Architectural Aspects of Long-Lived Ground Systems

Charles ("Bud") Hammons

Ground Systems Architecture Workshop 2006

Sponsored by the U.S. Department of Defense
© 2006 by Carnegie Mellon University
Topics

Ground Systems Challenges

A motivating example - TSAT*

Architecture Strategy

Architecture Tactics

Realization

Summary / Q&A

*Disclaimer: personal views, not necessarily those of the Transformational Satellite Communications (TSAT) PMO
Ground Systems Challenges

• Unprecedented Operational Capability

• Interoperability with external systems also in development

• Interoperability with Legacy Systems

• Evolution in CONOPS

• Evolution in protocols and underlying technology

• Architecturally significant attributes
• Drive lifecycle evolution/change into development cycle
A Motivating Example - TSAT

Goals include

- mission-critical satellite-based packet and circuit communications for the warfighter
- quality of service, info assurance, comm. on the move,…
- seamless integration into the Global Information Grid (GIG)
- complex interactions with military planners/systems

Other programs have similarly challenging objectives and complexity (e.g. business enterprise integration exploiting RFID*, network communications,…)

Overarching Challenge – develop a large, complex, long-lived, software intensive systems in an environment that is fluid both during and after development

*RFID – Radio Frequency IDentification
Architecture Strategy

At the risk of stating the obvious, identify what is fixed, what is variable

**Fixed/Slow-moving**
- domain-specific data
- essential behavior
- software/hardware split

**Variable/Evolving**
- standards, protocols
- external interfaces
- CONOPS, deployment
- time constraints
- value-added features
- technology refresh
- human-machine task split

**Tactics:** identify architectural features that allow change and protect invariants
Architecture – Tactics

Separation of Concerns

Explicit domain-specific data model
- most resilient piece of large system-of-systems
- desirable to version elements
- unambiguous units of measure
- include behavior with roles, permissions, etc.

Separate CONOPS from data model
- CONOPS is mechanized as an explicit element of architecture
- captures policies that drive behavior
- describes human-machine task division

Separate domain-specific behavior from supporting infrastructure
Define Capable Infrastructure

Generalized inter-component communications
• messaging ‘middleware’
• asynchronous to near real-time constraints
  - multiple transport mechanisms transparent to application components

Explicit management model for components
• formal model for control and monitoring
• ‘component registry’
• include version as lookup criteria
• enable automated & remote component

Isolate external interfaces from applications/services
Architecture – Tactics

Exploit Legacy & COTS Software

• Treated as components in architectural model

• Individual choices should neither “break” nor drive architecture

• Unique structure hidden by common packaging conventions

• On case-by-case basis, revision/replacement is pre-planned
Realization

Architectural Styles
- Client-Server
- Service-Oriented Architecture (SOA)
- Agent-based systems
- Hybrids

Communications Models
- XML-based (including “Web Services”)
- CORBA and relatives
- Problem-specific binary communications protocols (e.g. WSTAWG* real-time model)

*WSTAWG – Weapons System Technical Architecture Working Group
Realization 2

Organizational Issues

• Recognize going in that this is difficult work
• Requires organizational buy-in and sustained management attention
• Expect numerous objections
• Complexity and long time frame ensures mistakes will happen – architecture can mitigate effects when domain mutates or market forces influence what is available or appropriate
Summary

• Developing complex net-centric systems while we are still trying to fully understand what it means to be net-centric represents unique opportunities and risks.

• Rapid evolution in technology, standards, and protocols increases variability that programs must comprehend.

• Architecture can mitigate some of the difficulties.

• There is still no silver bullet.