# A MODEL-DRIVEN SOFTWARE COMPONENT FRAMEWORK FOR FRACTIONATED SPACECRAFT

**Abhishek Dubey**[(1)], **Aniruddha Gokhale**[(1)], **Gabor Karsai**[(1)] , **William R. Otte**[(1)], **Johnny Willemsen**[(2)]

[(1)]*Institute for Software-Integrated Systems, Vanderbilt University, 1025 16th Avenue South, Nashville, TN 37212, USA, +1(615)343-7472,{dabhishe,gokhale,gabor,wotte}@isis.vanderbilt.edu*

[(2)]*Remedy IT, 2650 AC Berkel en Rodenrijs, The Netherlands, jwillemsen@remedy.nl*

***Keywords:*** *Space Computing, software components*

## ABSTRACT

Fractionated spacecraft operated as a re-purposable multi-application platform for varying missions poses a number of challenges to software developers: in addition to providing specific functions (like sensor data processing) the software (1) has to manage scarce computational and communication resources, (2) has to provide a framework for managing faults (both in the hardware and the software), (3) has to provide the required quality of service (QoS) to components implementing the services, and (4) has to provide the proper security isolation between users to ensure the confidentiality of information flows. Arguably, such challenges can be addressed by a platform-based architecture where a reusable and highly configurable software platform provides services to varying software applications that implement specific functions. The F6 program (supported by DARPA) is developing such an Information Architecture Platform (IAP) that aims to be a generic, reusable, and open-source software infrastructure for building fractionated space systems. The F6 hardware platform consists of two or more satellite modules that communicate via wireless links, and at least one module has (mostly) continuous ground connectivity.

The F6 IAP incorporates a suite of state-of-the-art software technologies that include both design-time and run-time elements. The essential architectural principle for F6 IAP software applications is that they consist of actors constructed from reusable software components. An actor is a unit, similar to processes in other operating systems: it has a guaranteed share of the CPU resource, has access to the communication resources with provisioned QoS, is a unit for fault containment and management, and has security labels for all information flows it participates in. Actors of an application can be distributed across the modules, or be co-located on one module.

Components that make up an actor are reusable units of software code with well-defined execution and interaction semantics. The supported interaction mechanisms include both point-to-point (synchronous and asynchronous remote method invocations) and many-to-many publish/subscribe communications. Work performed by a component is broken up into individual non-preemptible but time-bounded operations that are triggered either by interaction-related events or by the expiration of timers. The scheduling of component operations can follow different paradigms: first-in-first-out with or without priority, and earliest deadline first. By design, component operations do not block each other and prevent deadlocks and race conditions.

While the core abstractions for application writers are the actor and components, they are constructed using a middleware layer that is built upon an operating system

layer. The middleware layer implements the generic functions of the software component framework (patterned after the DDS4CCM standard) and the support for component interactions (publish/subscribe based on DDS, and synchronous and asynchronous remote method invocation based on CORBA). The operating system layer of the F6IAP, implements low-level services typical of operating systems.

Although the F6IAP describes the actor and their components and the OS, developing, deploying and configuring the actors and their components is fraught with both accidental and inherent complexities. The accidental complexities arise from the tedious and error-prone nature of the process used to (a) compose components to form actors, (b) deploy the actors in the nodes of spacecraft modules, and (c) configure the actors for their QoS properties. The inherent challenges stem from multiple sources. First, F6OS does not provide any notion of an operating shell, which means that command-line based deployment and configuration of applications is not feasible. Second, intermittent and highly fluctuating network connectivity implies that ground-controlled orchestration of applications on the fractionated spacecraft is not an option. Third, scheduling and admission control of actors on the F6OS is based on runtime availability of resources and security policies. Owing to such factors, significant autonomy is desired in handling the lifecycle of actors and their components. Manual approach is neither feasible nor would it provide the stringent assurances on correctness properties for the mission critical applications.

Designing and constructing distributed applications for such a complex and powerful software platform is therefore extremely challenging: there are many accidental and inherent complexities that developers have to cope with. These challenges can be addressed by a model-driven software development environment (MDE) that application developers can use to build components and then compose them to architect and configure applications. The MDE is based on a domain-specific modeling language that allows (1) modeling of components, actors, and applications together with their interfaces and interactions, (2) how the applications are to be deployed on the platform, (3) security labels and their association to component information flows, (4) QoS properties and resource needs of the application and its components, and (5) hardware platform architecture and configuration. Note that component implementation is not modeled – this can be implemented in C++ or in another model-based tool (like Simulink/Real-time Workshop) that is capable of producing executable code. Configuration files and glue code are generated from the domain-specific models, and the code is compiled and linked to implementation code to form binaries for components, which are then deployed on the platform using the mechanisms provided. The models also permit design-time analysis: system integrators can make admittance decisions based on the models. Such analysis is needed to decide if the resource needs of the applications can be satisfied on the space-based platform.

Currently a reference implementation of the F6 IAP is being created, based on several open source packages: Linux (for F6OS), ACE/TAO and OpenDDS (for the middleware), and GME (for the modeling environment). This paper outlines the capabilities of the F6IAP software component framework and its supporting model-driven development environment and it illustrates how it could be applied in a variety of software development tasks for various missions.