DREMS: A Toolchain and Platform for the Rapid Application Development, Integration, and Deployment of Managed Distributed Real-time Embedded Systems

William Emfinger, Pranav Kumar, Abhishek Dubey, William Otte, Aniruddha Gokhale and Gabor Karsai
ISIS, Dept of EECS, Vanderbilt University, Nashville, TN 37235, USA

The DREMS[1] toolsuite is a software infrastructure for designing, implementing, configuring, deploying, operating, and managing distributed real-time embedded systems. It consists of two major subsystems: (1) a design-time environment for modeling, analysis, synthesis, implementation, debugging, testing, and maintenance of application software built from reusable components, and (2) a run-time software platform for deploying, managing, and operating application software on a network of computing nodes. The platform is tailored towards a managed network of computers and distributed software applications running on that network: such as a cluster of networked nodes such as fractionated satellites or a group of smartphones deployed in a coordinated fashion to provide ad-hoc distributed services that can be used in disaster relief.

It is a complete, end-to-end solution for software development: from modeling tools to code to deployed applications. Open and extensible, it relies on open industry (OMG) standards, well-tested functionality, and high-performance tools. It supports 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 the system. The run-time software platform reduces complexity and increases reliability of software applications by providing reusable technological building blocks: an operating system, middleware, and application management services.

DREMS applications platform are built from software components that interact via only well-defined interaction patterns using security-labeled messages that support Multi-Level Security [1], and are allowed to use a specific set of low-level services provided by the operating system. Low-level services include messaging and thread synchronization primitives, but components do not use these directly, only via the middleware-provided framework abstractions. Specialized services distributed across the platform are used to control the lifecycle and update applications on demand.

The middleware libraries implement the high-level communication abstractions (synchronous and asynchronous interactions) using low-level services provided by the underlying distributed hardware platform. The DREMS Operating System, a set extension to the Linux kernel, implements all the critical low-level services to support resource sharing (incl. spatial and temporal partitioning), actor[2] management, secure (labeled and managed) information flows, and fault tolerance. The OS also provides strict capability checks for the services an application can use. Three different task levels can exist on the platform: Critical (run as fast as possible), Application (run in a periodic temporal schedule), and Best Effort (run whenever possible).

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 provided that simplifies the tasks of the application developers and system integrators.

Demonstration: We cover a complete application development cycle from design in the modeling tools to execution on a set of fanless computing nodes used to emulate a cluster of three satellites. These nodes contain a 1.6 GHz Atom N270 processor and 1 GB of RAM and communicate on a private gigabit subnet. To this subnet are also connected a physics simulation node running the Orbiter spacecraft simulation tool (http://orbit.medphys.ucl.ac.uk/) and a development node running Dummynet[2] to control the subnet's bandwidth, latency, and packet loss on a per-link basis, similarly to Emulab.

Each satellite in the emulated cluster will run two applications of different criticality levels: a cluster management application and a CPU-intensive image processing application. The cluster management application controls the (simulated) satellite hardware (satellite state, propulsion system, etc.) to maintain orbit and ensures safe cluster operation, therefore it is run as a critical application. The image processing application is not as critical and therefore runs in temporal partitions.

This application will demonstrate utility of the platform. We will also show the initial research results from our work on design time verification of properties such as network quality of service (QoS) and component performance characteristics for the applications developed and deployed on the platform.

Acknowledgments: This work was supported by the DARPA System F6 Program under contract NNA11AC08C.

References

[1] J. Alves-Foss, C. Taylor, and P. Oman, "A Multi-layered Approach to Security in High Assurance Systems," in *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS '04)*, 2004, pp. 10–.

[2] M. Carbone and L. Rizzo, "Dummynet revisited," *SIGCOMM Computer Communication Review*, vol. 40, no. 2, pp. 12–20, Apr. 2010.

[1] **D**istributed **RE**altime **M**anaged **S**ystems

[2] Actors are processes with persistent identifiers

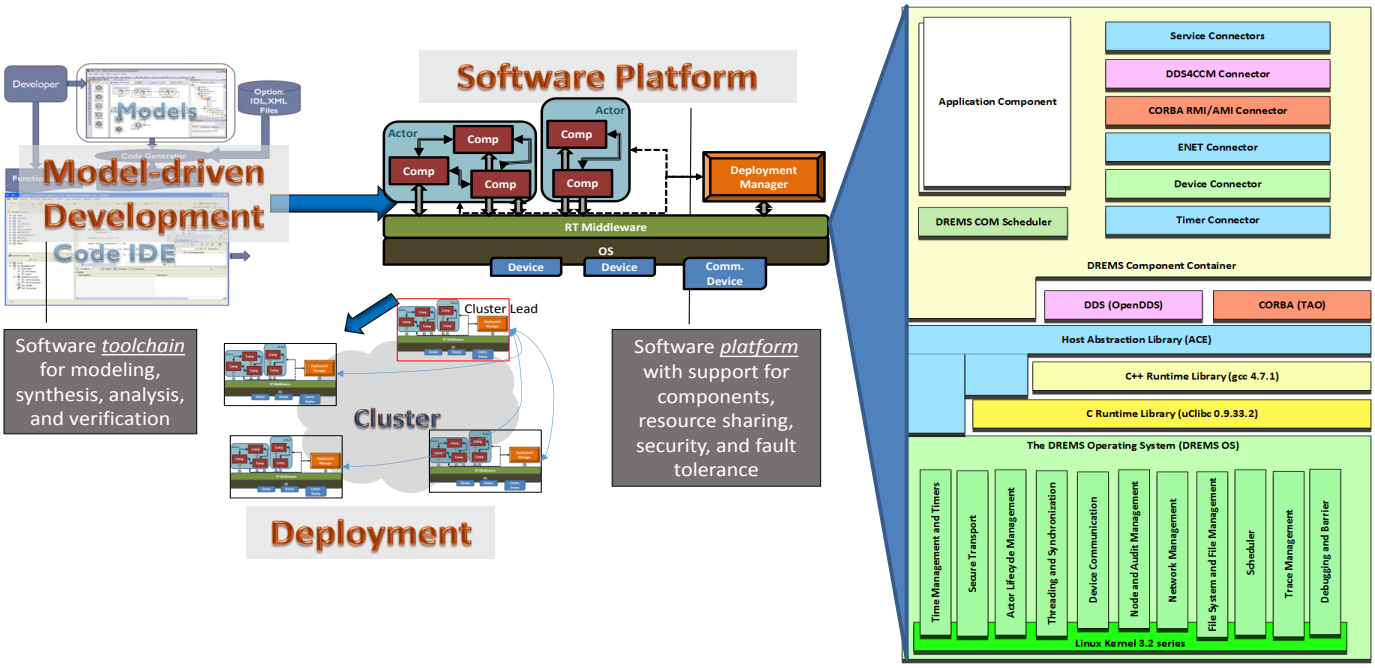

Fig. 1. DREMS Overview : shown is the application development and deployment process and the parts of the *DREMS* platform used in each step. The software toolchain consists of the modeling language, the constraint checking (*i.e.* information flow checks at configuration time, network admittance checks, and partition schedulability checks), the code generation, and the verification tools. The software platform consists of the operating system services and the middleware infrastructure code which provides all allowable services securely to the components. The management of the applications is also handled by dedicated platform applications which are secure and maintain the lifecycle of the applications.

Fig. 2. Development system and DREMS cluster : The bottom right of the image shows the 3 computing nodes used for this application deployment; the left screen shows their three corresponding satellites simulated in orbiter (which is communicating with the three nodes); and the right screen shows the application development using the modeling tools.

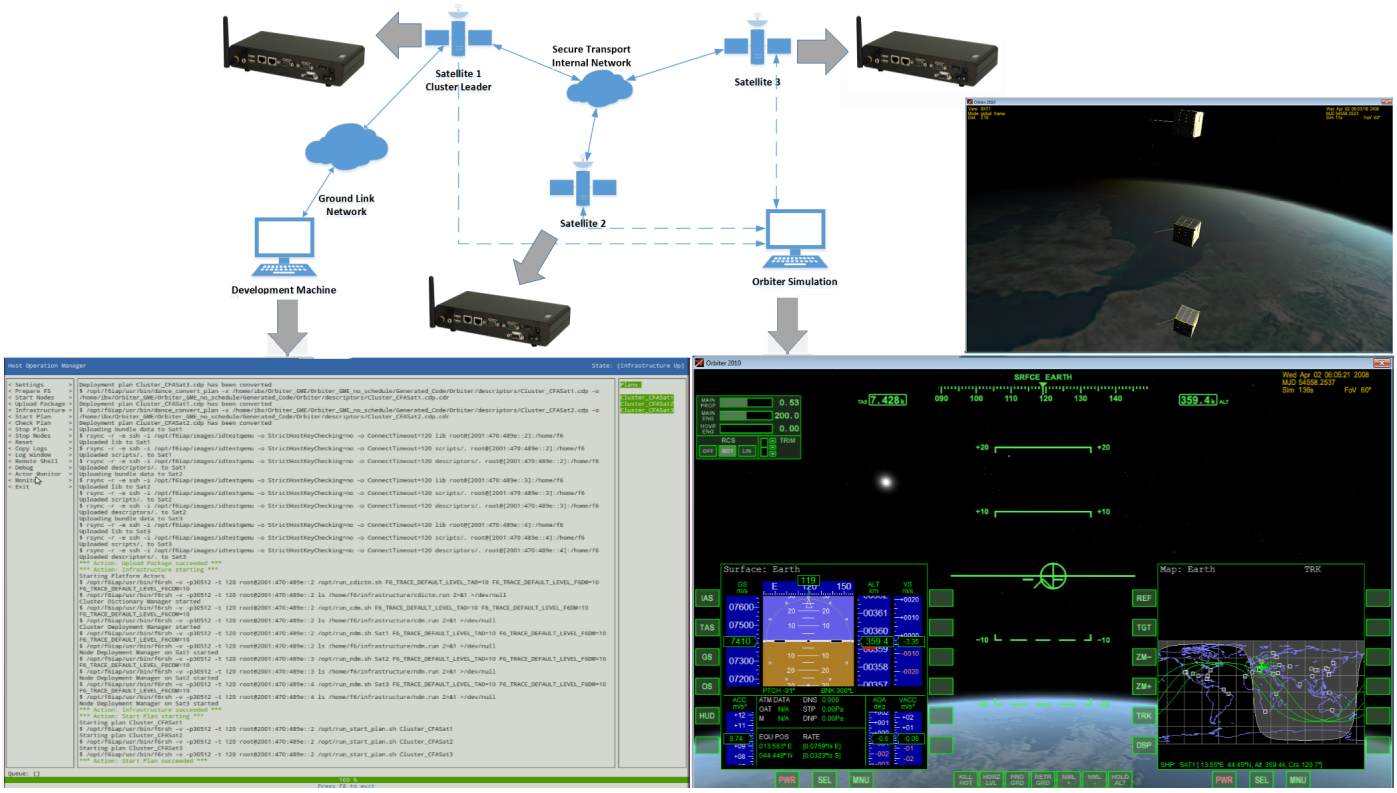

Fig. 3. Network setup : This image shows an overview of the network between the nodes and the simulation and development machines shown in Figure 2. Only satellite 1, which is the cluster leader, communicates with the ground network. When satellite 1 receives a command, *e.g.* scatter, from the ground network, it relays it to the other satellites so that all satellites can perform the maneuver. All satellites communicate with Orbiter which simulates each satellite's orbital mechanics.

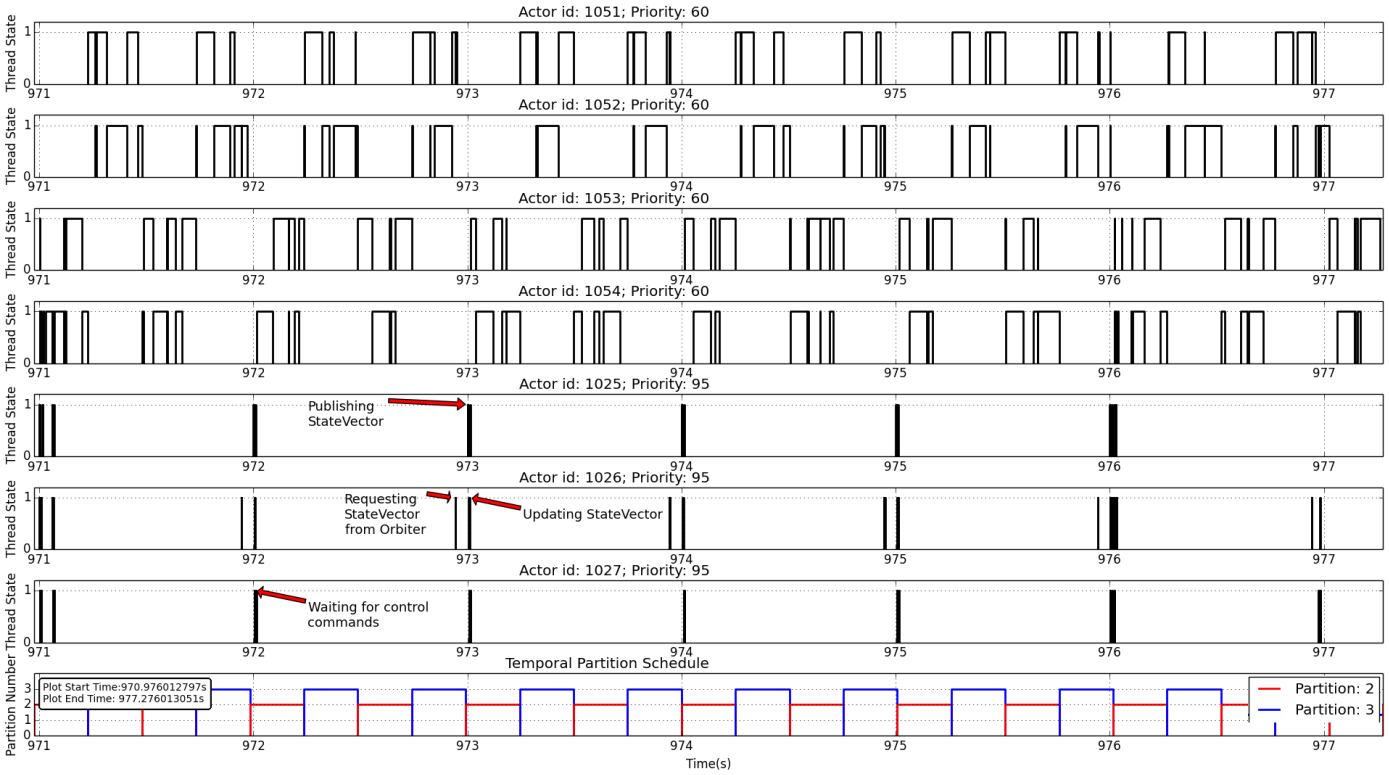

Fig. 4. Application activity log : Actors 1051,1052,1053, and 1054 belong to the CPU-intensive image processing application, which tries to consume as much CPU as possible, but runs in temporal partitions at a lower priority than the cluster management application. Actors 1025,1026, and 1027 belong to the critical cluster management application which is not constrained by temporal partitioning and runs at a higher priority than the image processing application. The partition schedule of the four image processing application actors is shown at the bottom for reference; Actors 1051 and 1052 belong to partition 3 and actors 1053 and 1054 belong to partition 2. The cluster management application activity is annotated in the activity log.