Science Data Processing Framework using UML and Rational Rose

Akbar Thobhani
Software Engineer
Jet Propulsion Laboratory
California Institute of Technology
Agenda

- Introduction to TES Framework
- UML
- Rational Rose
- Framework Design
- Lessons Learned
- Questions
NASA / JPL's TES

![Diagram of TES Instrument with labels: Limb View and Nadir View]
TES

Tropospheric Emission Spectrometer

TES will make the first direct global measurements of tropospheric ozone as it measures atmospheric infrared radiation at high-wavelength resolution. TES will also measure tropospheric carbon monoxide, methane, nitric acid, water vapor, nitric oxide, and nitrogen dioxide.
Tropospheric Emission Spectrometer (TES)

- Provide world’s first three-dimensional map of Tropospheric ozone and its photochemical precursors
- Launches in 2002 for a six-year mission

Data processing Per Year
Data Products Generated

Interferogram

Spectra

Profiles
Organization of TES - Product Generation Executable (PGE)

Data Packets → Decommutation → Interferograms

Subset Interferograms → Calibration Processing → Target Processing

Generate 3D Map → Atmospheric Profiles

Reconstruct Dataset → Subset → Ordering

Retreival
Data Processing Environment

- Input Files
- Parameters
- PGE
- Algorithm
- Environment
- Product Files
- Log Files
Science Data Processing System

- Must operate on more than one facility
  - Batch environment
  - Interactive environment
- Provide distributed processing
- Data I/O in more than one format
- Adapt to constant revisions of processing algorithm
Need for Framework

- Abstract PGE related issues and solve them once
- Reduce cost by reusing code
- Provide rapid development of PGEs

"A partially completed software application that is intended to be customized to completion" - Gregory Rogers

Framework Design Goals

- Protect science algorithm from environment changes
- Provide PGEs with a generic “shell” for Algorithms
- Implement common functionality (e.g. File I/O, Parameter handling)
- Encapsulate COTS and GOTS Libraries
SDPS Framework

Level Specific Algorithm

Data Objects  Product Files

Algorithm Interface

Generic Support
  Error Handling  Process Control

COTS Library Interface

Parameter Handling

Log & Error Files

File Handling
Using UML

Diagrams we most commonly use
Analysis

Data Flow Diagram

- To understand the problem
- To communicate with non OO members of the team

Use Case Diagrams

- To show coverage of requirements within developers
Conceptual Design

- Object Diagram used to show "high level" objects in the system
- Collaboration Diagrams used to show the interaction between these objects.
- Activity Diagram used more like a flow chart
High Level Design

Class Diagrams often used to show
  The class hierarchy
  Relationship between classes
  Interfaces (public and protected methods only)

Analyze if any
  Design Patterns apply
  Data Structures are needed
High Level Design (cont’d)

- Object Diagram
- Sequence / Collaboration Diagram
- Activity Diagram
- State Diagram
Detailed Design

All the diagrams

Class' documentation includes
- Public, protected and private member
- State charts

Operations
- English description with pseudo-code
- Pre and post conditions
- Assertions and testing criteria
Rational Rose

CASE tool for OO Modeling

JPL

NASA
Overview

Document all stages of design
  • Diagrams using UML notation
  • Specification of classes, operations, attributes
  • Rose Model is one stop for all documents
  • Organize design in subsystems, and categories
  • Highly customizable
Generating Documents

• Documents we generate
  • Interface Specification
    • Class Interface
    • File Interface
  • Design Documents
• Create Visual Basic-like scripts to generate documents
• Rational Soda simplifies generating word documents using Rational Rose
Code Generation and Reverse Engineering

- Forward Engineer
  - Generates skeleton code for classes (both .h and .cpp files)
  - Writes documentation in code
- Reverse Engineer existing code into the Model
- Keeping code and documentation in Sync
Framework Design
SDPS Framework

Level Specific Algorithm

Data Objects  Product Files

Algorithm Interface

Generic Support

Error Handling  Process Control

COTS Library Interface

File Handling

Parameter Handling

Log & Error Files
Design Overview
Parameters

- Provides uniform access to all parameters from Command Line, Environment, and File Parameters
- Apply precedence rule to the parameters
- Global access provided using **Singleton** pattern
- The parameter file can be organized in nested blocks of parameters
Parameter Hierarchy

FW_ParameterFile
(from LogicalFile)
- getParameterBlock()
- open()

FW_ParameterBase
- name : String
- source : String
- <<virtual>> set()
- <<virtual>> get()
- <<virtual>> toString()

<<Singleton>>
FW_Parameters
- Instance()
- GetParameter()
- toString()
- SetCommandLine()
- readParams()
- readDefinitionFile()
- readFileParams()
- readEnvParams()
- validateCmdLine()
- FW_Parameters()

FW_ParameterBlock
- parameterList : map
- add()
- FW_ParameterBlock()
- set()

<<template>>
FW_Parameter
- value : <template>
- type : String
- <<virtual>> performValidation()
- FW_Parameter()
- set()
Framework Design

Environment
Environment

- Uniform interface for different environments
- Algorithms can run on different environments without any major changes
- Environment executes algorithms using Algorithm Interface (AI)
- AI is the middle man between Algorithm and Environment
Environment

<<Singleton>>
FW_CTES_Environment

- m_OS_Information
- m_Other_Information
- $ _Instance : FW_CTES_Environment *

readEnvParams()
<<virtual>> executeAlgorithm()
<<virtual>> -FW_CTES_Environment()
<<static>> Instance()
<<virtual>> writeToOperatorConsole()
<<virtual>> writeToDevice()

<<Singleton>>
FW_SPParameters
(from parameters)

FW_ATES_AI
(from TAI)

SCF_Env
- SCF_Env()
- ~SCF_Env()
- <<virtual>> executeAlgorithm()
- writeToOperatorConsole()
- writeToDevice()

SIPS_Env
- SIPS_Env()
- <<virtual>> ~SIPS_Env()
- <<virtual>> executeAlgorithm()
TES Algorithm Interface

<<Singleton>>
FW_CTES_Environment
(from Environment)

FW_ATES_AI
- FW_CTES_AI()
- <<virtual>> CreateFileList()
- <<virtual>> Startup()
- <<virtual>> Run()
- <<virtual>> Shutdown()

FW_ATES_Algorithm
- m_InputFiles : FileList
- m_OutputFiles : FileList
- FW_CTES_Algorithm()
- <<virtual>> Run()
- <<virtual>> ~FW_CTES_Algorithm()

L1A_AI
L2_AI

L1A_Algorithm
L2_Algorithm
TES_Environment Execution
Diagram

10: createFileList()

9: Startup()
12: Run()

11: TES_Algorithm(inFileList, outFileList)

13: run()

main

1: SetCommandLine(argc, argv)

2: TES_Environment()
8: executeAlgorithm()

3: readParams()

TES_Environment

: Parameters

4: readDefinitionFile()
5: validateCmdLine()
6: readEnvParams()
7: readFileParams()
Framework Design

File I/O
File I/O Overview
Reading and Writing Data

All TES Data Types should inherit from Logical Data Objects

Logical Data Object, using the Data Access Interface, should read and write itself from a file

Data Access Interface is overloaded by all TES Data Physical Files
Logical Data Object

FW_CLLogicalDataObject
- getProcessHistory()
- <<<<virtual>>>> write()
- <<<<virtual>>>> read()

read / write itself to / from an interface

FW_CLDO_Group
- FW_CFocalPlane
- FW_CScan

FW_CMMath_Vector
- FW_CIfgm
- FW_CSspectra
Data Access Interface

<<Interface>>
FW_ITES Data Access Interface

read(name, data : int 8&): void
read(name, data : uint 8&): void
read(name, data : int 16&): void
read(name, data : uint 16&): void
read(name, data : int 32&): void
read(name, data : uint 32&): void
read(name, data : float 32&): void
read(name, data : float 64&): void
read(name, data : char 8&): void
read(name, data : uchar 8&): void
read(name, data : String&): void
read(name, data : BUFFER&): void
openGroup(name : String, type : String): void
closeGroup(): void
write(name, data : int 8): void
write(name, data : uint 8): void
write(name, data : int 16): void
write(name, data : uint 16): void
write(name, data : int 32): void
write(name, data : uint 32): void
write(name, data : float 32): void
write(name, data : float 64): void
write(name, data : char 8): void
write(name, data : uchar 8): void
write(name, data : BUFFER): void
write(name, data : String): void
File I/O

- Uses the **Bridge** pattern
  - Logical File provides an abstraction -- A class hierarchy for file interface
  - Physical File provides an implementation -- A class hierarchy for file type-specific implementation
- File Wrapper (**Adaptor** pattern) encapsulates the file API
Physical File Logical Diagram

FW_CASCII Wrapper
(from FileInterface (Wrapper))

FW_APhysical File

FW_CPHysical Parameter File

FW_CL0_Pkt_PhysicalFile

FW_CTES Data Physical File
(from TES Data Files)

FW_CBinary TES Data File
(from TES Data Files)

FW_CRaw Binary Wrapper
(from FileInterface (Wrapper))
File I/O Dependency

L1A_AI
(from TAI)

FW_CScan
(from DataTypes)

FW_CFocalPlane
(from DataTypes)

FW_CLfgm
(from DataTypes)

FW_CL1AFile
(from LogicalFile)

FW_CHDF TES Data File
(from TES Data Files)
1: FW_Scan()

4: L1AFile(filename, "HDF")

6: putNextScan(Scan1)

7: write(physicalFile)

2: FW_FocalPlane()

11: write(physicalFile)

8: openGroup("Scan", uniqueScanName)

9: write("scan number", scanNumInt)

10: write("observation type", nadirStr)

5: TES Data Physical File()

22: closeGroup()

3: FW_lfgm()

15: write(physicalFile)

12: openGroup("Focal Plane", uniqueFPName)

13: write("FP ID", fpIDInt)

14: write("Detector Temperature", dtTempDouble)

21: closeGroup()

physicalFile: HDF

TES Data File

16: openGroup("lfgm", uniquelfgmName)

17: write("pixel", pixelNumInt)

18: write("quality data", qdInt)

19: write("lfgm 1 data", lfgmData[])

20: closeGroup()
File Layout Diagrams

- Every Logical Data Object (LDO) has an associated layout
- The Layout object specifies how to organize a LDO in a file (i.e. how the read/write calls are to be interpreted)
- Avoids coupling the Logical Data Objects with the File Layout by using the Chain of Responsibility pattern
File-layout Diagram
Using Patterns
Data Object-file Relationships

FW_LogicalDataObject (from DataTypes)

FW_LDO_Group (from DataTypes)

FW_Math_Vector (from DataTypes)

TES Data File (from LogicalFile)

L1AFile (from LogicalFile)

L2File (from LogicalFile)

Layout

HDF TES Data File (from PhysicalFile)

Binary TES Data File (from PhysicalFile)

HDF4x Wrapper (from HDF4x Interface)

IfgmHdf4Layout

ScanHdf4Layout

LayoutFactory

FW_ITES Data Access Interface (from TES Data Files)
Lessons Learned
For Managers

- Selling the idea
- Framework should be a stand alone system
- Define scope of Framework and abide by it
- Iterative development fits best
For Managers

- Design meetings and reviews benefit from small groups
- High learning curve for tools and methodology
- Training all team members is essential but expensive
For Designers

Communicating your design to new (or non) OO designers

  - Class diagrams can be too complicated
  - Object diagrams, Data Flow diagrams, and Block diagrams work best

  - Traditional design tools are still very useful (e.g. DFD, State diagrams ...)


Any Questions?
Contacting the Author

Akbar Thobhani
Software Engineer
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109

akbar.thobhani@jpl.nasa.gov
(818) 354-5158
References

Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, *Design Patterns*, Reading, MA: Addison-Wesley, 1995

Acronyms

AI - Algorithm Interface
API - Application programming Interface
COTS - Commercial off-the-shelf software
FW - Framework
GOTS - Government off-the-shelf software
HDF - Hierarchical Data Format
JPL - Jet Propulsion Laboratory
L1A - Level 1A (subsystem of TES)
L2 - Level 2 (subsystem of TES)
LDO - Logical Data Object
NASA - National Aeronautics and Space Administration
OO - Object Oriented
PGE - Product Generation Executable
TAI - TES Algorithm Interface
TES - Tropospheric Emission Spectrometer
UML - Unified Modeling Language
Acknowledgements

I would like to thank the following individuals for their valuable assistance
- Stephen Watson
- Ken Scott

The research described in this presentation was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration