An Information Architecture Platform for Mobile, Secure, and Resilient Distributed Systems

Abhishek Dubey*, William Otte and Gabor Karsai; Vanderbilt University/ISIS
Alessandro Coglio
Kestrel Institute

Supported by DARPA and NASA/ARC under contract # NNA11AB14C
Outline

- Overview and challenges for Dynamic, Distributed, Real-Time Cyber-Physical Systems.
- Solution overview
  - Trusted Platform Services.
  - Generated System Interfaces
  - Modularized Application Construction with strict Resource Bounds
  - Temporally partitioned CPU Schedule
  - Information Flow Partitioning
- Example
DRE systems are ubiquitous and pervasive.

However, fractionated space craft, swarms of UAVs, and intelligent power devices forming the ‘smart power network have some novel properties and challenges.

These systems are indeed dynamic platforms.
Challenges for a Software platform
Enabling DREMS 1/2

- Applications typically span multiple nodes, for reasons related
  - to the availability of resources: some nodes may have sensors,
  - some may have actuators, some may have the computing
  - or storage resources.
- Support multiple missions over the system’s life.
Challenges for a Software platform
Enabling DREMS 2/2

- Remote access.
- Share applications from different vendors and users with different privileges.
- Information sharing/leakage between applications must be controlled under an overall system security policy.
- Performance isolation is critical
  - One application should not be able to affect the functionality of another application.
  - Absence of strong performance isolations are also going to result in security problems.
Solution Approach

- Trusted Platform Services
- Generated System Interfaces
- Modularized Application Construction with strict Resource Bounds
- Temporally partitioned CPU Schedule
- Information Flow Partitioning
Solution: DREMS

Distributed REaltime Managed System. DREMS consists of:
1. A software *platform*, consisting of an OS and middleware
2. A software *toolchain*, for modeling applications

1. Software *toolchain* for modeling, synthesis, analysis, and verification
2. Software *platform* with support for resource sharing, security, and fault tolerance
Trusted Platform Services

- The kernel
  - MLS and strict information flow partitioning
  - Strong data structure integrity checks and timeout policies on the Application Binary Interfaces

- A managed distributed runtime software platform
  - Privileged Runtime Software Actors responsible for system level tasks
  - Coordinated deployment of distributed applications
Modularized Applications

- Apps are architected as one or more actors that share information via secure messaging.
- Actors encapsulate components, with well-defined interaction semantics.
- Actors and their ports have security labels that designate the security classification of messages the actor/port can send/receive.
- All actor communications are via secure transport (through the OS kernel) that enforces security policies. No other interactions are possible.

Actor’s resource needs are declared by developer, admitted by the system integrator; and enforced by the platform.
Distributed Application Infrastructure

- **Infrastructure - Middleware:**
  - User-space software library to provide actors with higher-level capabilities on top of system services
  - Interactions
    - call/return and publish/subscribe
  - The middleware provides common, reusable services for building complex, distributed applications: secure component interactions, location transparency, network addressing, resource management, and fault tolerance
Robust Software Component Model

- Clearly delineate computational aspect from communication aspect.
- Component work is divided into operations which are scheduled one at a time.
  - The computation code written by developer is free from any synchronization code, which is one of the common mistakes made by the developers.
  - The interleaving of the operations is guaranteed by the component scheduler
    - First in First Out, Priority, Earliest Deadline First.
Operating System

- Resource sharing is strictly monitored and controlled.
- Fine-grained capability model that controls access to different system services.
- All interactions among actors are via ‘secure transport’.
- Part of the Trusted Computing Base that enforces the MLS/MAC security policies.
- The operating system must protect itself from malicious/faulty applications, and isolate the applications from each other.
How are we different from Linux?

- System calls that are simpler and more restrictive
- Interfaces spec’d in IDL, generated skeleton code automatically includes data integrity, process capability checks and data movement code.
- Temporal partitioning with work conserving behavior
- Secure transport (new communication mechanism)
Operating system scheduler

- Real-time scheduling model that supports mixed-criticality systems.
  - System (platform Actors – uses simple priority scheduling)
  - Application (uses temporal partitioning)
  - Best effort (uses CFS)
- Temporal partitioning. Each partition is temporally separated from others.
- Partition schedule is synchronized across nodes in a cluster.
- Partitioning has been extensively used in avionics system. We extend the concept
  - Enables multiple actors within a partition (can control processor utilization of each actor)
Temporal partitioning
Information flow partitioning: Secure Transport

- Disabled BSD sockets
  - Regular Actors cannot initiate or configure network communications
  - Can use socket-like communication APIs with security labels and time-stamping

- Actors communicate through messages using a mechanism called Secure Transport

- All messages have a security label and must obey a set of Mandatory Access Control (MAC) policies
Secure Transport

- The secure transport mechanism enforces the following:
  - Actors write messages only to **endpoints**
  - Endpoints and **flows** are configured only by trusted platform Actors; used by regular Actors
  - Enforces mandatory access control for the messages.
  - Supports both UDP and SCTP protocol.
Secure Transport Concepts: Endpoints

- Created on behalf of application.
- Analogous to ‘socket handles’ in traditional networking APIs

Endpoint categories

- Remote Message Endpoint – May be used to communicate messages across distributed nodes
- Local Message Endpoint – May be used to communicate with other actors *on the same node*
- Kernel Message Endpoint – Used to communicate with kernel – notifications to actor, messages to devices
- Loopback Endpoint – May be created by actor for communication amongst its threads. Not subject to MAC/flow restrictions
Secure Transport Concepts: Flows

- Connectionless, logical associations between endpoints
- Communication only possible between endpoints with compatible flows
  - Source: Allows messages to flow from the endpoint to the specified destination
  - Destination: Allows message to flow from the specified source to the endpoint
  - Bidirectional: Allows bidirectional traffic with the specified destination
Secure Transport Concepts: Mandatory Access Control

- All messages must have an assigned label *by the sending actor*.
- All communication objects (endpoints, actors, nodes, network links) are assigned *compatible ranges*.
  - Endpoints owned by an actor **must fall within** the assigned range of that actor.
  - Actors must have label ranges that are a **subset** of the label ranges of the node that hosts them.
  - Messages may only travel on a network link with a label that is **within the label range** of the network link.
  - The messages can go to a destination if and only if destination has a label that **dominates** the label of destination.
- Messages having labels **not compatible** with a communication object are **rejected** by the operating system.
Security Context: Using Secure Transport Within Software Components

- Software components use higher level APIs that provide
  - Publish subscribe interaction
  - Remote/Asynchronous method invocation.

- A special feature of the components is that they execute operations one at a time in a non-preemptive manner.

- The middleware ensures that the component enters a security context with the label of the received message.

- The component can change the security context if it has access to multiple labels and it wants to use a different label altogether.
Each satellite runs two applications:
(1) A system critical cluster flight application (CFA)
(2) An image processing application (IPA).
(3) Nodes are synchronized on the order of 10 micro seconds.
Review of key ideas

1. Applications are built from **Actors**
2. Actors encapsulate **components**
   1. Built from software components
   2. Interact through well-defined patterns
   3. Can only use restricted set of low-level OS services
3. The **middleware** libraries implement high-level communication abstractions (synchronous and asynchronous interactions) and a component model
4. The **Operating System** implements all the critical low-level services for resource sharing
   1. Built as a set of extensions to Linux
Summary: DREMS Information Architecture Platform

- A design-time tool suite
  - Debugging and Testing
  - Reusable Software Construction
  - Performance Verification e.g. Network Admittance tests
  - Information Flow Checks

- A managed distributed runtime software platform
  - Privileged Runtime Software Actors responsible for system level tasks
  - Coordinated deployment of distributed applications
  - MLS and strict information flow partitioning
  - Strong data structure integrity checks and timeout policies on the Application Binary Interfaces.