A Multi-mission Event-Driven Component-Based System for Support of Flight Software Development, ATLO, and Operations first used by the Mars Science Laboratory (MSL) Project

Navid Dehghani
Michael Tankenson
California Institute Of Technology
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA

June 19, 2006

Agenda

- Overview
- Architecture
  - Downlink
  - Uplink
  - Messaging
- Transition from flight software “sandbox” to flight operations
  - Test Like You Fly
- A word about the Agile Development Process

Overview – A Brief Summary

- The Mission Data Processing and Control System (MPCS) component architecture is derived from the Event Driven Model used on MER, with JMS messaging capability (reliable data transfer) used to trigger events and status
  - Enabling plug and play architecture
  - MPCS has strong inheritance from GDS components that have been developed for other Flight Projects (MER, MRO, DAWN, MSAP), and are currently being used in operations and ATLO
  - MPCS components are Java-based, platform independent, and are designed to consume and produce XML-formatted data
- MPCS components are designed to be multi-mission, or easily adapted for future missions
- The MPCS Ground System will be used to support MSL, as its first customer, during Testbed, ATLO, and Flight Operations

Overview – MPCS Key Driving Requirements

- Integrated FSW/GDS development environment on Linux platforms
  - Scripted, Repeatable Test Procedures
  - Hardware Cost Savings (No need for Suns on every desktop)
- FSW agility in development and test
  - New CMD/TLM/Product/EVR Dictionary turnaround in 15 minutes or less
  - Rapid Checkout of Flight/Ground Interfaces
- GDS Design that enables MOS Event-Driven Operations
  - Build on the MER experience, potentially high return for MOS
- Test Like You Fly, Fly Like You Test
  - This has demonstrated benefits for all missions
- DSMS Architecture (DISA) Compliance

Overview – Event Driven Concepts Drive the MPCS Design

- GDS/MOS processing is driven by Spacecraft and Ground events, rather than realtime telemetry monitoring
  - Events are generated & sent on a JMS Information Bus
- Information is represented at a high semantic level in reports, products, and messages on both Spacecraft and Ground.
  - Mission operators plan based on activity and performance reports
- This leads to more efficient mission operations, more automation, and better science return (ala MER)
  - End of test message => generate reports
  - Command message => display evr's or channels
  - Product complete message => execute specialized display programs

Architecture - Basic Design Policies (1 of 2)

- Continuous refinement -- Phased Requirements, Design, Development, Test, and Deployment “make things better, don’t try to get it perfect the first time”
  - Lightweight implementation
  - Lightweight development processes
  - Daily builds run functional tests; as-committed builds run unit tests
- Linux environment
- Java programming language
- Distribute events using Java Messaging Service; also store in MySQL database
- Distribute meta-data using MySQL database
- Distribute products as files over a shared NFS mount.
- No per-node license fees for deployed systems

Architecture - Basic Design Policies (2 of 2)

- Major processes communicate via JMS Message Bus messages and files on the file system. JMS Messages use a standard DSMS JMS header and an XML body.
  - Why use JMS Message Bus?
    - Easy n-n distribution of information
    - Guaranteed delivery of data
    - Decouples applications
    - Industry standard distributed system design pattern
    - DSMS New Architecture

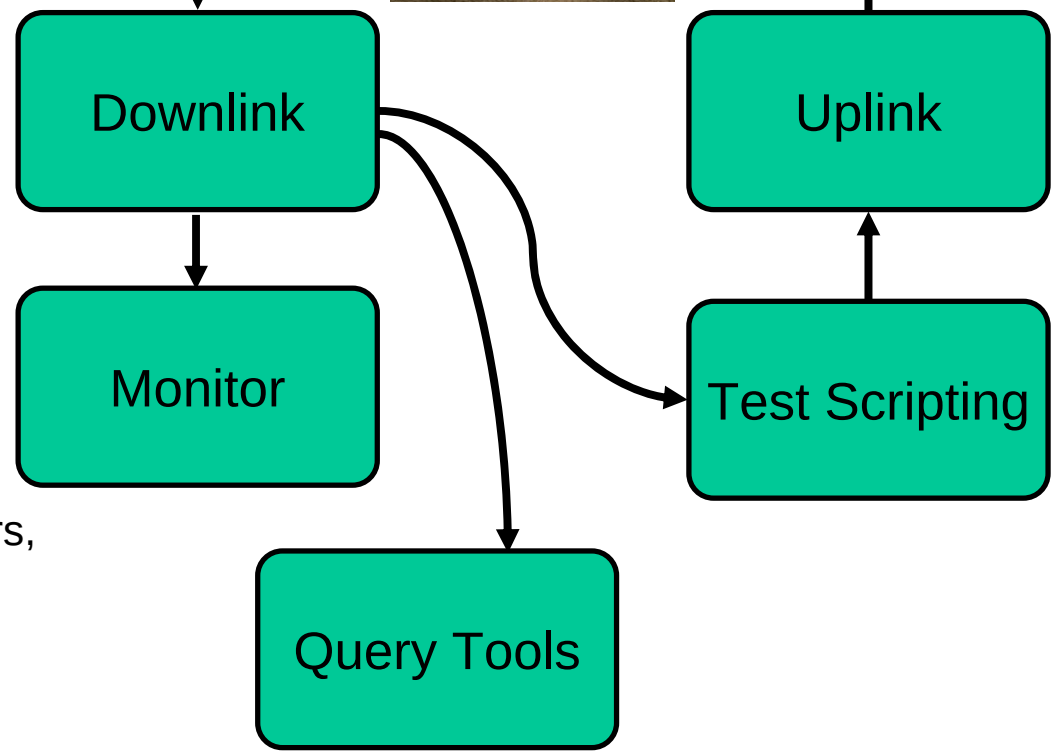
- Meta-data is written to a file in XML and stored on a SQL database.
  - Fast random access; Fast query; Complex query
  - Not a proprietary data format
  - Why XML?
    - Industry standard data description language
    - Standard parsers, generators, transforms, validators are available in all languages
    - Use RelaxNG Compact format for schema (easy to read and understand)

View From 20,000 Feet

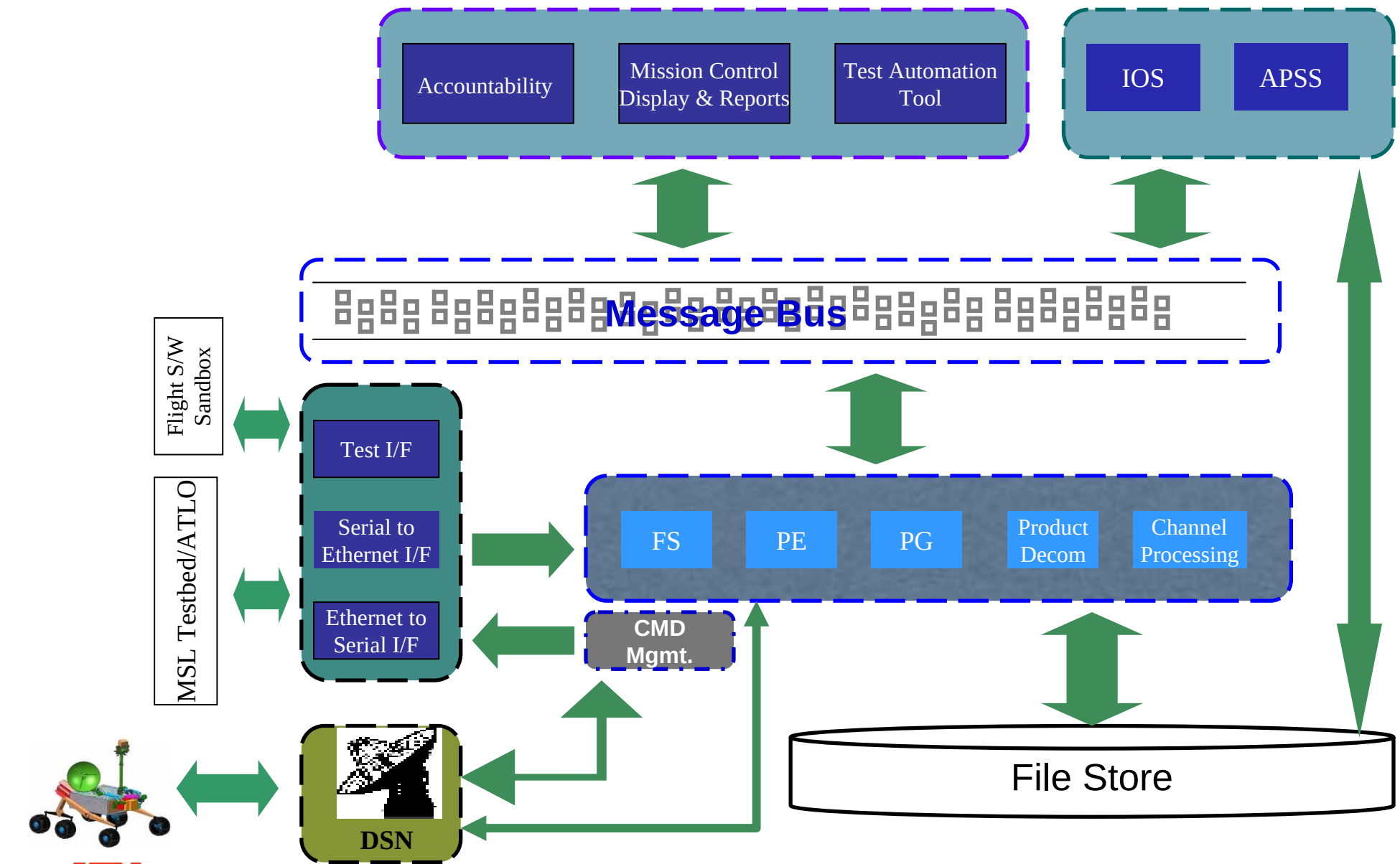
- **There will be four main processes**
  - Downlink process -- core downlink process
  - Uplink process -- core uplink process
  - Monitor process -- distributed message display; eventually test script processor
  - Test script process

  test script processor, and
  - Query tools -- report generators from archived data

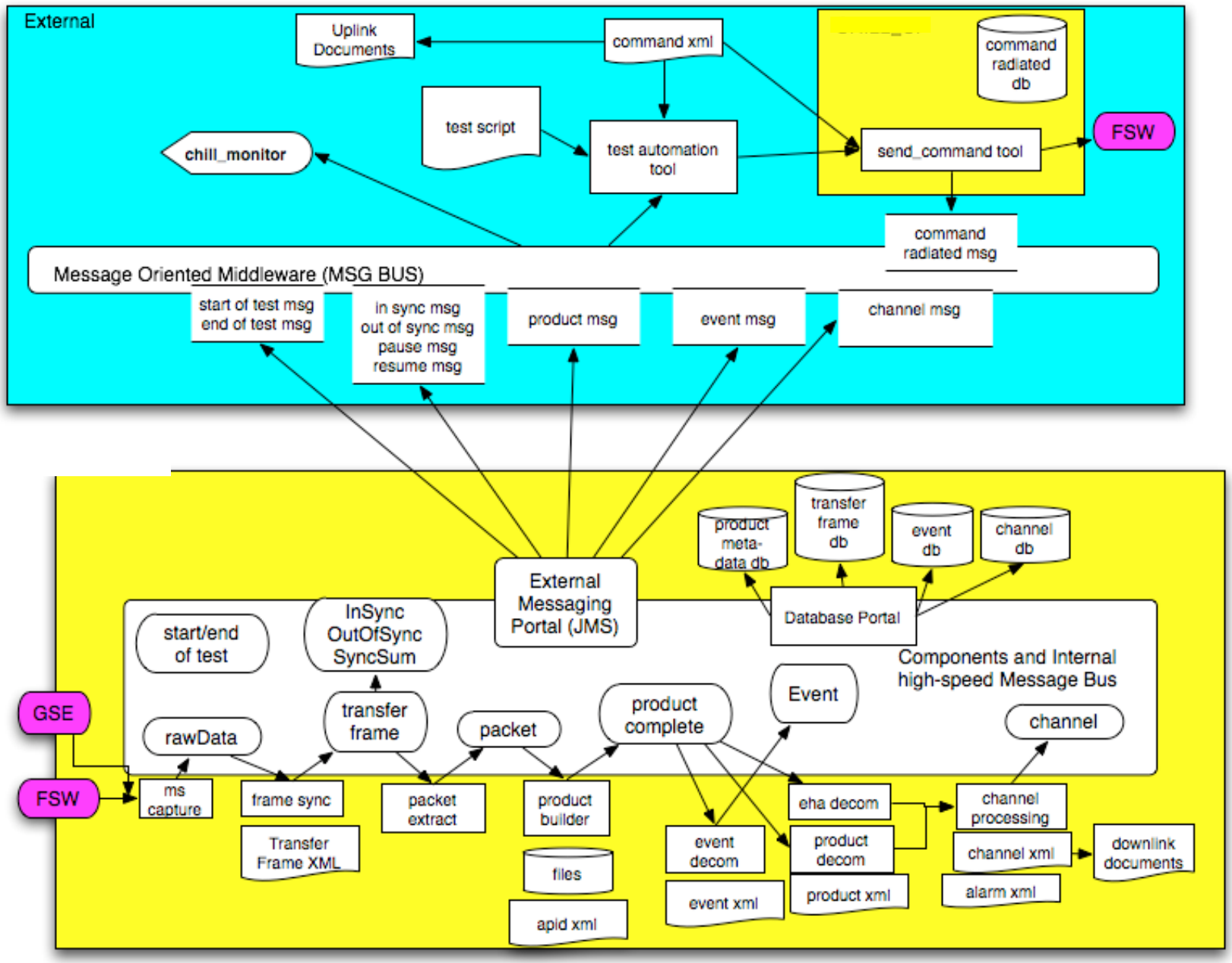
- And an umbrella tool around these processes to initiate and monitor the system
  - There will be a number of secondary scripts and tools to automate processes; simple triggers, simple queries, simple plot builders, …



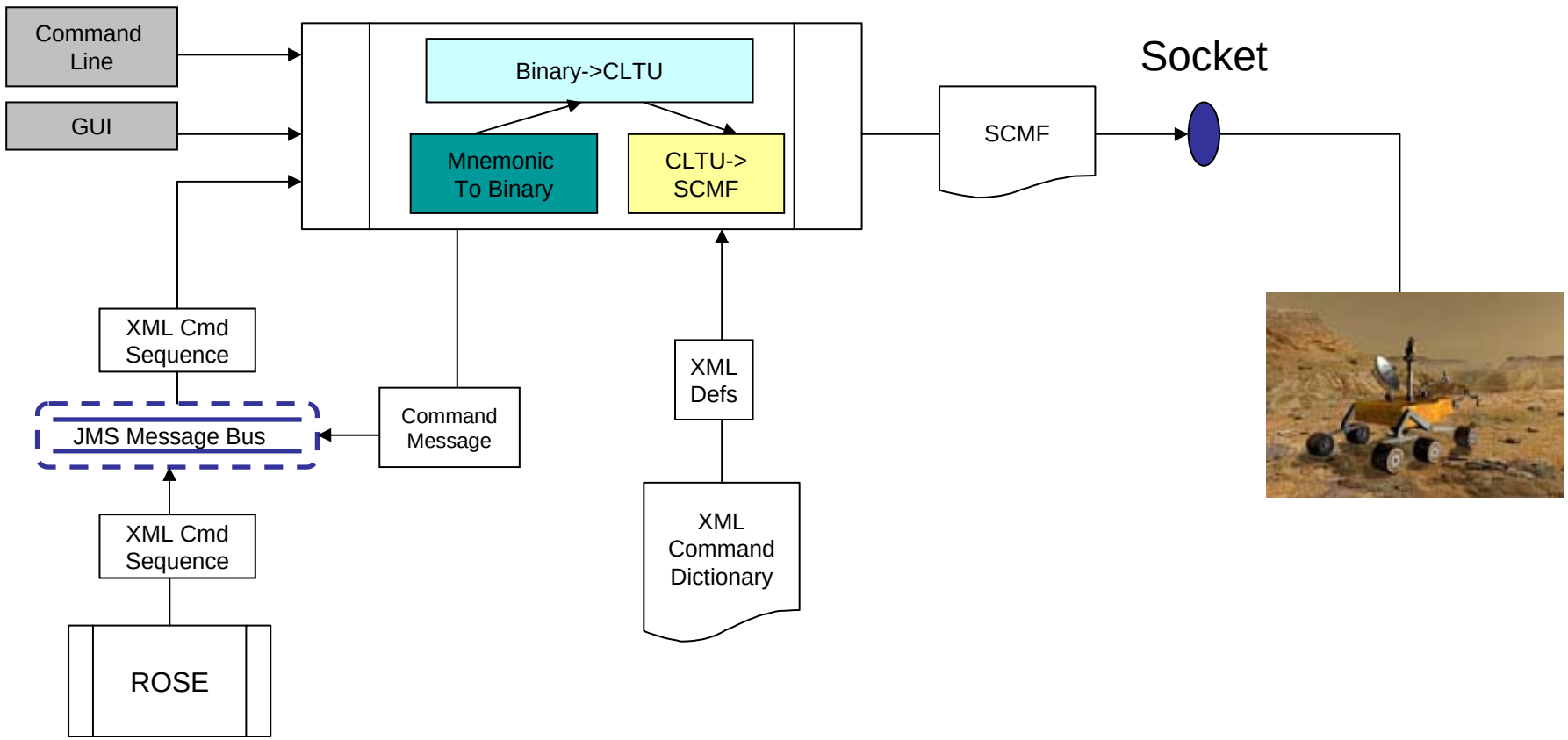
View from 10,000 Feet
MPCS Functional Diagram



View from 1000 Feet
MPCS Mostly Downlink Architecture



MPCS Uplink Process



Messaging

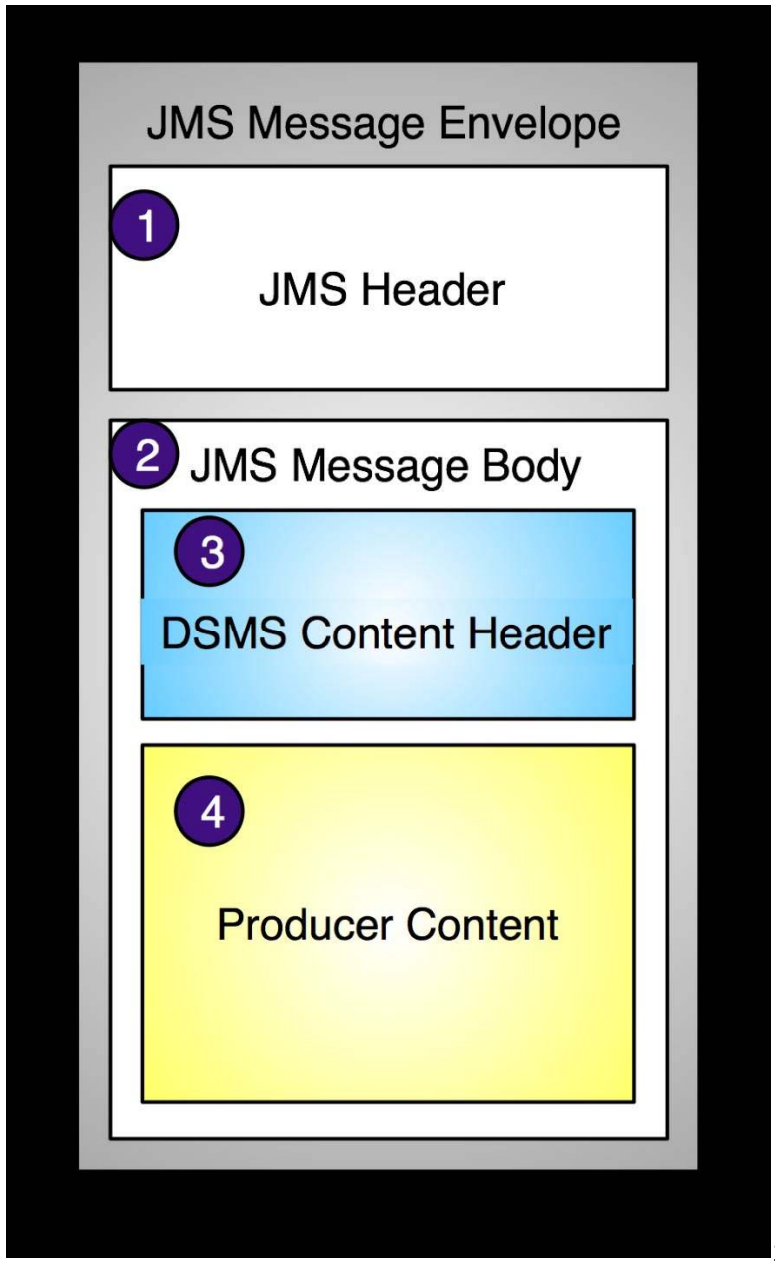
- Event-Driven Operations requires reliable data delivery
  - Ground system cannot miss critical mission events
- Internal Messages deemed to be of use to other applications must be published to the external JMS bus by MPCS applications
- Which messages are published and how often they are published must be configurable
- Subscribers must be able to filter by:
  - Mission
  - Spacecraft
  - Message Type
  - Test ID
- A variety of message display formats must be supported.
- Messages must conform to DSMS Standards

Messaging - Format

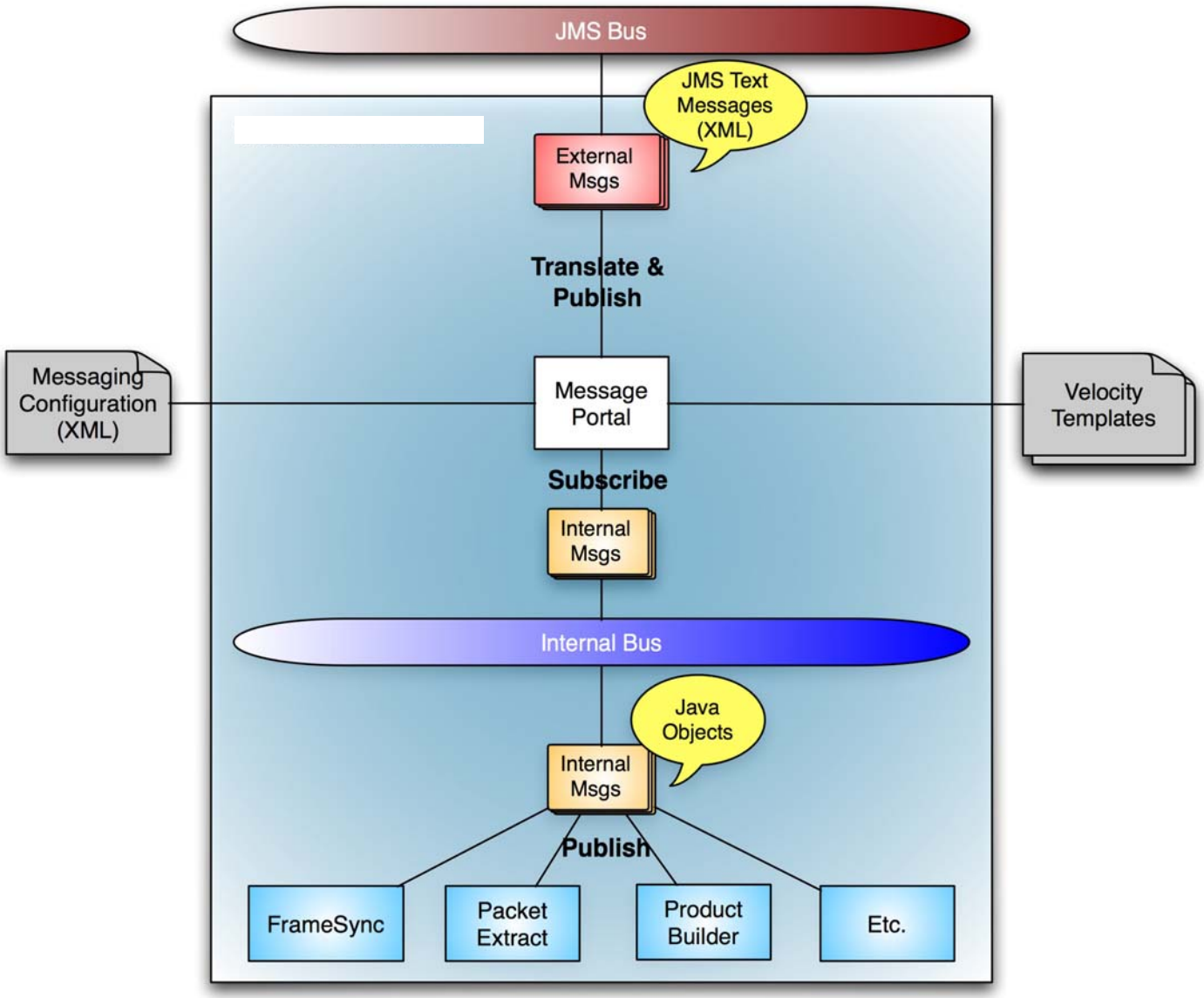
- JMS TextMessage Envelope
  - JMS Properties for Filtering
  - JMS Message Body (ASCII)
  - DSMS XML Header
  - Internal Message Content (XML)

DSMS formats are captured in Velocity Templates, so it could be changed easily if DSMS requires it

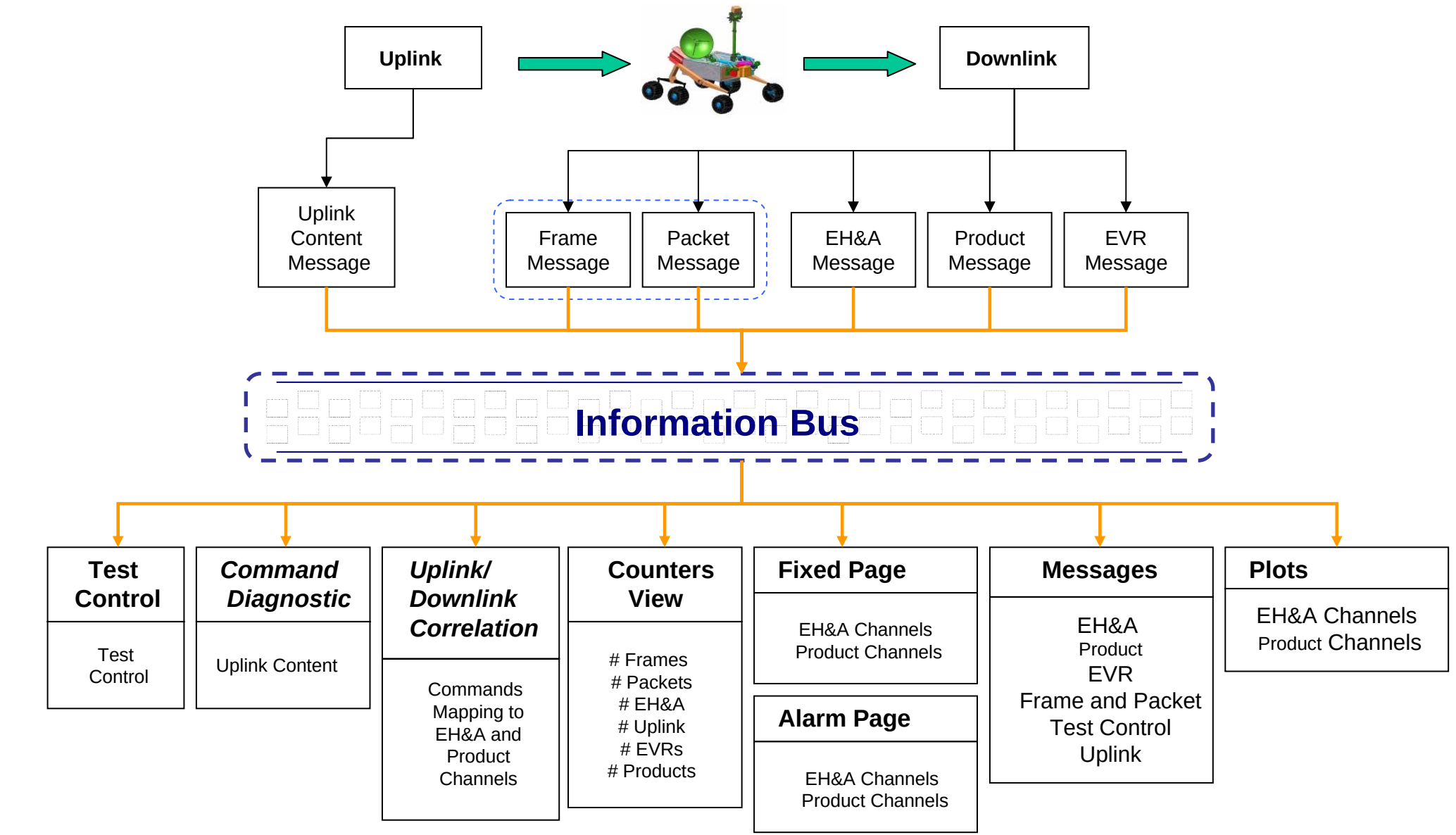
- Other Message Issues
  - MPCS “Topics” are being defined
  - Ontology and Naming Conventions are being captured in Information Models



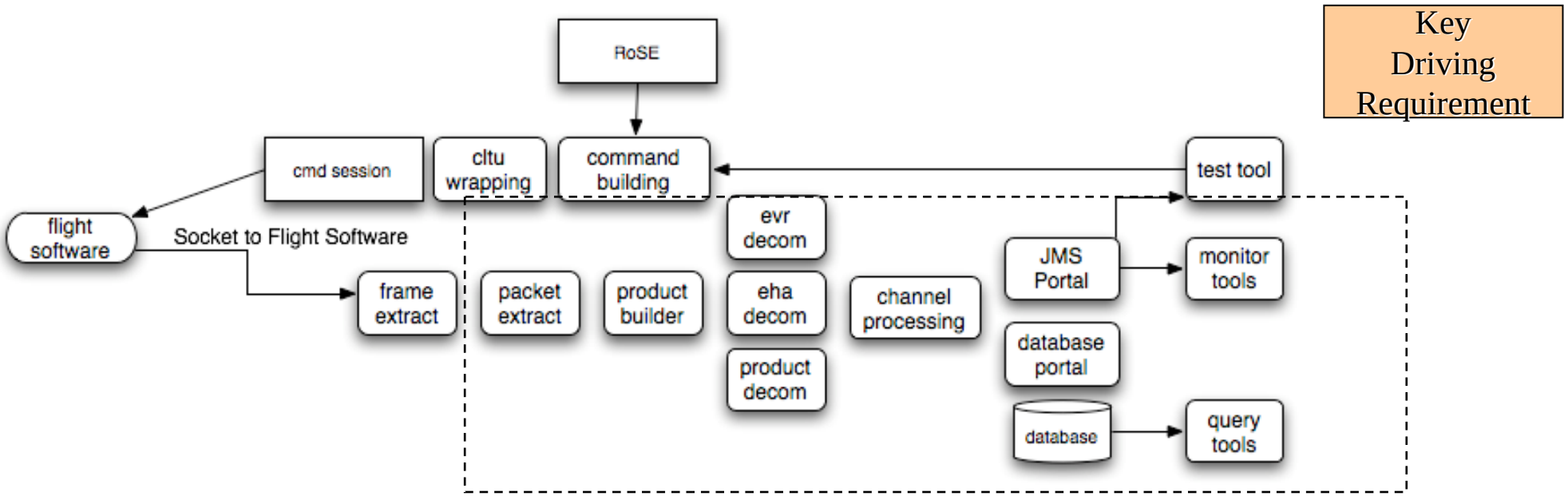
Messaging – Downlink Process



Messaging - Visualization Information Flow



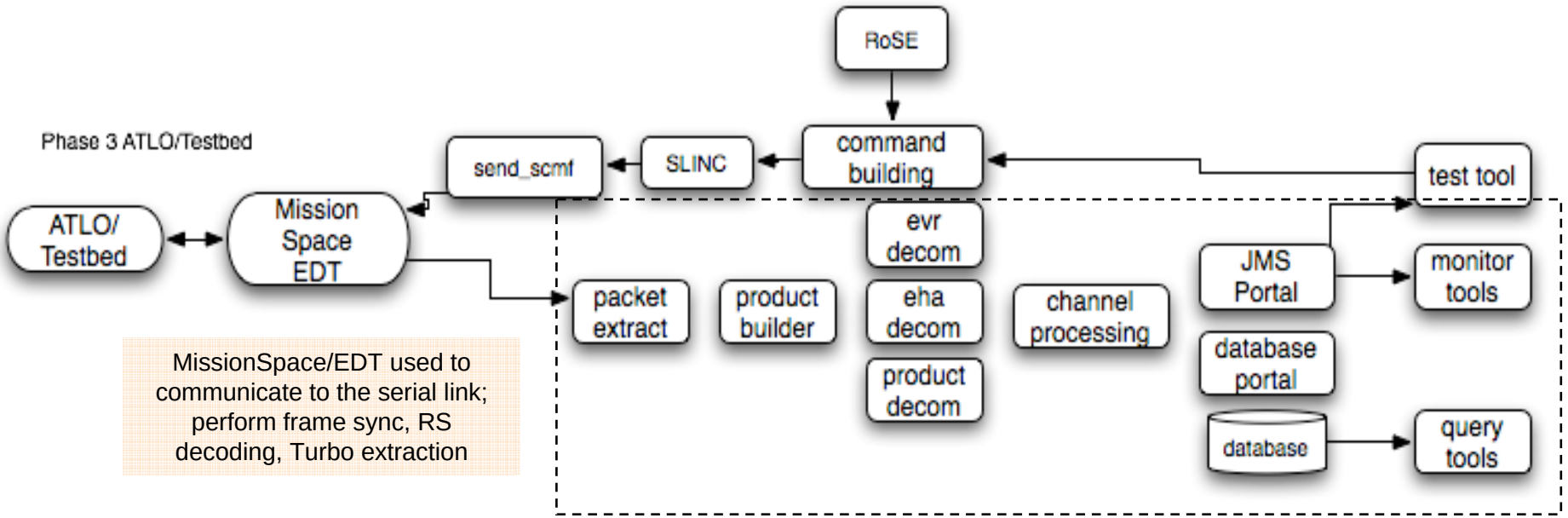
MSL FSW Test Set Environment



FSW Test Set

FSW Test Set (Sandbox) = LINUX workstation for Flight Software Developers
Consists of a development environment for Flight Software code and MPCS
About 20 to 30 sandbox environments will be deployed
Performance Driver: new command or channel database in 15 minutes
Fine Tuning “Knobs” for changing command parameters and error injection
No cross test query capability. User does own data management (store and archive).

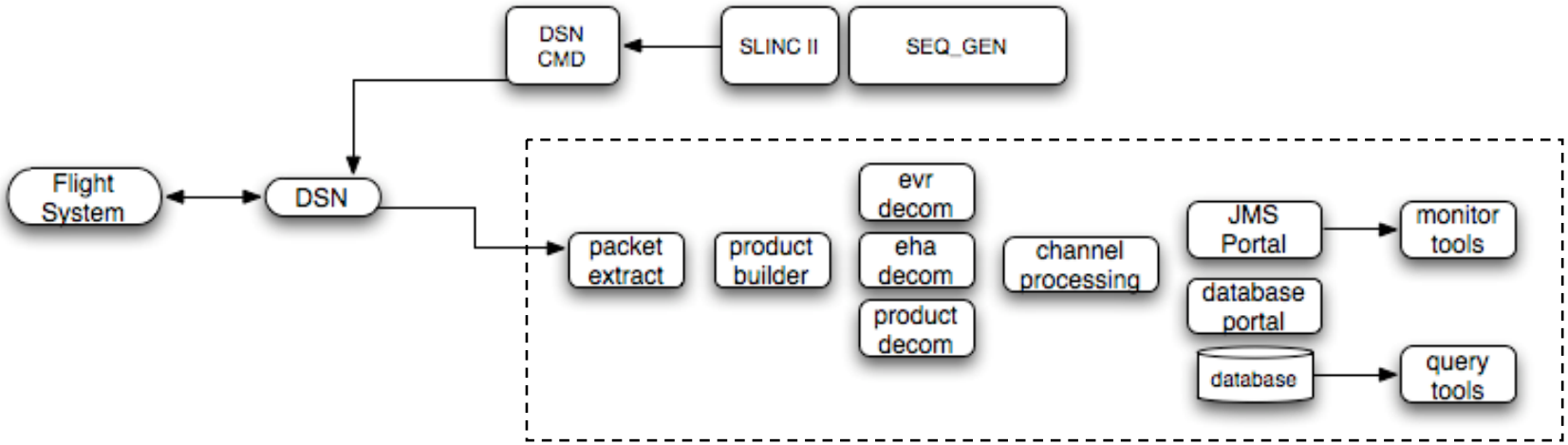
MSL I&T Testbeds and ATLO



Testbeds & ATLO

(1) “downlink/uplink/test” process per stream
10-20 workstations of “monitor/query tools”
Shared NFS for product data
Central database and web service
Query across tests
Automated data management (store, archive, restore)

MSL Operations



Operations

“downlink” for each DTE stream

(1) “downlink” for relay processing

20-200 “monitor/query tool” workstations

AMMOS uplink used to produce uplink products and send SCMF to spacecraft

Central NFS server for products

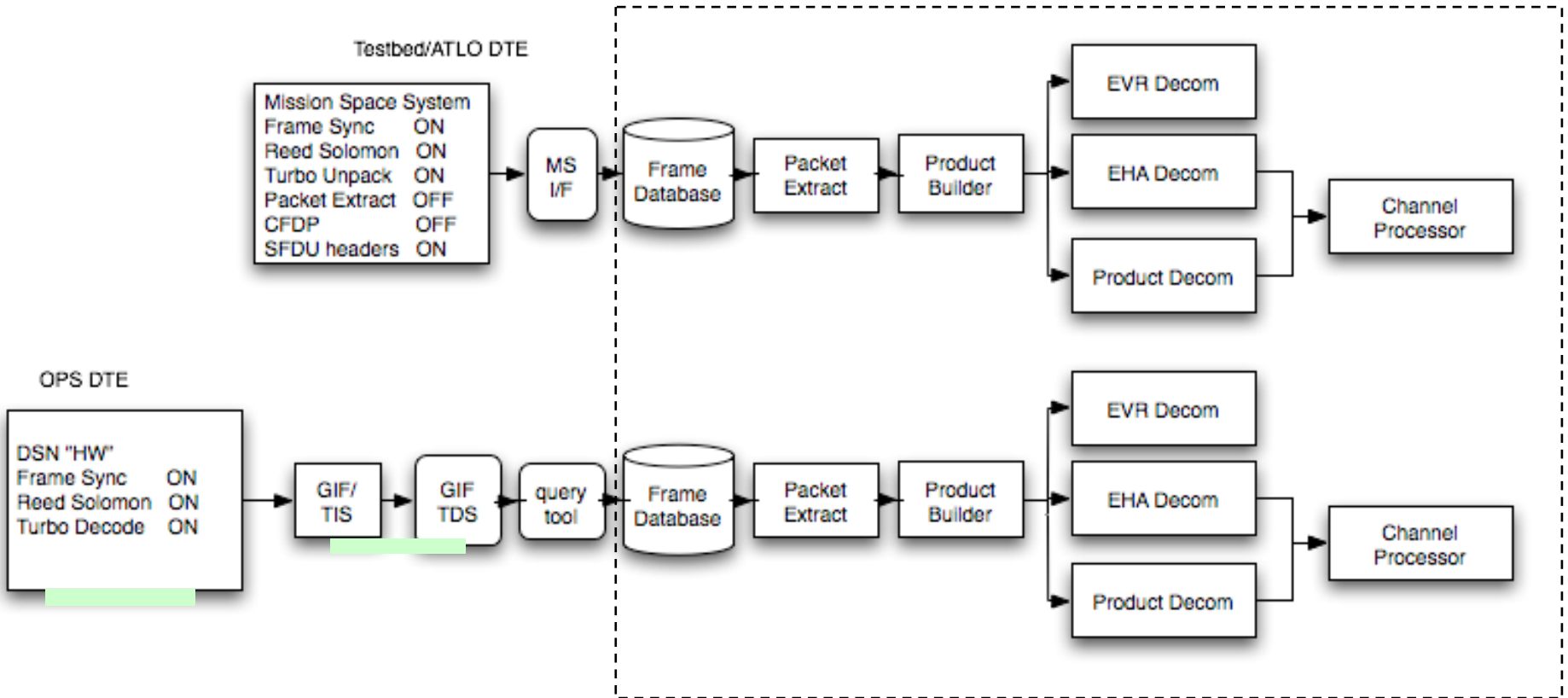
Central database and web server

MIPL interface is same as a report tool (wait for a product event, get product file)

Direct To Earth (DTE) Dataflow

Key Driving Requirement

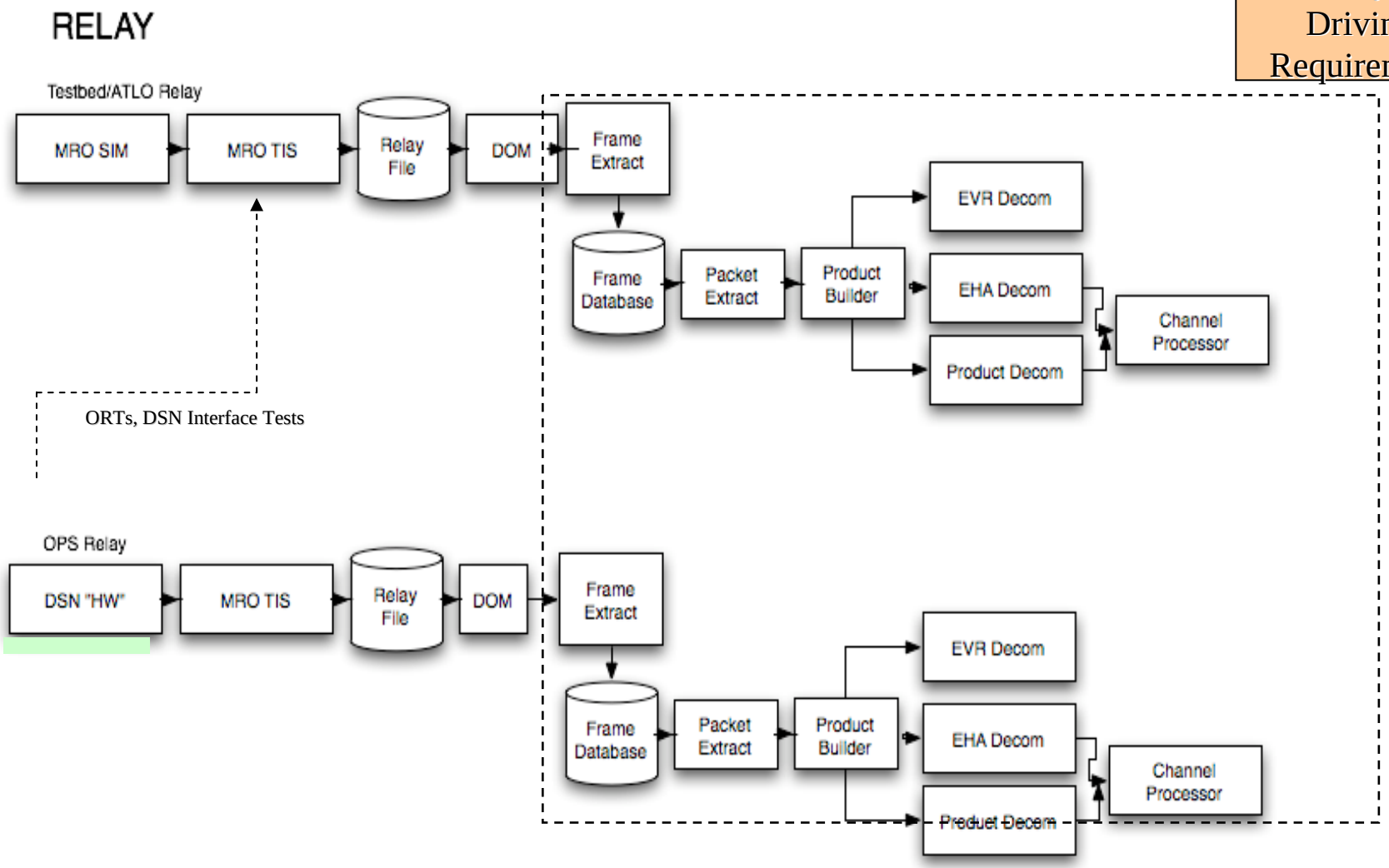
DTE



DSN

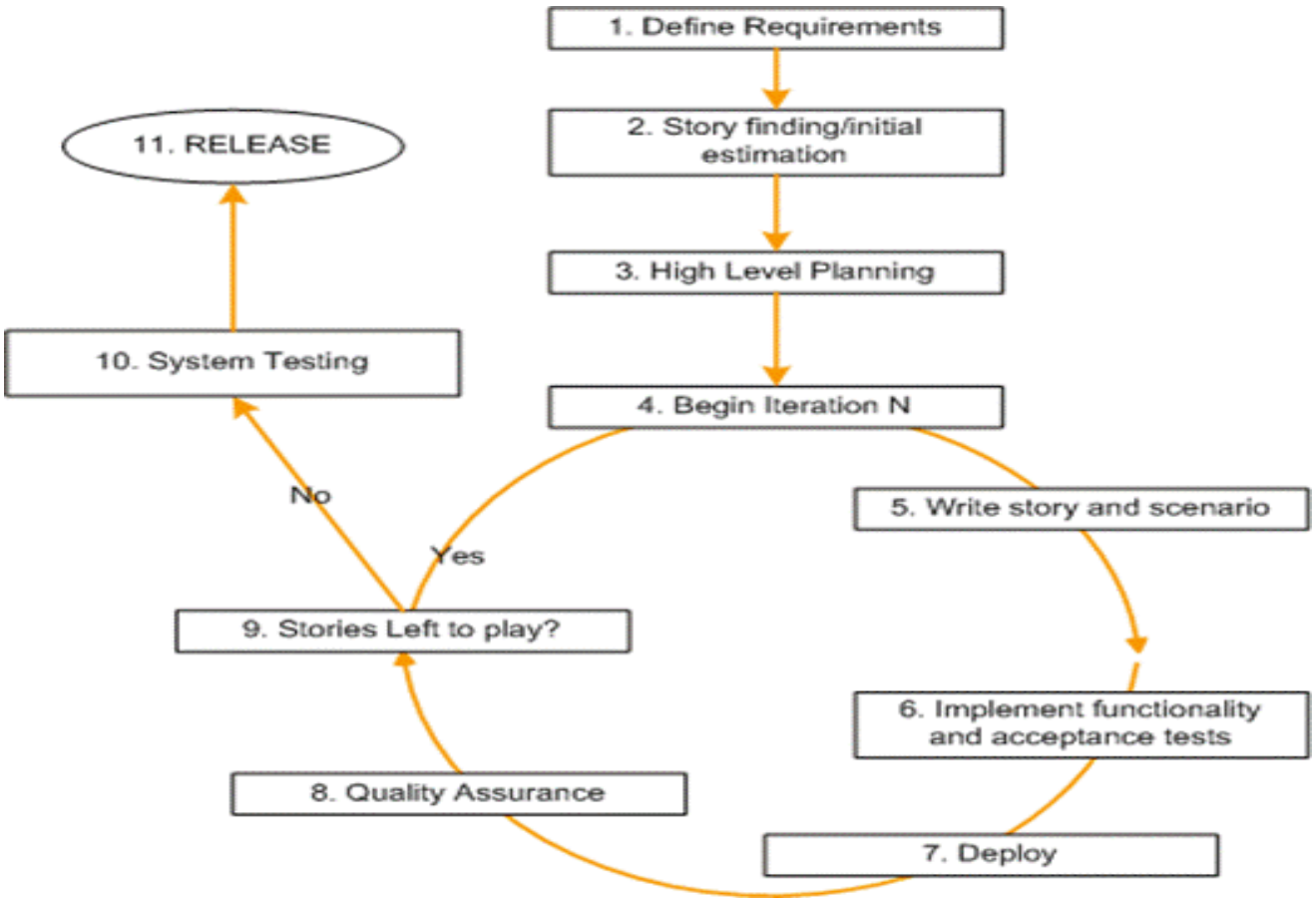
Relay Dataflow

Key Driving Requirement



DSN

Agile Development Process



From The Agile Release Process by Sam Newman

Acknowledging the development team

- Jesse Wright (architecture)
- Lloyd Deforrest (CAM)
- Dan Allard (developer)
- Marti Demore (developer, DSMS architecture)
- Rick Borgen (developer, db expert)
- Quentin Sun (SE)
- Kathy Sturdevant (CM, Test)
- Terry Himes (SE, developer)
- Brent Nash (developer)

- Mike Tankenson (SE)
- Dave Noble (consultant)
- Ron Dupitas (SA)