



University of California

Institute for Software Research

## Event-based systems and Software Architectures: Out of the Shadows and into the Mainstream

- Panelist: Nicolas Rouquette, NASA JPL
- Context: The Mission Data System project (MDS)
- Relation:
  - MDS uses two architecture styles
    - State Analysis (invented at JPL)
    - Component/Connector style based on xADL2.0

# Timing @ JPL

---

## ■ Internal factors

- ◆ State analysis fundamentally involves events (e.g., state change notification)
- ◆ In our xADL runtime, function calls can be reified into objects that can be operated on (I.e., enabling factor)

## ■ External factors

- ◆ JPL-Sun collaboration on Real-Time Java
- ◆ RTSJ specification involves several events

# Applicability @ JPL

- Thread scheduling (a la RTSJ)
  - ◆ Scheduler posts "miss" and "overrun" events (RTSJ)
  - ◆ Thread state changes are event sources (MDS)
- Mission Planning & Scheduling (MDS)
  - ◆ How should the system react to events when it is involved in other competing activities?
  - ◆ Low-level controllers & estimators must be instrumented to send events
- Verification & Validation (w/ NASA Ames)
  - ◆ Decouple verification & checking using instrumentation
  - ◆ Livelock, deadlock are two sample problems solvable with logs of lock/unlock events.

# Scalability: Performance matters but architecture knowledge is key

---

- The performance syndrome
  - ◆ Events everywhere...
  - ◆ ...progress nowhere!
- Strategy:
  - ◆ Optimize event communication
    - ☞ Requires knowledge of the architecture
      - Global vs. local knowledge => closed vs. open world
    - ☞ At runtime
      - E.g., during architecture prescription
      - E.g., during software reconfiguration
    - ☞ At design time
      - E.g., state machine code generation
      - E.g., model-based software transformation

# Training

- Traditional "flight software" at JPL
  - ◆ A bit of magic, a lot of wisdom
  - ◆ A lot of experience & attention to detail
  - ◆ A lot of confidence, creativity and testing
  - ◆ => Very difficult to teach how to do it
- MDS approach: Architecture hoisting
  - ◆ Focus on the two architectures
    - ☞ State analysis (states, controllers, estimators, sensors,...)
    - ☞ Software architecture (components, connectors, ...)
  - ◆ Code is synthesized from the architecture
    - ☞ With the right QoS properties built-in
  - ◆ Need: architecture transformation culture
    - ☞ Traditional code generators make homomorphic transformations

# Technology: Transforming Architectures into Code

---

- Taxonomy of connectors
  - ◆ Many dimensions & attributes => many implementations
- Architecture-based transformation
  - ◆ Quality of Service properties may be:
    - ☞ Enforced by design (no runtime overhead)
    - ☞ Actively monitored (needs reification)
  - ◆ Transform the architecture into the software that is engineered to make the selected trades
  - ◆ Paradigm shift from
    - ☞ software-centric
      - people writing lots & lots of code
    - ☞ architecture-centric engineering
      - people writing architectures & transforming them into code