F6 Model-driven Development Kit (F6MDK)

Gabor Karsai, Abhishek Dubey, Andy Gokhale, William R. Otte,
Csanad Szabo; Vanderbilt University/ISIS
Alessandro Coglio, Eric Smith; Kestrel Institute
Prasanta Bose; Lockheed-Martin Space Systems
Future Fast, Flexible, Fractionated, Free-Flying Spacecraft (F6)

Objective: Develop and demonstrate a satellite architecture where the functionality of a traditional monolithic spacecraft is replaced by a cluster of wirelessly connected modules

Advantages:
- Increased flexibility during design and acquisition
- Reduced development and launch costs
- Increased adaptability and survivability of space systems on-orbit
- Potential to apply economies of scale to satellite design & manufacture

Key program objective is the promulgation of open interface standards for hardware and software.
Challenges

1. **Distributed system with network addressability**
   - Everything and anything (modulo security permissions) can be accessed and addressed

2. **Dynamism**
   - Dynamically deployed applications, security configurations, and cluster architectures

3. **Resource sharing**
   - Specific resources can be shared across applications: CPU, communication links, memory, services

4. **Fault tolerance**
   - Faults in components, services, communication links, computing nodes are detected, isolated, and their effects mitigated

5. **Multi-level security**
   - The architecture shall address the requirements of MLS
Solution:
F6MDA (Model-driven Architecture)

Layered architecture supported by a model-driven development toolchain
Solution: F6OS

- The ‘Operating System’ that provides
  - Restricted OS calls for application actors
  - Privileged calls for platform (‘service’) actors
  - All system calls are time-bounded

- Provides messaging services
  - All component interactions are via messages
  - No other interactions are possible

- All component interactions are facilitated by a ‘secure transport’ that verifies security labels on messages

- Resource management functions
  - CPU time: temporal partitioning for actors, utilization cap per actor within partition
  - Memory: space partitioning, limit caps
  - Network bandwidth: diffserv, bandwidth budget, differentiated routing

- F6OS is part of the TCB
Solution: Middleware

- The ‘middleware layer’ that provides
  - Synchronous and asynchronous point-to-point communication with call/response semantics (→ Subset of CORBA RMI)
    - Location transparency
    - Request (de)multiplexing
    - Message (de)marshalling
    - Error handling
    - Support for QoS (client timeouts, reliable one-ways)
  - Anonymous publish/subscribe communications with one/many-to-many data distribution patterns (→ Subset of DDS)
    - Datatype specification
    - Static discovery
    - Selected QoS: reliability, time-based filtering, latency budget, etc.
Solution: F6 Component Model

Applications are constructed from interacting components.

Components interact and exchange data via ports.

Publishers are decoupled from subscribers; publishers can send data at anytime, subscribers are triggered when data is available/can poll data.

Provided and required interfaces are connected and support tightly and loosely coupled, synchronous interactions with call/response semantics.

Component state may be exposed (accessible to the middleware) for fault tolerance.

Components request resources (memory) only from and through the middleware.

Components are triggered (scheduled and released) by the middleware – single-threaded.

Components have >=1 ports.

Applications ➔ Component assemblies.

Actors ➔ Groups of >=1 components.
Solution: Platform Actors

- The ‘platform services’ that manage
  - Operations
    - Autonomous and commanded
  - Application deployment
    - Actors, components, connections, resource limits, labels
  - Dictionary
    - Addressing of all ‘objects’
    - Data topics
  - Certificates
  - Faults
    - Application restarts, redeployment
  - Communication resource
    - Wireless Inter-Module Communications, including bandwidth and routes
Solution: Fault Management

- The overall layered approach to manage faults on various levels of the system:
  - **Localized:**
    - Anomaly detection → diagnostics → mitigation:
      - Network HW, CPU HW, F6OS, Component/Actor
    - If mitigation fails, report ‘up’
  - **Global: (Fault Manager)**
    - System-wide diagnostics and mitigation
    - Fault-tolerant, replicated, hierarchical
    - Autonomous recovery from faults:
      - Failover and reconfiguration
Solution: Development Environment

- **Model-driven development**
  - Software architecture models capture:
    - Component interfaces, ports, assemblies, deployment
    - Anticipated resource needs (budgets)
  - Middleware glue code and configuration artifacts are automatically generated from models

- **Model-driven system integration**
  - Integration models capture:
    - System-wide data type definitions
    - Complete system architecture (all apps)
    - All security labels and lattices

- **Conventional development process can also be used**
  - System integration requires configuration artifacts
Current Status

- Currently three months into the implementation phase
- F6OS prototype is being implemented using a modified Linux kernel, to be eventually replaced with a high assurance microkernel
- F6ORB and F6COM are being implemented using subsets of the open-source ACE/TAO/CIAO CORBA Component Model (CCM) implementation and OpenDDS

Projected timeline:
- Completed implementation: End of 2012
- Verification and Validation: End of 2013
- Ground-based testing: End of 2014
- On-orbit demo mission: 2015