GSAW 2007

Framework concept for satellite operations

Miguel Angel Molina
Head of the Flight Engineering Business Unit, GMV Aerospace & Defense, Madrid, Spain

Gonzalo Garcia
VP of Operations, GMV Space Systems Inc., Rockville, MD, USA

INNOVATING SOLUTIONS

INTRODUCTION

INNOVATING SOLUTIONS

Introduction

- Presentation shares **applied design principles** and **lessons learned** in the design, implementation and deployment of a framework-oriented solution for multi-satellite, multi-user operational Flight Dynamics Systems (FDS)

- Many conclusions **can be extrapolated** to other areas

- Some **similarities to other architectures presented at GSAW** in the past (NASA GSFC's GMSEC, ESA's EGOS, ...). Main differences:
  - Fully financed by a corporation (vs. coordinated or financed by a space agency)
  - Subsystem level (vs. ground system level)

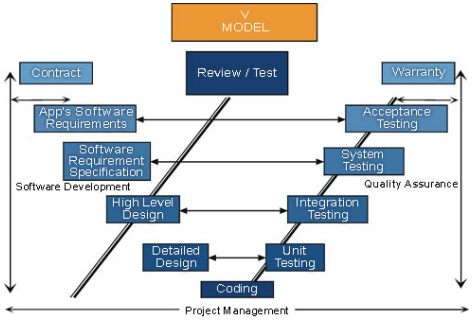
Conceived in 1998, operational since 2001

Selected for multiple missions (70+ satellites) by reference operators in the US, Europe, Asia and Australia

THE PROBLEM

INNOVATING SOLUTIONS

Typical issues in the development of Operational Flight Dynamics Systems



- Significant amount of mission-specific requirements: **Mission-driven designs**
- Future **reuse is not a requirement**
  - *Who pays for that?*
- **Reliability** is key: Reuse of flight-proven legacy modules is an implicit requirement
  - *Nobody wants to be the first user*
- Typical: **Multi-satellite, multi-platform, multi-user**
- System must be **easy to maintain and customize**
- **Programmatic risks** must be minimized
  - *Schedule & cost*
- In some areas, FDS is a **commodity**: Small margins, little incentive to invest
- **Obsolescence** issues: Typical GEO mission: 15 years
  - *Anybody ordering their operational HW through ebay?*

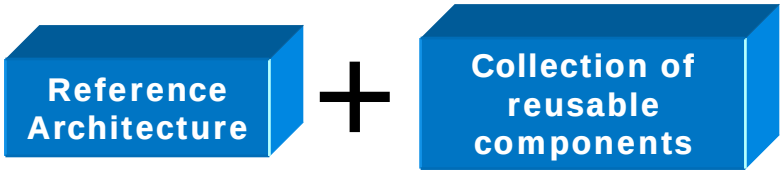
FRAMEWORK CONCEPT

INNOVATING SOLUTIONS

Framework Concept

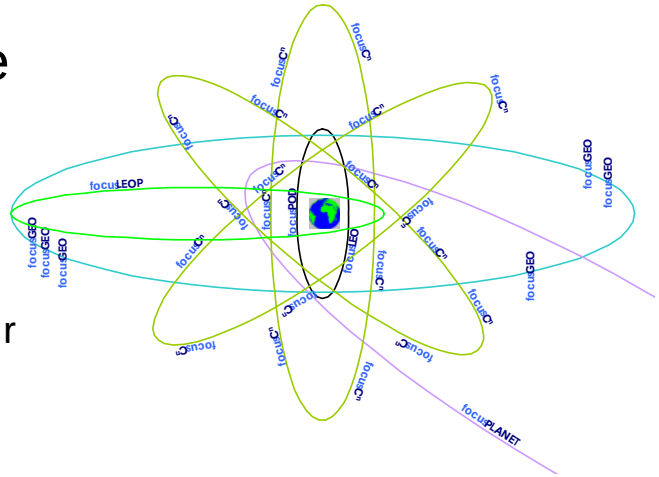
- **Framework** (per *Design Patterns*, by Gamma et al, Ed. Addison and Welsey):
  - *A set of collaborative classes that makes up a reusable design for a specific class of software.*
  - *A framework provides architectural guidance by partitioning the design into abstract classes and defining their responsibilities and collaborations*
  - *A developer customizes the framework to a particular application by subclassing and composing instances of a framework of classes*

- A **simplified view** of a framework is:



Framework Development Steps (1)

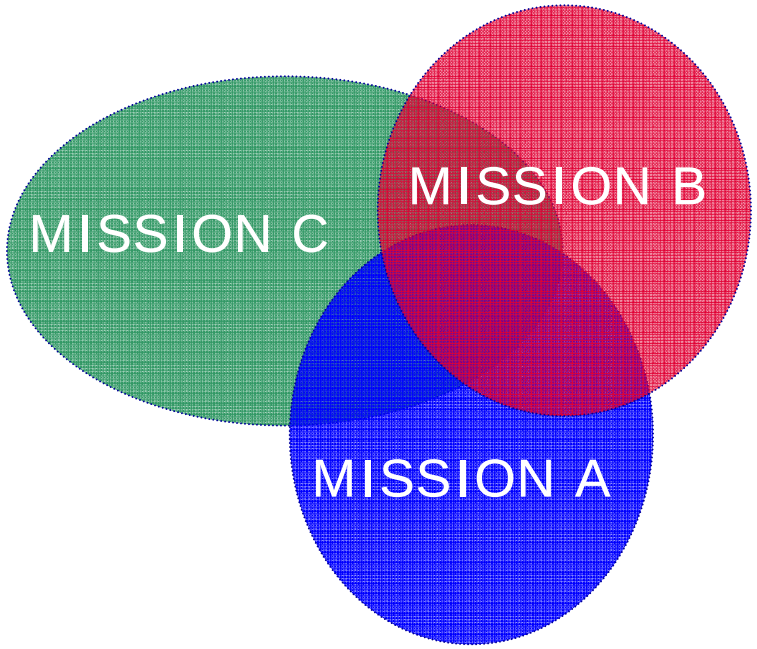
- 1) Identification of **target systems** to be developed in the future
  - In our case, very ambitious:
    - GEO, LEO, LEOP, MEO, HEEO, interplanetary, interstellar??
    - Multi-satellite (up to 800…), multi-platform, multi-user

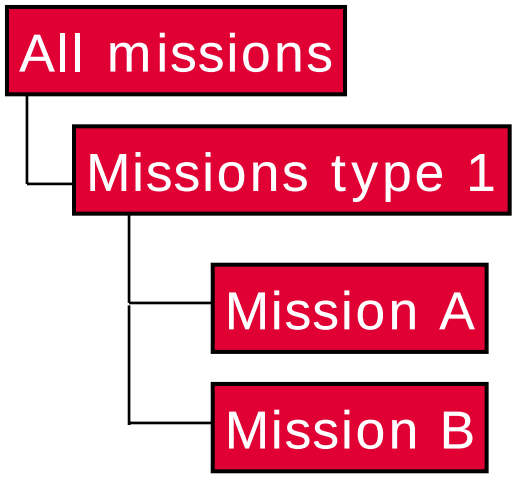


- 2) Capture of **key reference requirements** that define baseline for the design and would support target systems
  - Flight dynamics algorithms capable of supporting multiple types of missions
  - Data manipulation and visualization
  - Event logging
  - Internal communications
  - Process management
  - Automation support
  - Reporting
  - Flexible & scalable architecture
  - Portability
  - OPEN:
    - Capable of integrating external components and mission-specific components
    - API: Application Programming Interface
  - Powerful User's Interface
  - Plotting capabilities
  - 2D / 3D Visualization
  - Multi-user support, many access levels
  - Undo/redo: Impact on architecture

Framework Development Steps (2)

- 3) Separation between **stable requirements** (applicable to all missions) and **variable requirements** (to be covered by ad-hoc modules)
- 4) Definition of a **reference architecture** that will support all those requirements
  - It must be scalable, able to support large constellations as well as single-satellite missions via a stand-alone deployment
  - Ensure architecture fits all target missions
- 5) Identification of the standard components of the framework that would support ALL target systems. Create **hierarchy** and timeline (**roadmap**)
  - Find the right balance between abstraction and specialization



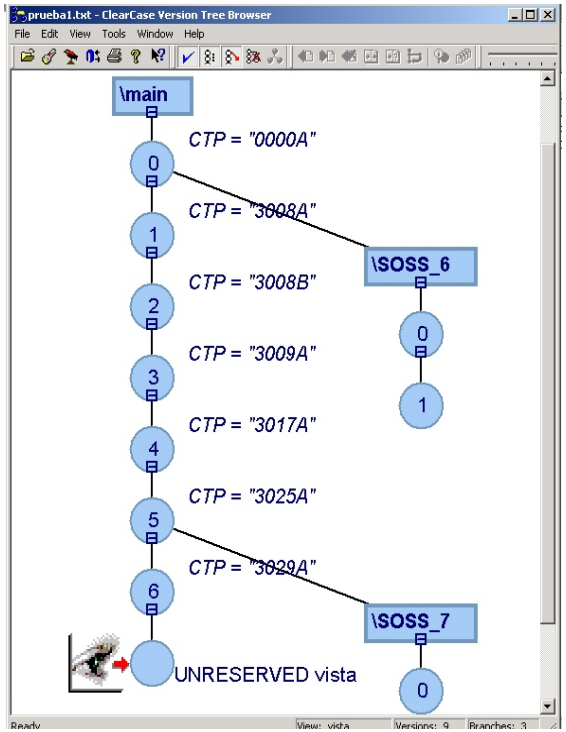


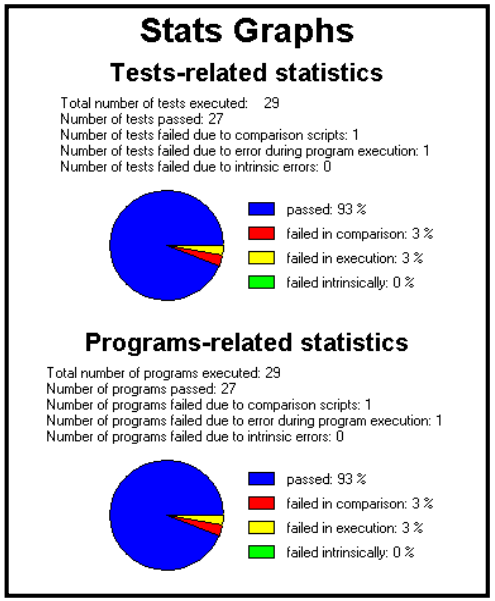
Framework Development Steps (3)

- 6) Selection of **key technologies** for the development of the components in the framework
  - This includes reuse or of legacy components (e.g. PEPSOC, NAPEOS, originated by ESA)
- 7) Development of the **core standard components**
  - They must be easily configurable to enable a quick deployment for a particular mission via a simple modification of configuration files
- 8) Successful **deployment** of the first operational system
- 9) Use new missions to develop the remaining standard components, enhance existing ones and **follow the roadmap**

Framework Maintenance

- Take advantage of each deployment of the framework to **retrofit improvements** to the standard components
- A **strict configuration management** is key to support the different branches (sometimes they cannot be avoided) and to continuously improve the root of the framework



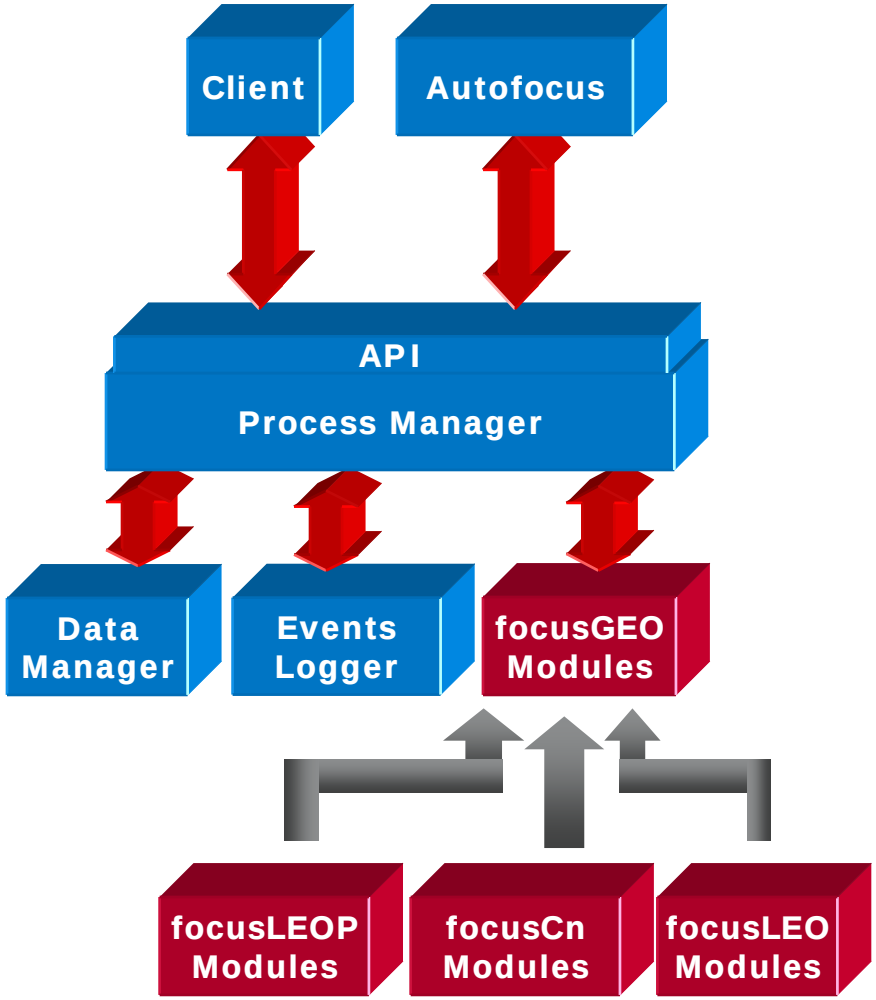


- **Automatic regression testing** of framework components is essential to
  - reduce the costs of each deployment
  - exploit the benefits of the framework

A REAL-LIFE EXPERIENCE: *focusSuite*

INNOVATING SOLUTIONS

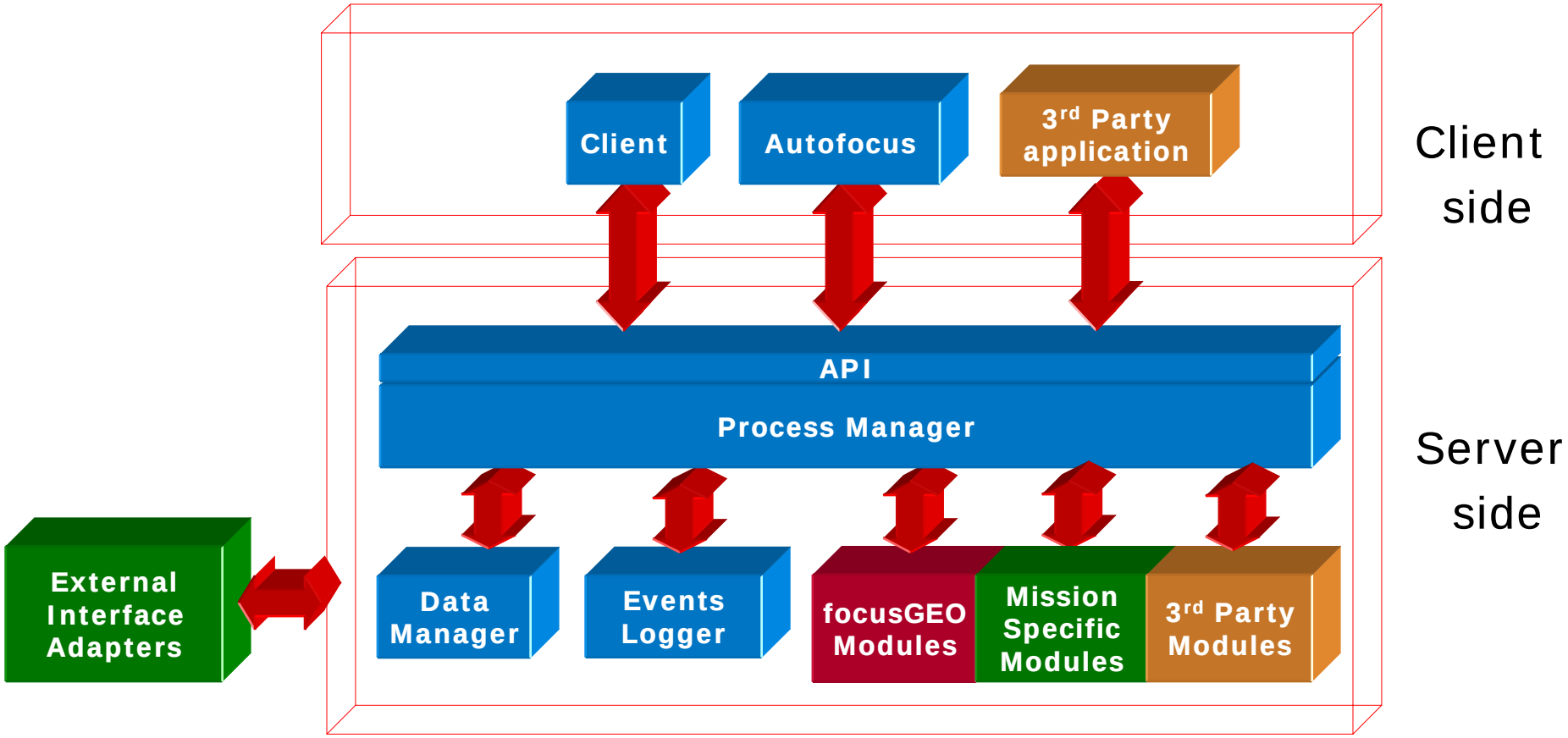
focusSuite initial design and technologies



- **Three-tier architecture**, to promote flexibility and modularity, allow distribution and scalability:
  - 1st Tier: Presentation (clients)
  - 2nd Tier: Process management
  - 3rd Tier: Data management / Computation

  Typical distribution = client / server
- Same technology allows **quick deployment** of stand-alone systems
- Standard **API** for interaction with Process Manager and easy integration with other architectures (e.g. NASA's GMSEC) via adapters
- Internal communications: TCP/IP based (sockets)

focusSuite: A typical deployment

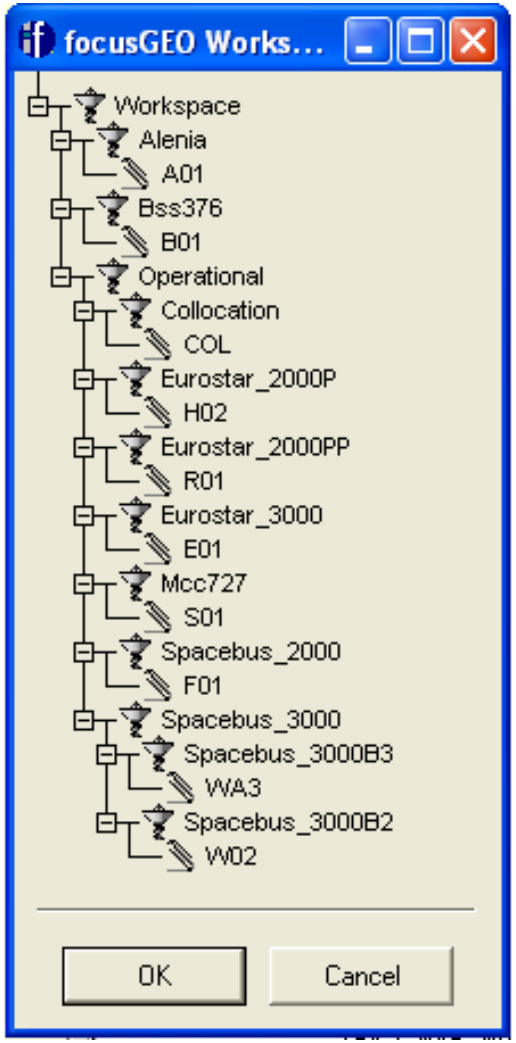


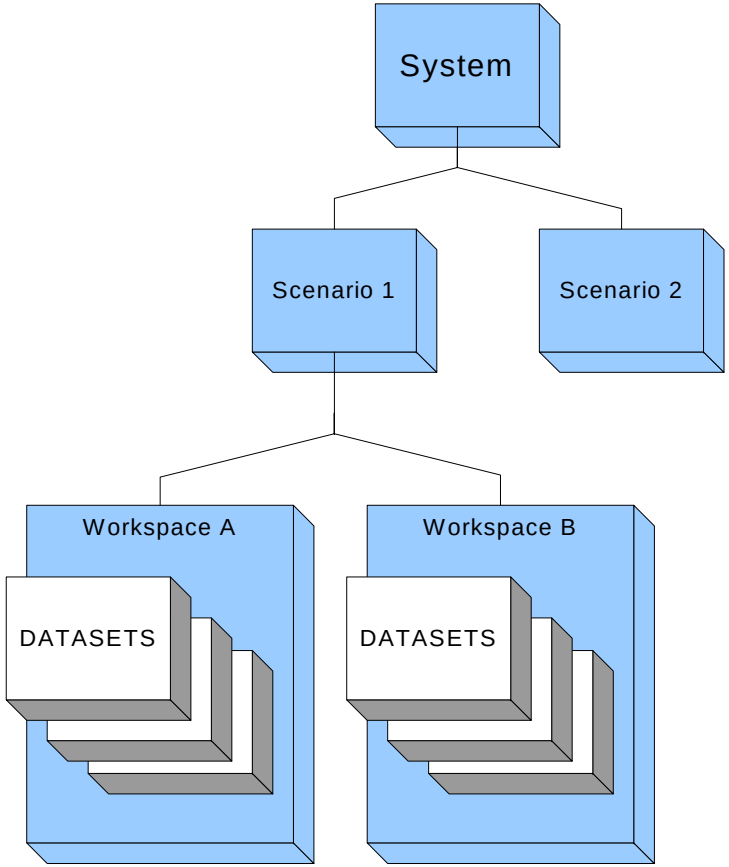
Other design principles:

- Adaptation of the system via **configuration files** as much as possible to avoid the need to recode and to enable direct integration of external computation modules:
  - GUI panels: TkForms toolkit
  - Structure of FD functions
- Promote **separation** between software technologies and flight dynamics technologies
- Use of reliable, stable, portable **programming languages**:
  - Tcl/Tk for the infrastructure (GUI, communications, process management, etc)
  - Fortran (77 / 90) for the computation layer. Why?
    - Reuse of flight proven modules and libraries
    - Stability & portability
    - Well known by FD experts
- Definition of XML-based languages for the exchange of products with external subsystems: E.g. **OrbitML,** or standards under definition by CCSDS

focusSuite Scenarios & Workspaces

- Data are organized in a tree structure
- Enables multi-platform & multi-mission support
- Users privileges may be different for each workspace
- Useful to separate different areas (e.g. operational / testing / training) or platforms

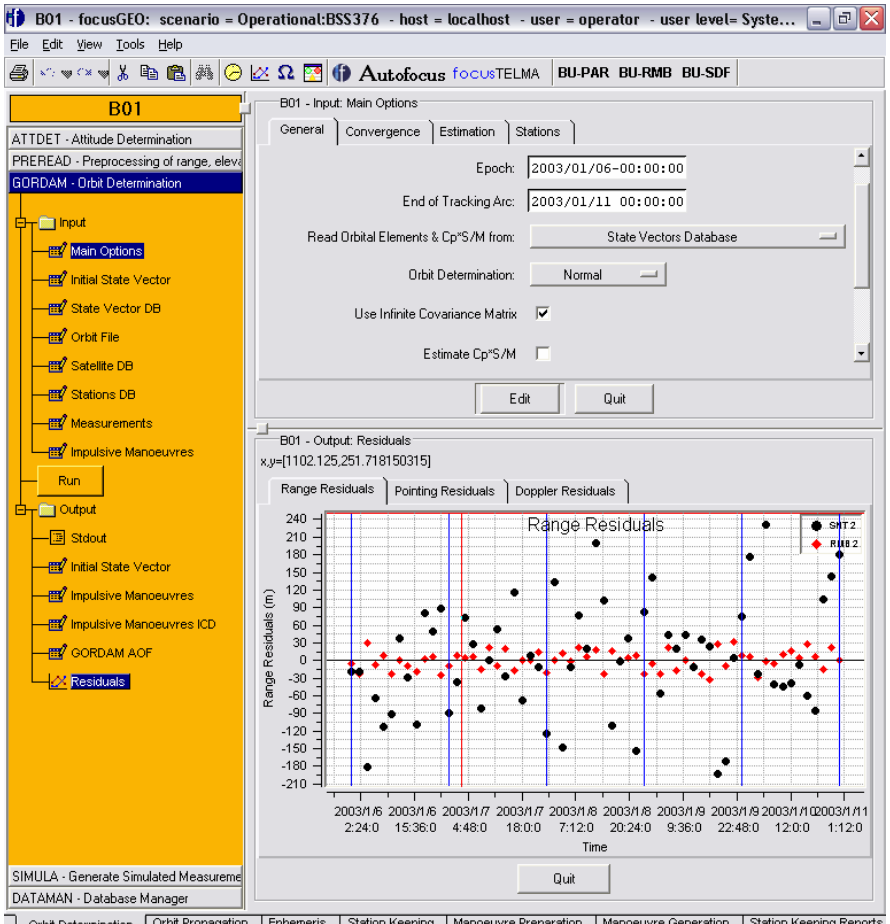


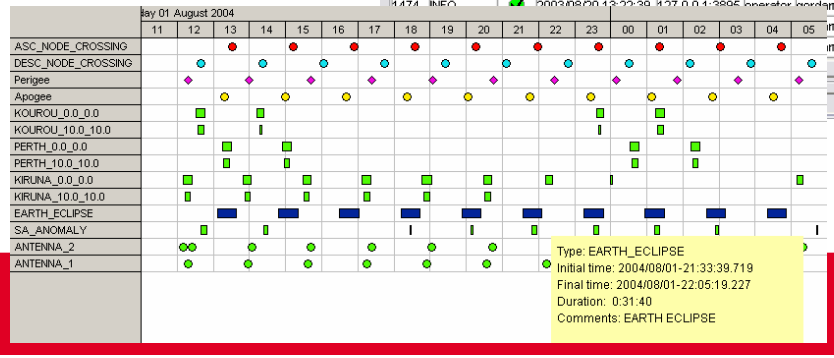


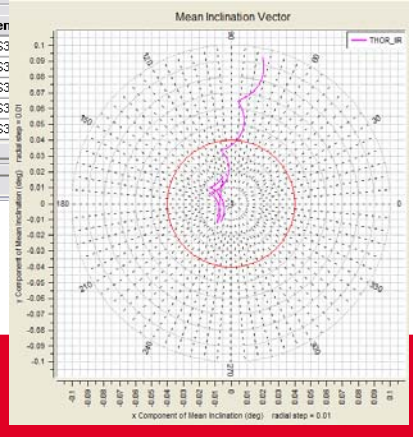
focusSuite

GUI design principles

- All-in-one
- One-click-away
- Promote graphical information
- Easy navigation (functions, panels)
- Adapted to FD applications
- Powerful plotting tools & Gantt charts
- Help user avoid errors
  - Undo/redo (multi-user)
  - In-line help
  - Disable / enable fields if applicable

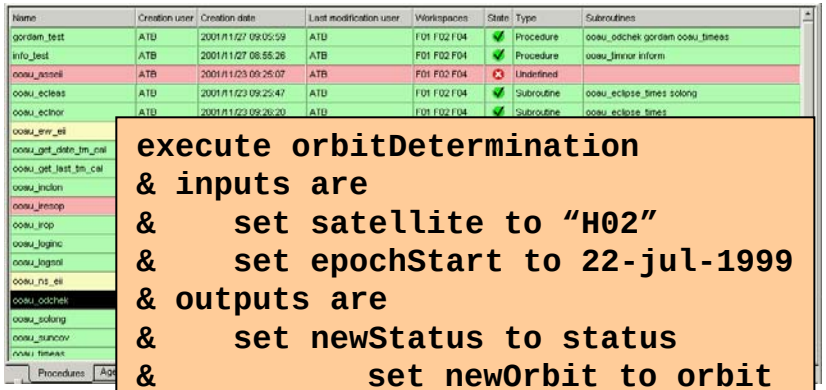






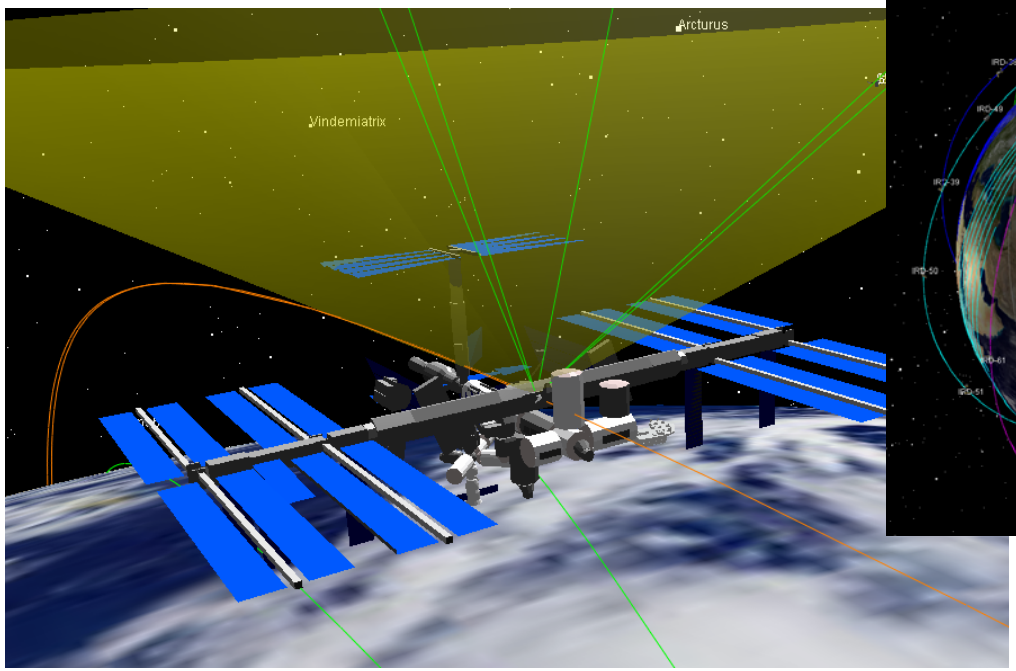
Optional components

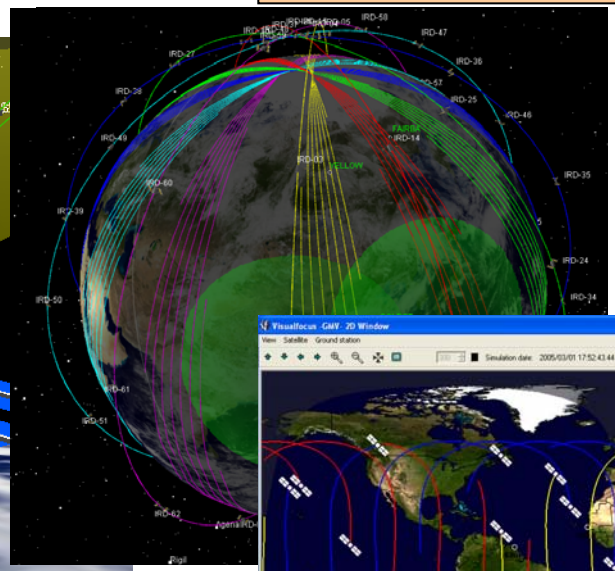
- Autofocus:
  - Full automation support for hands-off operations, based on procedures
  - Designed from scratch specifically for FD operations
- Visualfocus: 2D/3D visualization

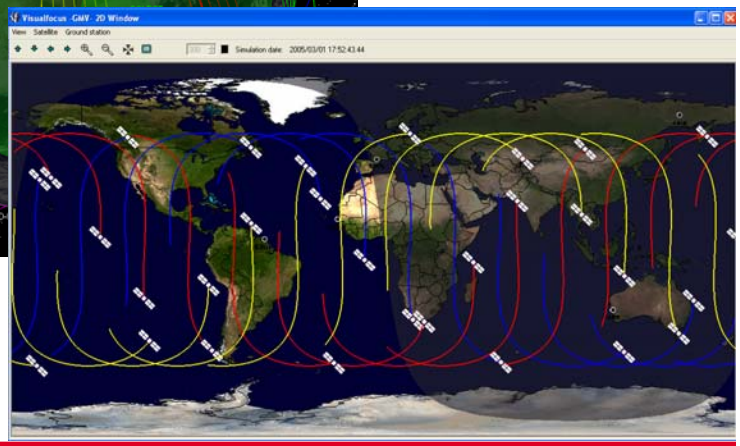


```
execute orbitDetermination
& inputs are
&     set satellite to "H02"
&     set epochStart to 22-jul-1999
& outputs are
&     set newStatus to status
&             set newOrbit to orbit

repeat while residuals > 1.0e-3
   # Perform computation here
end repeat
```







focusSuite customers

Some key references:
- EUTELSAT fleet (24 GEOs)
- Galileo (30 MEOs)
- OCO FDS (NASA JPL, LEO)

LESSONS LEARNED

INNOVATING SOLUTIONS

Lessons learned (1)

- **First user** is the hardest. Convince them that long-term benefits compensate for risks linked to change
  - In our case, EUTELSAT
- Use **prototypes** and get end users involved in the process (use-centered design)
- Use a solid base of adaptable **legacy components**
  - In our case, technology licensed from ESA (e.g. NAPEOS)
- **COTS independence** is essential to guarantee portability, avoid recurring costs and avoid long-term HW obsolescence risks
- **Start-up costs are very significant**. Only justified if they can be spread through multiple missions. Easy to lose hope.
- **Funding** to complete the roadmap must be a constant concern.
  - Customer base must reach **critical mass** in a limited amount of time

Lessons learned (2)

- **Software** experts and **domain** experts (FD) must be involved in the process
- **Synergies** are huge. Enhancements from one contract are fed back to the core components in most cases. All users benefit.
- Continuous **investment** in research & development is essential to maintain the framework alive
- Expect some **resistance** to the use of the standard framework
- Think **long term:**
  - Evolution of technologies?
  - Evolution of space mission requirements
  - HW obsolescence

  …, but not **too** long term

In the long run, we are all dead

Keynes

Thank you

www.gmv.com

INNOVATING SOLUTIONS