Verifying Distributed Adaptive Real-Time (DART) Systems

Pipelined ZSRM Scheduling
- Reduces pipeline to single-resource scheduling
- Avoids assuming worst alignment in all stages
But need to deal with transitive interferences due to zero-sack
Ongoing work: theory worked out, implementing scheduler in Linux

Functional Verification
Prove application-controller contract for unbounded time
- Previously limited to bounded verification only
Prove controller-platform contract via hybrid reachability analysis
- Done by AFRL
Working on automation and asynchronous model of computation

Proactive Self-Adaptation Using Probabilistic Model Checking

DART Vision
A sound engineering approach based on the judicious use of precise semantics, formal analysis and design constraints leads to assured behavior of (DART) systems while accounting for:
- critical requirements
- probabilistic requirements
- uncertain environments
- necessary coordination
- assurance at source code level

DART Process
1. Enables compositional and requirement specific verification
2. Use proactive self-adaptation and mixed criticality to cope with uncertainty and changing context
3. ZSRM Schedulability/Timing
4. Software Model Checking (Functional)
5. Statistical Model Checking (Probabilistic)

Example: Self-Adaptive and Coordinated UAS Protection

Distributed Statistical Model Checking
Batch Log and Analyze

DMPL: DART Modeling and Programming Language
- C-like language that can express distributed, real-time systems
- Semantics are precise
- Supports formal assertions usable for model checking and probabilistic model checking
- Physical and logical concurrency can be expressed in sufficient detail to perform timing analysis
- Can call external libraries
- Generates compilable C++
- Developed syntax, semantics, and compiler (dmplc)