U.S. Chamber of Commerce Lunar Surface Systems Workshop

Software Architecture Study

Lauren J. Kessler
C.S. Draper Laboratory

Lunar Surface Systems

AGENDA

Agenda

- Overview
  - Proposal Summary
  - Study Approach
- Results
  - Requirements
  - Architecture
  - Feasibility
- Summary

Lunar Surface Systems

OVERVIEW

Overview

Proposal Summary

Solicitation:

- Propose alternative approaches for software architecture and development to result in
  - Robust software performance, reliability
  - Lower development and certification cost
  - Improved software reuse across lunar systems, and previous systems

Proposal:

- Explore the application of a multi-level autonomous architecture to the Lunar Surface Systems, using
  - Vetted autonomous technologies from various domain applications
  - Open architecture and software product line approaches

Establish a software technology approach that enables:

- Effective and malleable distribution of tasks
- A range of automation
- A reduced barrier to changes and additions of functionality
- Development of separable functional components by different vendors, within a common framework

Overview

Study Approach

Establish Requirements

- Major functionality for LSS operations
- Software design constraints

Develop Architecture

- Design low-level and system level conceptual software architecture(s)
  - "Software first" approach
  - Workflow and autonomy analysis

Determine Feasibility

- Evaluate software architecture(s) feasibility
  - Use cases & Trade studies
  - Figures of merit analysis

Lunar Surface Systems

STUDY RESULTS

Requirements Development
Philosophy

Concentrated on identifying requirements & constraints that software directly impacts (and vice versa)…

…based on the premise:

We should be able to advance the capabilities of a system in an *evolutionary* way

We shouldn't need a software *revolution* each time we want to increase our system's capabilities

Requirements Development
Approach

***Evolutionary* capabilities**

- Identified Lunar Outpost functional needs / capabilities
  - Culled information from Lunar Outpost documents, and existing analogous systems (e.g. ISS, McMurdo)
  - Separated the functionality into potential phases – Assembly, Operation, Maintenance
- Leveraged previous experience from space and other domains
  - Identified existing automation technologies
    - Space Applications, e.g. International Space Station
    - Commercial Applications, e.g. Cable Services provider
    - DoD, e.g. Unmanned Undersea Vehicles
  - Utilized best practices from software engineering
    - Software partitioning, open software, software product lines
- Focused on cost as a primary constraint
  - Reshape the cost curve of the software over its lifetime…
  - …while providing appropriate capability over the lifetime of the Lunar Surface System

Architecture
Vision

Establish a software technology approach that enables:

- Effective and malleable distribution of tasks
- A range of automation
- A reduced barrier to changes and additions of functionality
- Development of separable functional components by different vendors, within a common framework

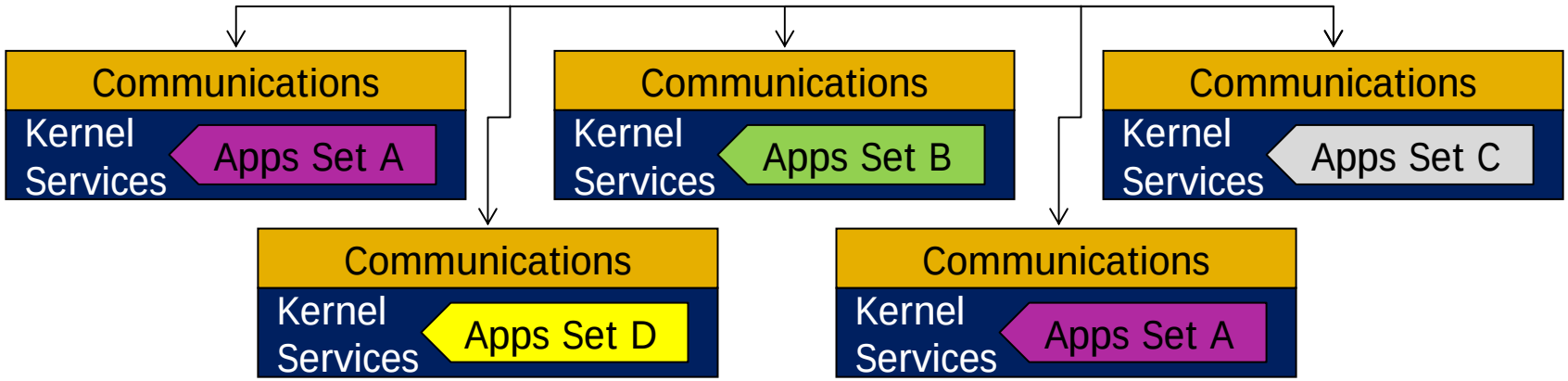
Key architectural design constructs:

- Federated (no central point of knowledge)
  - Common “kernel” or framework
- Fault tolerant (no system crash from a single fault)
  - Byzantine fault tolerant architecture

Architecture

High-Level View - Federated Kernel

- Core components that are repeated over and over
  - There are facilities for communications, application development, user interface and control
- Each software application is built using the core components
  - Configuration of each component, tailored to the operational requirements, is built using a combination of kernel services, and software applications



Functional Requirements

- Culled information from various sources:
  - Lunar Surface Systems documents
    - Surface Architecture Reference Document
    - Trade Set 1 (Surface Buildup Sequence)
    - Element documents
  - Analogous systems
    - ISS
    - McMurdo
    - Devon Island
  - Potential mission phases
    - Assembly
    - Operations
    - Maintenance
- Selected a single function as evaluation strawman
  - One that crossed as many surface elements and ops as possible: Power

	Lunar Surface System (from SARD v3.3)										
Subsystems & Functions	Lunar Lander	Power & Support Unit	Heavy-Lift Mobility System	Habitat	Crew Mobility System	EVA Systems	Power (Solar Arrays)	ISRU	Comm & Navigation	Science Packages	Logistics Elements
Structures & Mechanics	x	x	x	x	x	x	x	x	x	x	x
ECLSS	x			x		x					x
Thermal	x	x		x	x	x	x	x	x	x	x
Avionics	x	x	x	x	x	x	x	x	x	x	x
Power	x	x	x	x	x	x	x	x	x	x	x
Propulsion	x								x		
Communication	x	x	x	x	x	x	x	x	x	x	x
Logistics	x	x	x	x	x		x	x		x	x
Science			x	x	x			x		x	x
Crew Health	x			x		x					x
Training	x	x	x	x	x	x	x	x	x	x	x

Power	x	x	x	x	x	x	x	x	x	x	x

Software Constraints

To manage cost, the software should:

- Appropriately leverage human & computer capabilities
  - Variable as to distribution of capabilities over application and time
- Explicitly support the planned addition of new functionality
  - Non custom solutions for each functional element
- Be an integrated software system
  - Allows for a high degree of re-use
- Be extensible and modular
  - To provide an avenue for more efficient certification
- Open (at least in areas of extensibility)
  - To allow an avenue for multiple contractor participation
- Use standard tools and coding standards, plus tools to enforce coding standards
  - Note: these help, but do not guarantee, a simple, elegant solution
- Software should be built without assumptions about where it will be applied (or with those assumptions isolated)

Candidate Software Architecture Development Approach

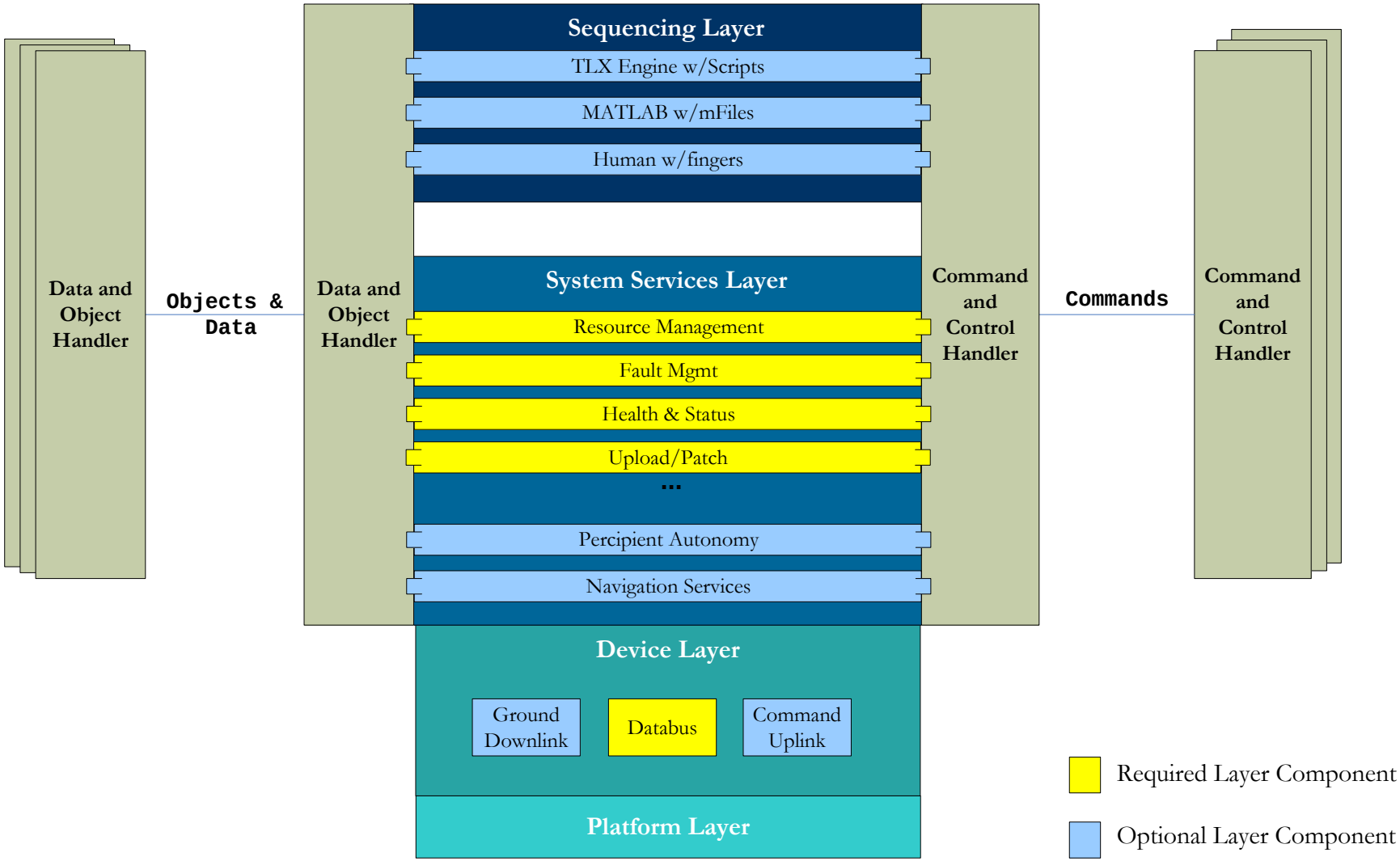
"Software first" design approach

- Performed a functional decomposition, with a deliberate software orientation
- Broke the traditional design methodologies cycle which often encourage stovepipe software development
  - Often, system functional decomposition results in a hardware oriented designing, leading to…
  - …software that is developed to service a specific piece of hardware performing a specific function
  - Software has increasing functional capabilities, that can be divorced from functional requirements specified by the hardware

Candidate Software Architecture Development

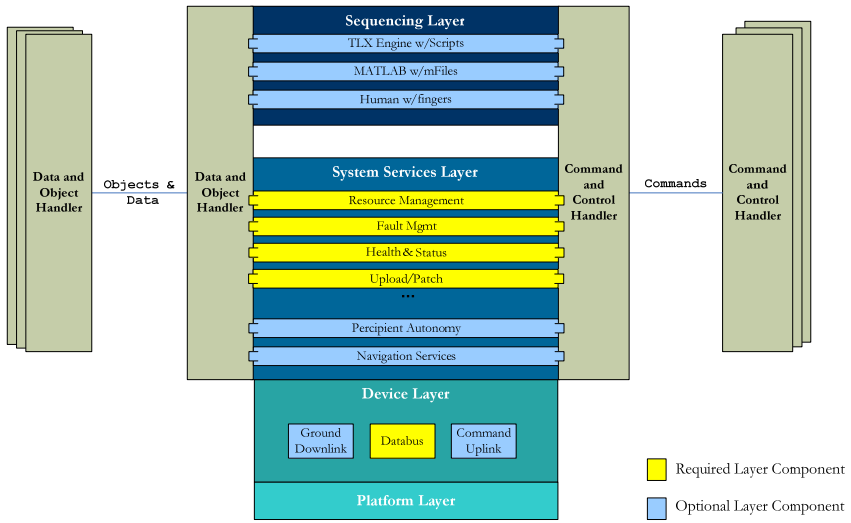
- Developed notional software architecture(s) that provide
  - Required LSS functionality
  - Fault tolerance
  - Malleable functionality – capability adjustments
- Considered software engineering practices
  - Business considerations (e.g. multiple contractors)
- Develop to the level of “node”, with the assumptions:
  - Minimal hardware inventory and software processing power to hold a node in our architecture
    - Not intended for a micro-controller
  - Communication ability between nodes

Architecture Component Diagram



Resulting Architectural Concept

- Federated, Open, Fault-Tolerant, Distributed, Autonomous Network (FOFTDAN)
  - Federated: no central point of knowledge (e.g. each entity can be both a server and a client)
  - Using common protocols
- Layered with common Middleware solution comprising of services that will handle the functions of the individual component.
  - Software should be common across components. We should build a common software infrastructure that everyone developing for LSS uses.
  - Architecture should be layered, in order to accommodate this
  - Note: this does cost more upfront, but it should payoff long term



Definitions

Autonomous	Ability of a space system to perform operations independent from any Earth-based systems. This includes no communication with, or real-time support from, mission control or other Earth systems.*
Automated	Automatic (as opposed to human) control of a system or operation.*

* NASA Procedural Requirements: Human-Rating Requirements for Space Systems, NPR 8705.2B, May 6, 2008

Levels of Automation and Decision Making

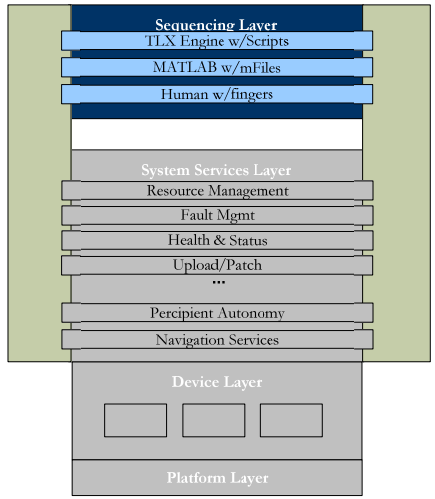
Levels of Automation of Decision and Action Selection									
1	2	3	4	5	6	7	8	9	10
The computer offers no assistance, human must make all decisions and actions	The computer offers a complete set of decision/ action alternatives	narrows the selection down to a few	suggests one alternative	executes that suggestion if the human approves	Allows the human a restricted time to veto before automatic execution	Executes automatically, then necessarily informs the human	Informs the human only if asked	Informs the human only if it, the computer, decides to	Computer decides everything, acts autonomously, ignoring the human

- An automated system may be designed for full or partial replacement of a function previously carried out by an operator.
- Automation is not all or none, but a continuum.
- Many human-in-the-loop supervisory control systems are designed for Level 2-4.

Parasuraman, R., T. B. Sheridan and C. D. Wickens (2000). "A Model for Types and Levels of Human Interaction with Automation." IEEE Transactions on Systems, Man, and Cybernetics 30(3): 286-297.

Sequencing Layer
Functional Description

- Provides high-level control for a particular component to
  - Mission control
  - Lunar outpost control
  - Automation services
- Contents within this layer would be either be modified or actuated differently from mission to mission, in order to drive component behavior
- Intent is to simplify, isolate, and plan for changes in the mission-level logic driving components
- Sequencing behavior could be accomplished via:
  - Human interface
  - Scripting language (e.g. Timeliner, SCL)
  - 3rd party tool (e.g. MATLAB)
  - Different sequencing layer implementations would be used, depending on the component application



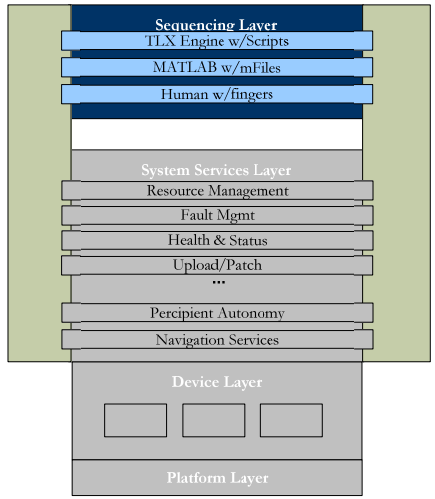
More Definitions

Scripted autonomy	• Scripted systems architectures include an execution engine, supplied by just-in-time human developed scripts • After the initial development of the execution engine are primarily data-driven, and therefore are lightweight and relatively simple to verify • Provides the human operator a significant level of control over the behavior of the system • However, such systems become increasingly complex to develop to be robust for highly dynamic environments
Percipient autonomy	• Provides built-in mechanisms for algorithmic adaptation for highly variable environments • When used on the systems for which they are best suited, there is a long term benefit from the comprehensiveness of such autonomous algorithms, and will allow the systems to operate in more situations than originally conceived

* NASA Procedural Requirements: Human-Rating Requirements for Space Systems, NPR 8705.2B, May 6, 2008

Sequencing Layer
Implementation: Timeliner

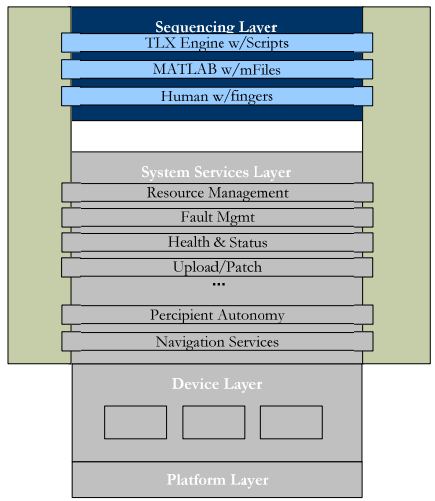
- Excellent fit for automating tasks for which there is an established manual procedure
- Scripts can be commanded without rebooting the execution engine
  - Install/Remove (allows “hot swapping” of new scripts)
  - Stop/Start
  - Jump to a key location
  - Pause and wait for operator confirmation
- Scripts are separated from each other, such that a failure of one script does not stop other scripts from executing
- Scripting language is English-like, making it readable by a broader audience than traditional code
- Any tool, including 3rd party tools, can be integrated into the sequencing layer by implementing interfaces to the Data Object and C2 handlers
- All interaction with the layers below is accomplished via these handlers



Sequencing Layer
Implementation: Human Interface

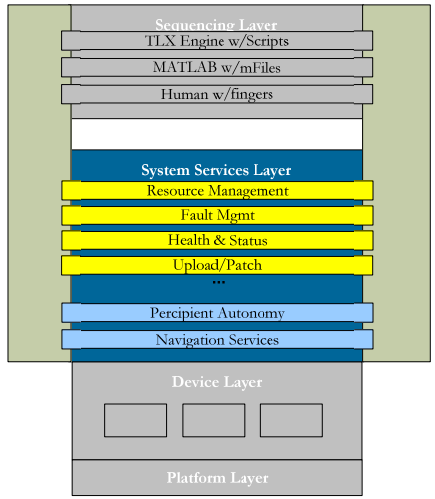
- Would be appropriate for components that are poorly suited to software-based control, or for which
  - Indirect human control would certainly always be available through commands to the scripting language or 3rd party toolBased on Model/View/Controller design pattern
- Implemented via
  - Remote or local display
  - A human input device
    - Button/Lever
    - GUI

…where direct human control is desired



Services Layer

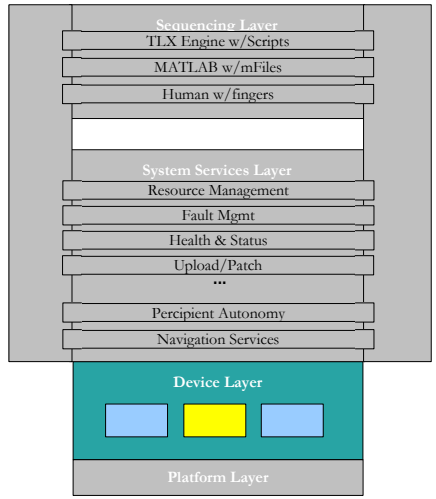
- Services layer provides standard software components from which to build each software application
- Some services would be common to all systems, for example
  - Resource Management – managing of system resources
  - Fault Management – standard fault approach
  - Health & Status – standard basic telemetry support
  - Upload/Patch – standard software maintenance
- Other services would only occur on systems that needed them
  - Navigation Services (vehicles only)
  - Percipient Autonomy (components that are good candidates for percipient control)

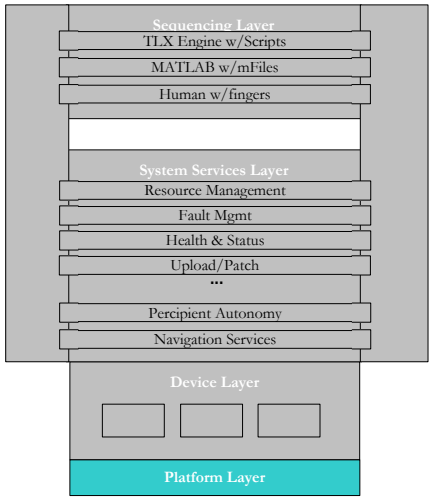


Device and Platform Layers

Device Layer:

- Provides drivers for communication with physical hardware devices
  - Ground Uplink/Downlink (if available)
  - Databus (required)
  - Other physical devices, for example
    - Solar panel controller
    - ECLSS
    - Vehicle controller





Platform Layer:

- Provides an interface to the operating system
  - Isolates OS specific functions
  - Provides portability
  - Standardizes which (and how) OS components are used

FOFTDAN Summary

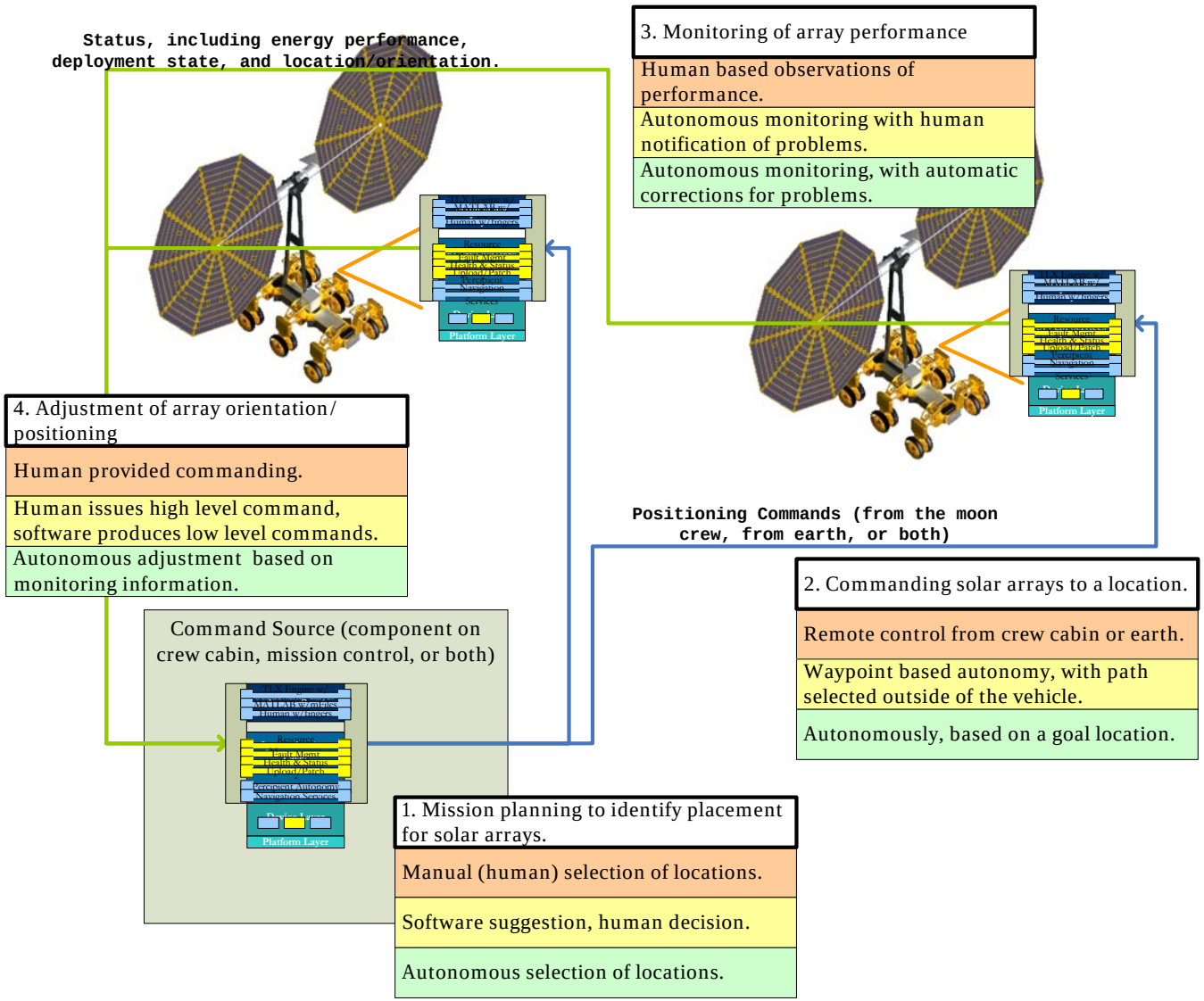
- What's different about FOFTDAN?
  - The sequencing layer provides
    - High-level personalization in terms of behavior
    - Configurable avenues of control
  - The services layer provides functional extensibility
  - The device layers allow you to customize the hardware
  - A fault tolerance approach, based on either hardware or software can integrate safety *into* the architecture
    - As a standard feature
    - Not depend entirely on design-assurance methods for safety
  - Capitalizes on distributed, federated approach to enhance existing fault tolerant methods

Feasibility

Use Cases

- We examined the design applicability of FOFTDAN, and resultant levels of automation against projected LSS elements, e.g. the Solar Array
- Selected the solar array LSS element due to the following features:
  - Has associated activities in the assembly, operations and maintenance phases;
  - Is an essential element of a cross-cutting function, namely the production of power;
  - The required functions of the array has the potential for varying levels of automation;
  - Can be easily isolated, for the purpose of this examination, from other LSS elements.

LSS Example


Status, including energy performance, deployment state, and location/orientation.
3. Monitoring of array performance
Human based observations of performance.
Autonomous monitoring with human notification of problems.
Autonomous monitoring, with automatic corrections for problems.
4. Adjustment of array orientation/ positioning
Human provided commanding.
Human issues high level command, software produces low level commands.
Autonomous adjustment based on monitoring information.
Positioning Commands (from the moon crew, from earth, or both)
2. Commanding solar arrays to a location.
Remote control from crew cabin or earth.
Waypoint based autonomy, with path selected outside of the vehicle.
Autonomously, based on a goal location.
Command Source (component on crew cabin, mission control, or both)
1. Mission planning to identify placement for solar arrays.
Manual (human) selection of locations.
Software suggestion, human decision.
Autonomous selection of locations.

Existing Technologies

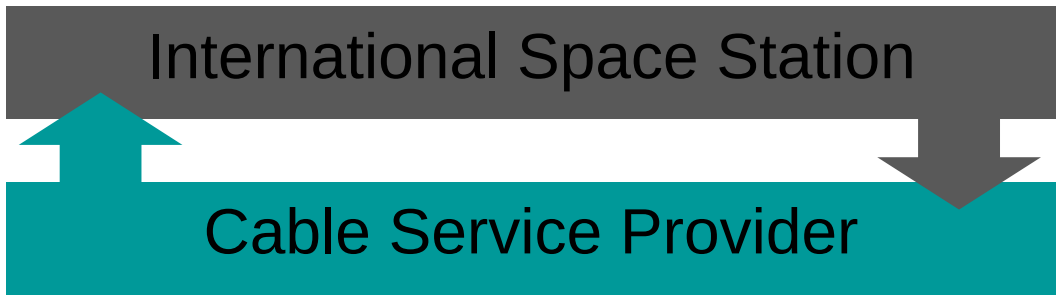
- Architectures such as FOFTDAN are feasible because they strategically re-use concepts, techniques, and technologies from existing architectures, embodying
  - Open architectures
  - Software re-use
  - Distributed computing
  - Federated and expandable systems

- Many of the lower-level technologies have been vetted in the commercial world, including real-time operating systems, device drivers, etc.
- Automation systems as well are not pie-in-the-sky technologies, e.g. ISS, cable servicing, etc.

Industry Effects

- What effect does this have on industry?
  - A demonstrated feedback mechanism has proven useful to advances in space and on Earth.

Automation Technologies



- On the ISS, this technology enabled astronauts and ground control to focus on other, more cognitively intensive tasks
- On Earth, this technology streamlined the call-center operations for diagnosing existing problems, and improved customer satisfaction

Lunar Surface Systems

SUMMARY

Summary

- Enabling software architectures, embodying the critical concepts of…
  - Extensibility and modularity
  - Controllable growth
  - Composable certification
  - Software product lines (e.g. core assets such as a kernel)
- …will be a fundamental source in accreting functionality for the LSS over time
  - Changes the workload of the human
    - Machine performs rule-based tasks
    - Human performs knowledge-based tasks
  - Reduces the need for a large Earth-based operations team
- …will be a fundamental source of new technologies for Earth based commercial operations

Team

- John West (Program Management)
- Lauren J. Kessler (PI)
- Emily Braunstein
- Stephen Duncan
- Kevin Duda
- Mark Lyon
- Michael Ricard

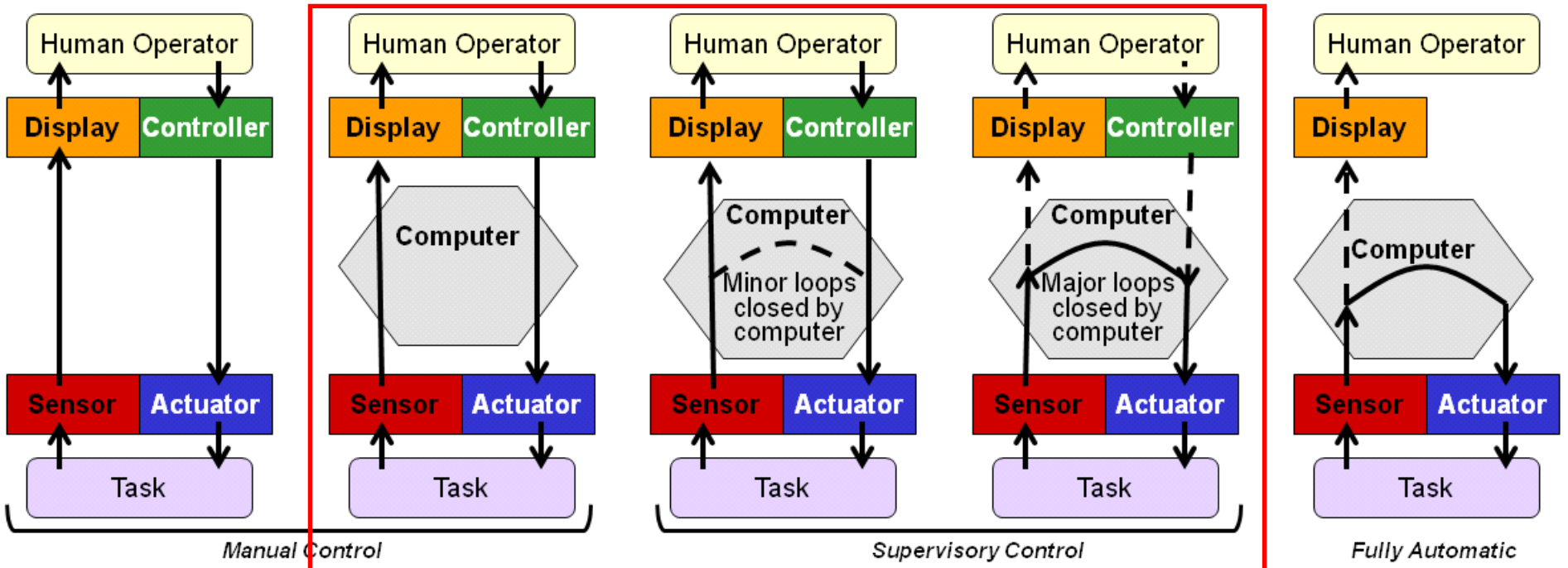
Lunar Surface Systems

QUESTIONS?

Lunar Surface Systems

ADDITIONAL MATERIALS

Multi-Loop Model of Supervisory Control



Multi-Loop Model of Supervisory Control

1	2	3	4	5
Task is observed directly by human operator's own senses.	Task is observed indirectly through sensors, computers and displays.	Limited number tasks are controlled by the computer's automatic mode.	Majority of the tasks are controlled by the computer's automatic mode.	Task is controlled completely within the computer's automatic mode .
Human operator directly affects task by manipulation.	Human operator indirectly affects task through controls, computers, and actuators.	Human operator gets feedback from the computer. A majority of tasks can be affected indirectly.	Human operator gets feedback from the computer. A limited set of tasks can be affected indirectly.	Human operator observes display. Has no ability to interact with the task.

Sheridan, T. B. (1984). "Supervisory Control of Remote Manipulators, Vehicles and Dynamic Processes." Advances in Man-Machine Systems research. W. B. Rouse (Ed.). New York, JAI Press.

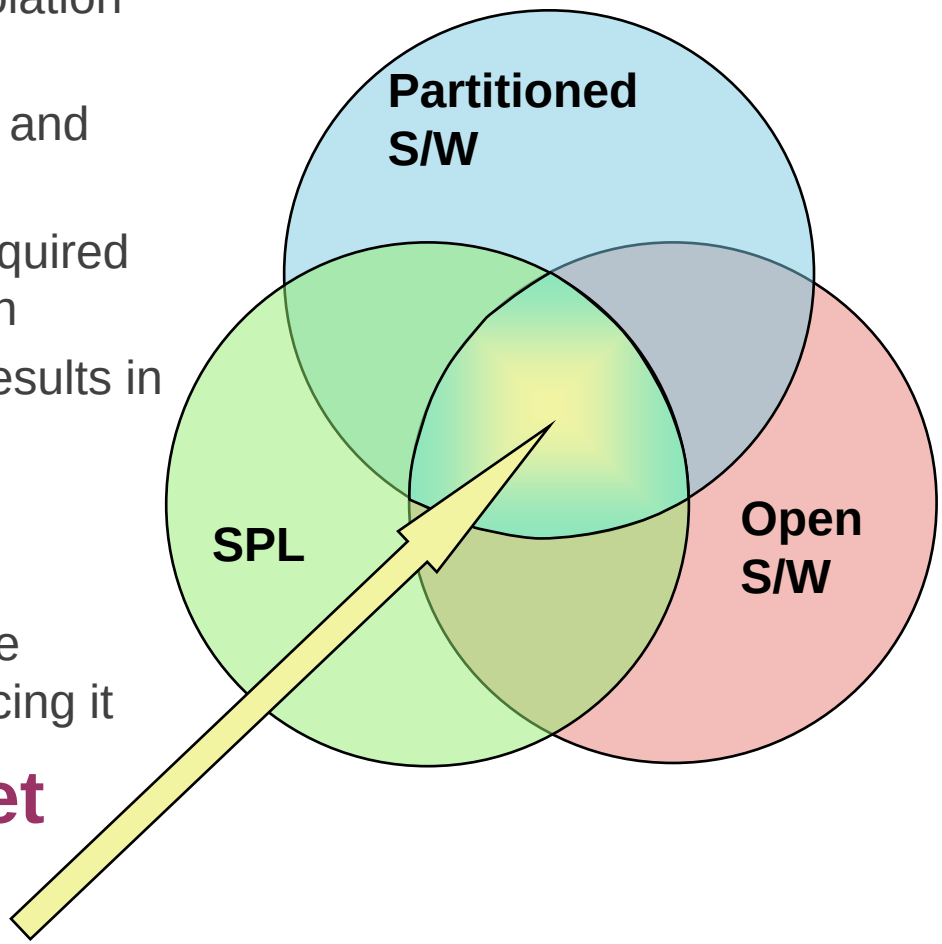
Software Approaches & Technologies

- **Concentrate on the strategic application of established software development techniques**
  - ***Software Partitioning***
    - Decomposition of the software for each vehicle subsystem into modules in order to break down the problem in a systematic way
    - There are many existing, well established techniques for partitioning that allow us to address the problem incrementally (development, testing…)
  - ***Open Software***
    - Originally designed, developed and documented to be usable by contractors other than the initial implementer to:
      - Study, change, and improve the software
      - Reuse it in modified or unmodified form to accommodate new system development
    - Open software can be secure software
  - ***Software Product Line***
    - A set of software-based subsystems that share common features and are developed in a prescribed way
    - Can be used in multiple applications as is, or tailored

How these concepts interrelate

Application

- Applying the techniques in isolation
  - Goals can be reached (*adaptable, maintainable,* and *cost effective systems)*
  - More rigor and effort is required than strategic combination
- Combining them coherently results in
  - Development efficiency
  - Encourages continuing competition
  - Focuses on enhancing the system, rather than replacing it



Example: International Space Station

International Space Station

- Timeliner Executor is on-orbit and operational aboard both the Payload and C&C MDMs
- Timeliner is also being used to automate ISS Core operations
  - Reconfiguration of C&W event tables upon C&C MDM switch
  - Upgrade of DCSU power controller firmware
  - Other procedures are in the “pipeline”
    - S-Band operations, TCS reconfiguration, HCOR reset, …
  - MOD concept for “lights-out” control center operations using Timeliner

Benefits of using Timeliner

- Reduction in ground operations workload
- Reduction in need for console positions

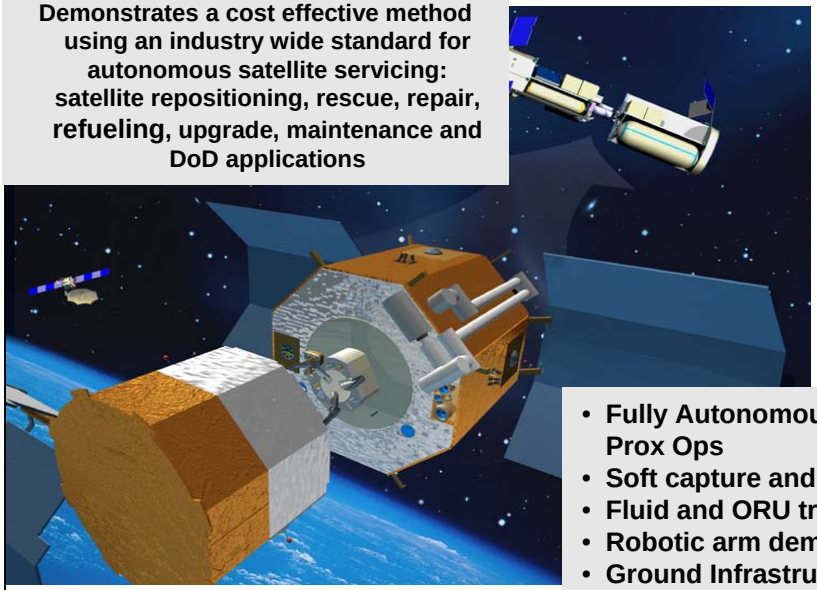
Example: International Space Station

HAL System

- Automates payload monitoring and commanding that would otherwise be conducted by ground personnel
  - Occurs even when there is no ground communication
  - “The HAL system was critical in a recent consolidation of ground controllers; it basically automated a complete ground position”
- Capability was built up through time through increasingly complex scripting logic
- Layered Timeliner Approach
  - “Master Bundle” provides high-level payload management
  - Payload specific bundles provide lower level monitoring and control
  - HAL Master bundle continuously monitors payload device power status and automatically executes startup/shutdown sequences in response to change in status

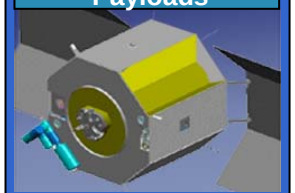
Example: Orbital Express Program

Demonstrates a cost effective method using an industry wide standard for autonomous satellite servicing: satellite repositioning, rescue, repair, refueling, upgrade, maintenance and DoD applications



- **Fully Autonomous Rendezvous and Prox Ops**
- **Soft capture and mating**
- **Fluid and ORU transfers**
- **Robotic arm demonstrations**
- **Ground Infrastructure**

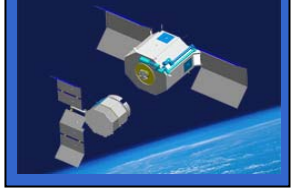
Auxiliary Payloads



Launch Opportunities

Atlas V
March07

Demonstration Enhancements



Operations with Microsats

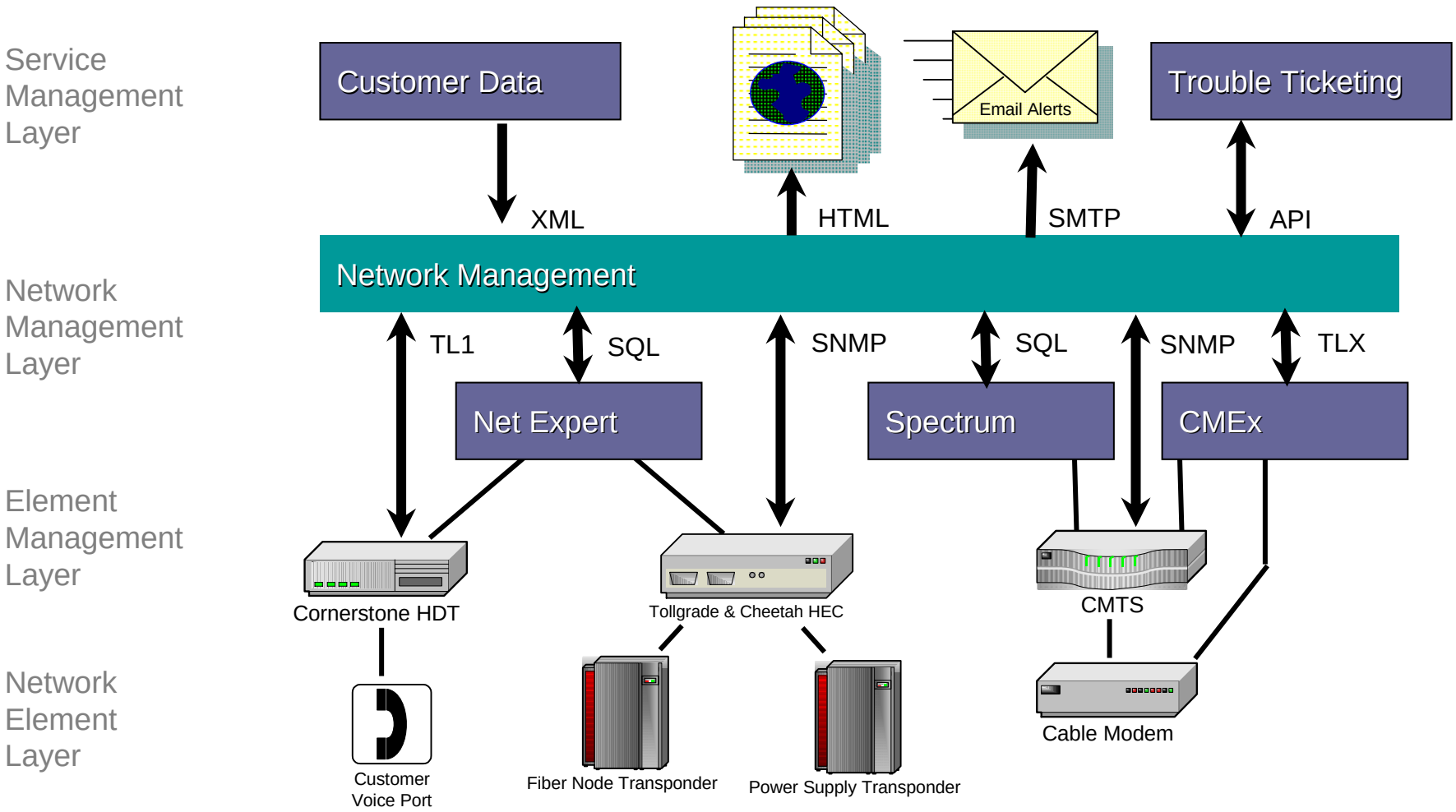
Program Objectives:

- **Design and build the ASTRO (servicing vehicle) and NEXTSat (Next Generation Satellite) vehicles to successfully demonstrate autonomous servicing**
- **Continue to refine conceptual operational missions**

Example: Orbital Express

- Several Timeliner features were highly beneficial for Orbital Express Mission Manager:
- Ability to easily upload new scripts, without rebooting
  - Allowed the mission logic to be modified in a contained manner
- Authorization-to-Proceed (ATP) mechanism required human authorization at key points
  - System was configurable to adjust the ATP level
  - Initially the level was set such that a high degree of human confirmation was needed
  - Operators modified the level of software autonomy by adjusting the ATP level
    - Early scenarios were run slowly with many ATP points
    - Later scenarios were run much more autonomously, as confidence was gained in the system capabilities
- Ground based commanding allowed for slight modifications of script behavior without uploading new scripts
  - Ground based pause/jump were useful for small logic changes, as a workaround to hardware anomalies

Example: Cable Modem Management



TLX – Key Architectural Features

- Provides built-in support for third party interfaces (via TIS/TIL architecture)
- Provides architecture to chain together multiple TLX engines that are working on the same problem
  - Auspice applications of this include
    - Multiple TLX engines working on the same task, coordinated via a SQL database work queue (provides increased data processing workflow)
    - Multiple TLX engines with different functional roles in an architecture
      - Addressing I/O bottlenecks via dedicated engines
      - TEAM architecture, which separates event detection, management, and handling into different TLX engines
    - Primary/Backup engines
    - Web interface for monitoring engines

The TLX Platform Architecture