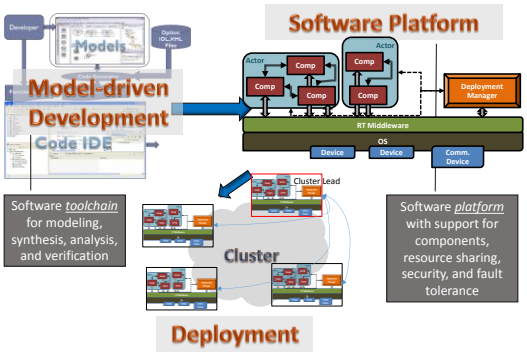
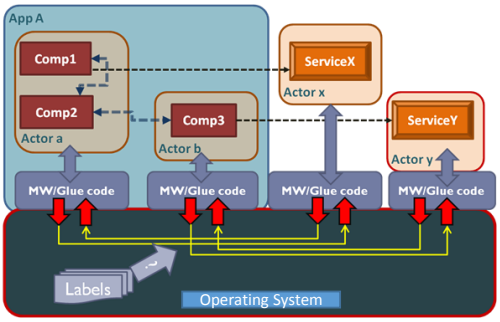
DREMS

A Software Infrastructure for Distributed Real-Time Managed Systems

Realizing the full potential of distributed embedded systems such as fractionated satellites necessitates a **software application platform** that supports secure and fault-tolerant sharing of resources: processors, storage, communication links, and devices. The system must enable on-demand secure collaboration between applications operated by different organizations. Clearly, the economic viability of the system depends on the rapid assembly of reliable distributed application from reusable software components, including those sourced from various vendors.

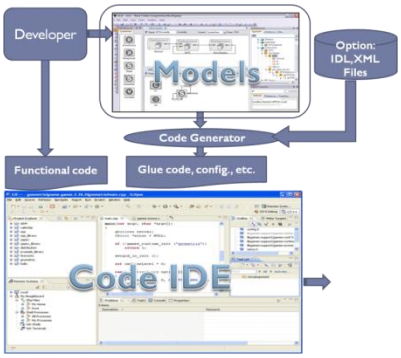


DREMS is a runtime infrastructure and a related toolsuite that facilitates a model-based paradigm of software development for distributed, real-time, embedded systems where *modeling tools and generators* automate the tedious parts of software development and also provide a design-time framework for the analysis of software systems. The *run-time software platform* reduces the complexity and increases the reliability of software applications by providing reusable technological building blocks in the form of an operating system, middleware, and application management services.



The platform includes a novel operating system that supports mixed-criticality scheduling with three levels: (a) critical system tasks, (b) application tasks, and (c) best-effort tasks. Temporal partitioning among application tasks and a new, security-labeled communication mechanism called Secure Transport provide additional fault- and security isolation between untrusted applications. Operating system entry points are protected by data integrity checks that are automatically generated from formal specifications.

Applications are built from *software components* that interact via *only* well-defined interaction patterns using security-labeled messages, and are allowed to use a permitted set of low-level services provided by the operating system. The low-level services include messaging and thread synchronization primitives, but components use these indirectly: via the middleware layer that facilitate the high-level interactions. Interaction patterns include (1) point-to-point interactions (in the form of synchronous and asynchronous remote method invocations), and (2) group communications (in the form of asynchronous publish-subscribe interactions). Component operations can be event-driven or time-triggered, enabling time-driven applications. All messages sent via the Secure Transport are time-stamped, thus message receivers are aware of when the message was sent. Hence temporal ordering of events can be established (assuming the clocks of the computing nodes are synchronized).

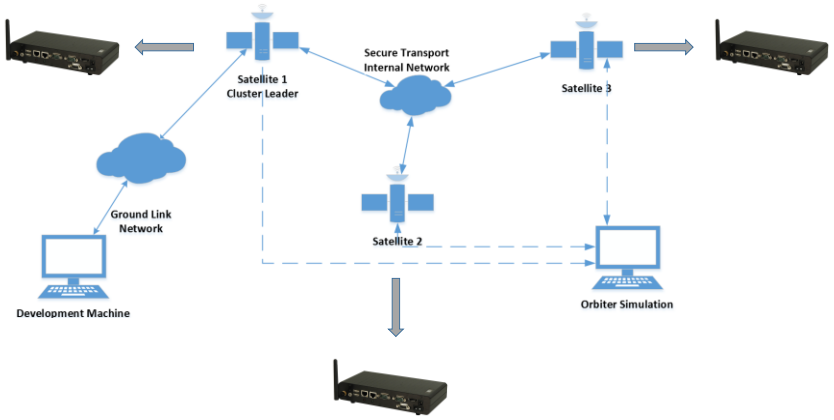


Configuring the middleware and writing code that takes advantage of the component framework is a highly non-trivial and tedious task. To mitigate this problem and to enable programmer productivity a model-driven development environment is available that simplifies the tasks of the application developers and system integrators. The toolsuite can be used in conjunction with existing IDE-s, including Eclipse.

[1] This work was supported by the DARPA through NASA ARC. See http://www.isis.vanderbilt.edu/drems for more information.
Contact: Gabor Karsai, gabor.karsai@vanderbilt.edu, (615) 343-7472, ISIS, PO Box 1829B, Vanderbilt University, Nashville, TN 37235.

RTSS@Work Demo

3 Satellite Cluster Running Image Processing and Cluster Flight Software



To demonstrate the utility of DREMS for resource constrained systems running mixed-criticality applications, we configured a cluster of three embedded computing nodes connected to Orbiter Space Flight Simulator[1] which simulates the orbit physics and satellite hardware. Each satellite runs two applications: (1) the system-critical Cluster Flight Application (CFA), and (2) an Image Processing Application (IPA). The CFA consists of four actors which perform orbit maintenance and respond to emergency commands from a ground station. The three CFA actors involved with emergency response run as critical tasks, while the periodic orbit maintenance CFA actor runs in a temporal partition. Orbit maintenance periodically polls the satellite sensors and disseminates the new state to the rest of the cluster. Responding to emergency commands from the ground station is critical, i.e. when a *scatter* command is received by the cluster leader, all satellites should activate their thrusters simultaneously and maneuver away from each other. The IPA consists of four actors, each running image processing algorithms to consume as much of the CPU as possible. The four IPA actors are split evenly between two temporal partitions. These two applications were developed with the platform modeling tools, which required only 500 lines of code to be written between the two applications, comprising less than 0.1% of the applications' code.

The cluster communicates over a private gigabit subnet, to which is connected a traffic shaping node (TSN). All application network traffic must go through the TSN, which runs dummynet[2] for full control over the bandwidth, delay, and packet loss to simulate the cluster's satellite network. The nodes also support NTP with synchronization on the order of 10 µs. This synchronization allows accurate measurement of emergency response latency.

These demo applications show the performance of mixed-criticality real-time tasks on the system. The figure below shows the latency from the cluster leader's reception of the scatter command to each satellite's thruster activation. The scenarios demonstrate the invariance in the latency in response to changing application load and partition scheduling.

(1) http://orbit.medphys.ucl.ac.uk/

(2) http://info.iet.unipi.it/~luigi/dummynet/