm Packages

- Provide a generic processing environment interface, so the same application code can run in any of the specified environments (SIPS/DAAC/SCF).
- Provide a generic application implementation.
- Support a mechanism to build an application from the components of framework.
- Provide a generic construct which supports the derivation of level-specific algorithm implementations.
TES_Environment Design
Algorithm Interface Design

- Based on bridge design pattern
- Intent: "decouple an abstraction from its implementation so that the two can vary independently."
Parameter Handling Requirements

- Provide a mechanism to receive parameters from multiple sources - command line, application's parameter file, production environment's parameter file, environment variables and application default setting.

- Provide methods/functions to access input parameter values.

- Inhibit updates to parameter values during the application execution.
Parameter Handling Design

- Implemented as a singleton design pattern
- Intent: "ensure a class only has one instance, and provide a global point of access to it."

```csharp
<<Singleton>>
FW_SParameters

// Parameters
masterParameters : FW_CParameterBlock
commandline : FW_CmdLinePB
// _instance : Parameters* = 0

// GetParameter(value : template<T>&, name : String, name2 : String, name3 : String) : <template>
GetParameter(name : string, name2 : string = "", name3 : string = "") : <template>

toString() : string
	toString() : stringstream
	GetCommandLine(argc, argv)
	readParams()
	readDefinitionFile()
	readFileParams()
	readEnvParams()
	isValidCmdLine()

<<template>>
FW_AParameter

// value : <template>
// type : String

// <<virtual>> performValidation()
// FW_AParameter()
// set()
```

```csharp
1..n
FW_CParameterFile (from LogTagFile)

// getParameterBlock()
// open()
// FW_CParameterFile()

FW_CParameterBlock

// blocksize : int
// parameterList : map<string, FW_IParameterBase >
// name2
// name

// add()
// FW_CParameterBlock()
// <<virtual>> set()

FW_IParameterBase

// name : String
// source : String

// <<virtual>> set()
// <<virtual>> get()
// <<virtual>> get()
// <<virtual>> toString()
// <<virtual>> toString()
```
Data Handling Requirements

- Use a generic structure that will support a broad range of application data types.
- Provide TES-specific data types such as interferogram, spectra, etc.
- Provide a mechanism for applications to extend the TES-specific data types to implement application-specific high level data types.
Logical Data Objects

- Based on composite design pattern
- Intent: "compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly."

```
FW_CLLogicalDataObject
  ◆getterProcessHistory()
  read / write itself to / from an interface

FW_CLLogicalDataObject_Group
  ◆add()

FW_CLMath_Vector
  ◆fft()

FW_CLFocalPlane
  ◆FW_CFocalPlane()

FW_CLScan
  ◆FW_CScan()

FW_CLfgm
  ◆FW_CLfgm()

FW_CLSpectra

<<Interface>>
FW_ILDO_RAW
(from TES Data Files)
  ◆<<<virtual>>>> read()
  ◆<<<virtual>>>> write()
```

JPL

NASA
File I/O Introduction

- File handling has proven to be most complex component (so far)
- Difficult to reduce coupling with data objects to a minimum
- Multiple File Format requirement causes some difficulty
File I/O Requirements

- Provide a common interface to a file, independent of its physical format, for use from an application.

- Insulate the application programs from the underlying library (such as SDP tool kit/ HDF-EOS libraries).

- Provide a mechanism for an application to write and read data to and from:
  - HDF 5
  - HDF EOS
  - Binary
  - ASCII
Additional Packages

- Log files
- Exception Handling
- Process Control
- Utilities
- 3rd-Party Packages
Current Status

• Architectural and High-Level Designs Complete
• Detailed Designs for Environment, AI, Parameters and File I/O under way
• Iterative Development - Additional requirements resulting from Level x subsystems result in new packages
• Initial internal delivery 00Q3, external β 01Q1
Conclusion

- High level of reusability
- Reliability is paramount
- Multi-platform support
- File handling most complex portion to date
- Complex, evolving system requirements imply iterative development approach
- Development continues until 2002 - interim results at IEEE Aeroconference 01!