Verifying Distributed Adaptive Real-Time (DART) Systems

Sagar Chaki, Dionisio de Niz

October 8, 2015

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Background

Distributed Adaptive Real-Time (DART) systems are key to many areas of DoD capability (e.g., autonomous multi-UAS missions) with civilian benefits.

However achieving high assurance DART software is very difficult

- Concurrency is inherently difficult to reason about.
- Uncertainty in the physical environment.
- Autonomous capability leads to unpredictable behavior.
- Assure both guaranteed and probabilistic properties.
- Verification results on models must be carried over to source code.

High assurance unachievable via testing or ad-hoc formal verification

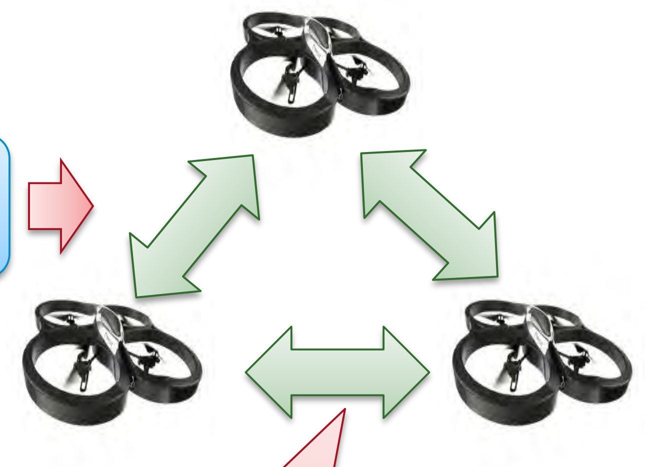
Goal: Create a sound engineering approach for producing high-assurance software for Distributed Adaptive Real-Time (DART)

DART Approach

1. Enables compositional and requirement specific verification
2. Use proactive self-adaptation and mixed criticality to cope with uncertainty and changing context

1. ZSRM Schedulability (Timing)
2. Software Model Checking (Functional)
3. Statistical Model Checking (Probabilistic)

System + Properties (AADL + DMPL) → Verification → Code Generation →



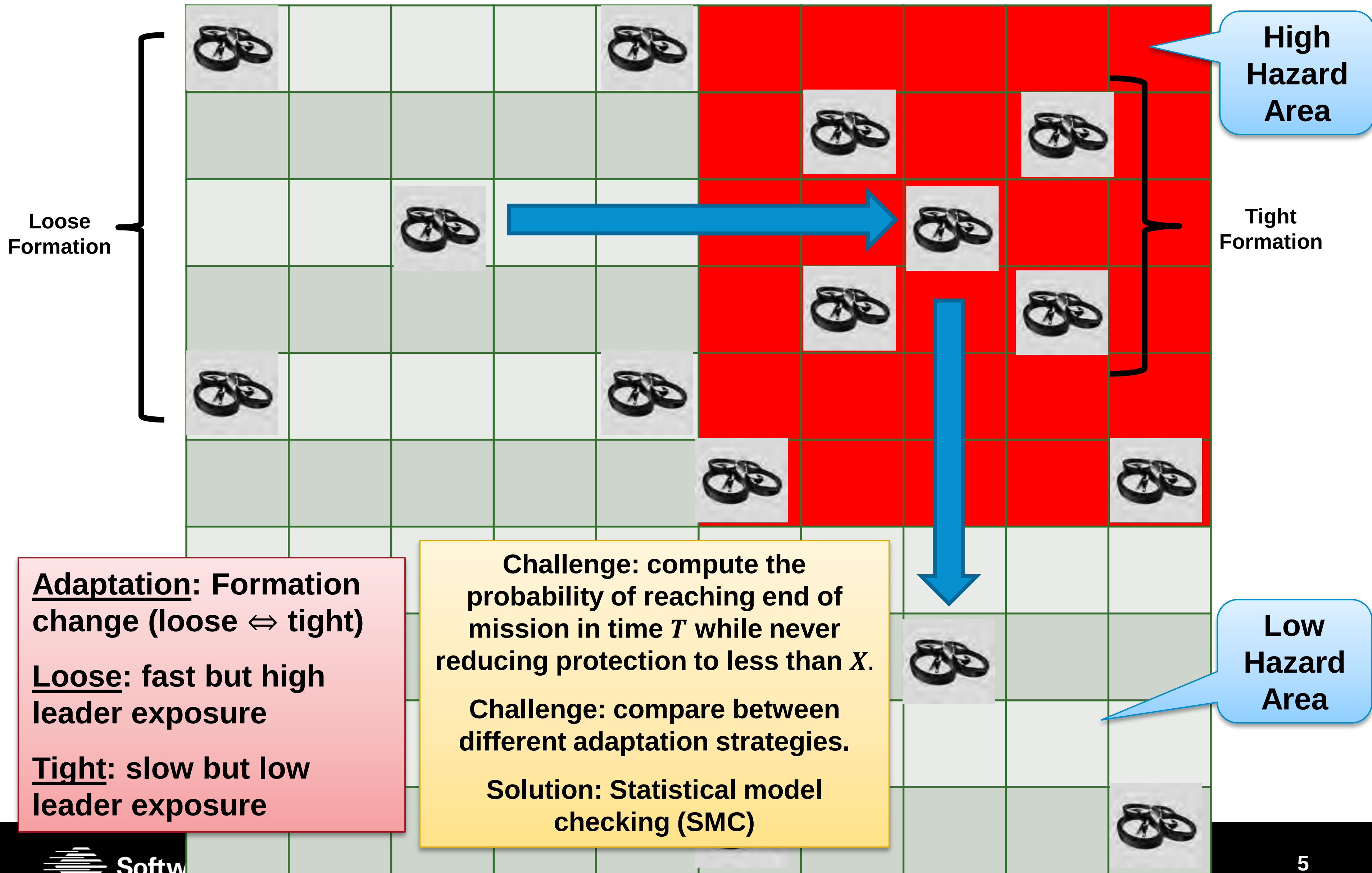
Brings Assurance to Code

1. Middleware for communication
2. Scheduler for ZSRM
3. Monitor for runtime assurance

Demonstrate on DoD-relevant model problem (DART prototype)

- Engaged stakeholders
- Technical and operational validity

Example: Self-Adaptive and Coordinated UAS Protection



Adaptation: Formation change (loose ⇔ tight)

Loose: fast but high leader exposure

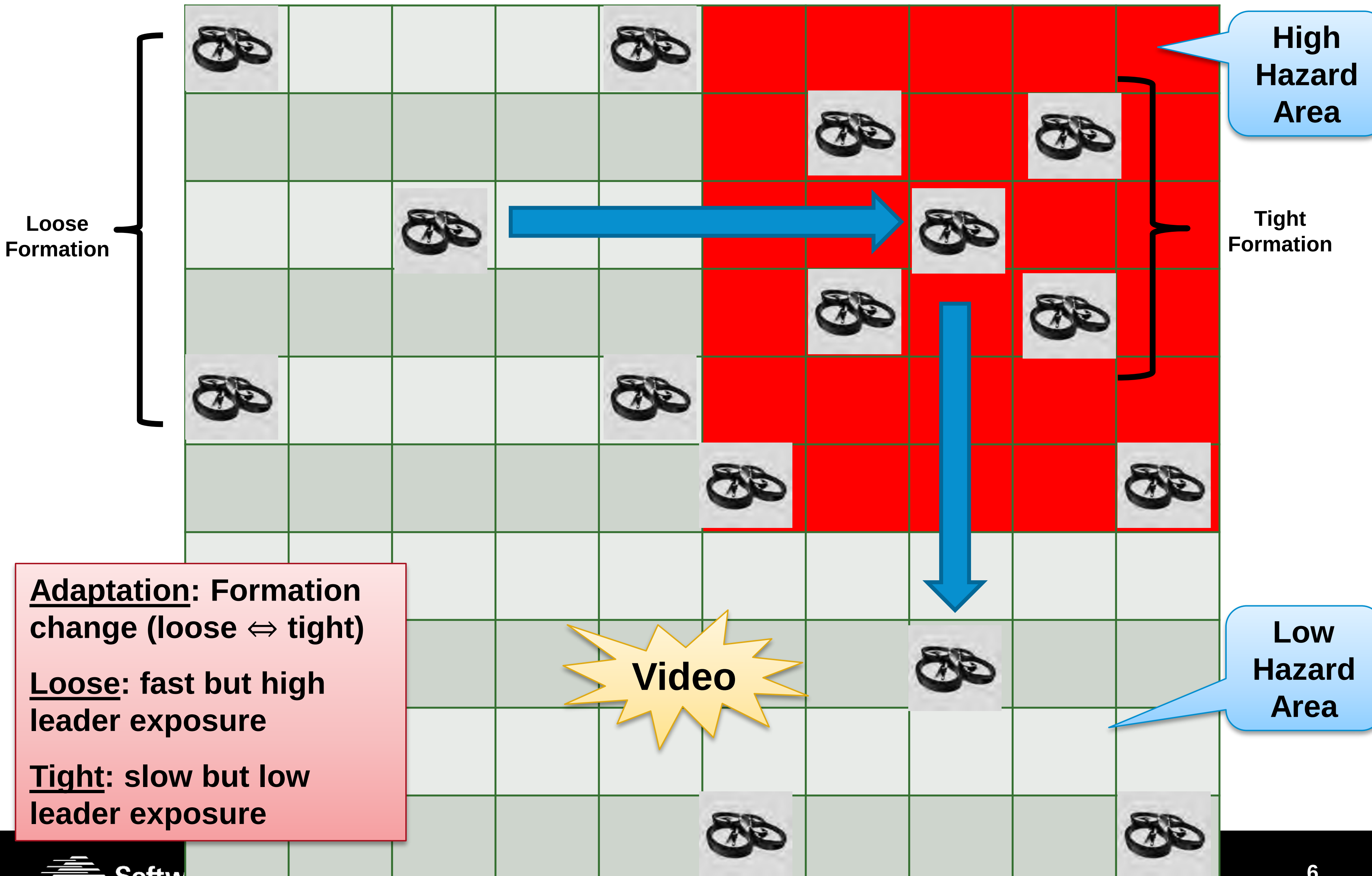
Tight: slow but low leader exposure

Challenge: compute the probability of reaching end of mission in time T while never reducing protection to less than X.

Challenge: compare between different adaptation strategies.

Solution: Statistical model checking (SMC)

Example: Self-Adaptive and Coordinated UAS Protection



Adaptation: Formation change (loose ⇔ tight)

Loose: fast but high leader exposure

Tight: slow but low leader exposure

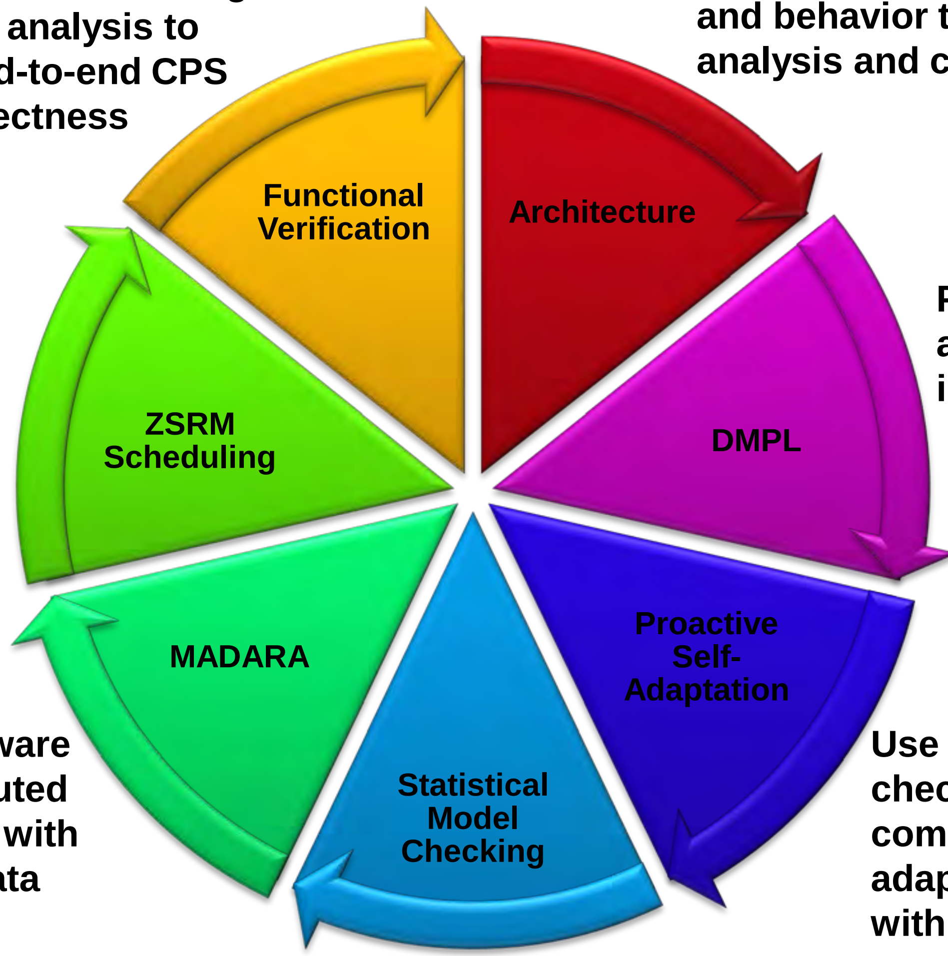
Key Elements of DART

Combine model checking & hybrid analysis to ensure end-to-end CPS correctness

Constrain the system structure and behavior to facilitate tractable analysis and code generation

Ensures high-critical tasks meet their deadlines despite CPU overload

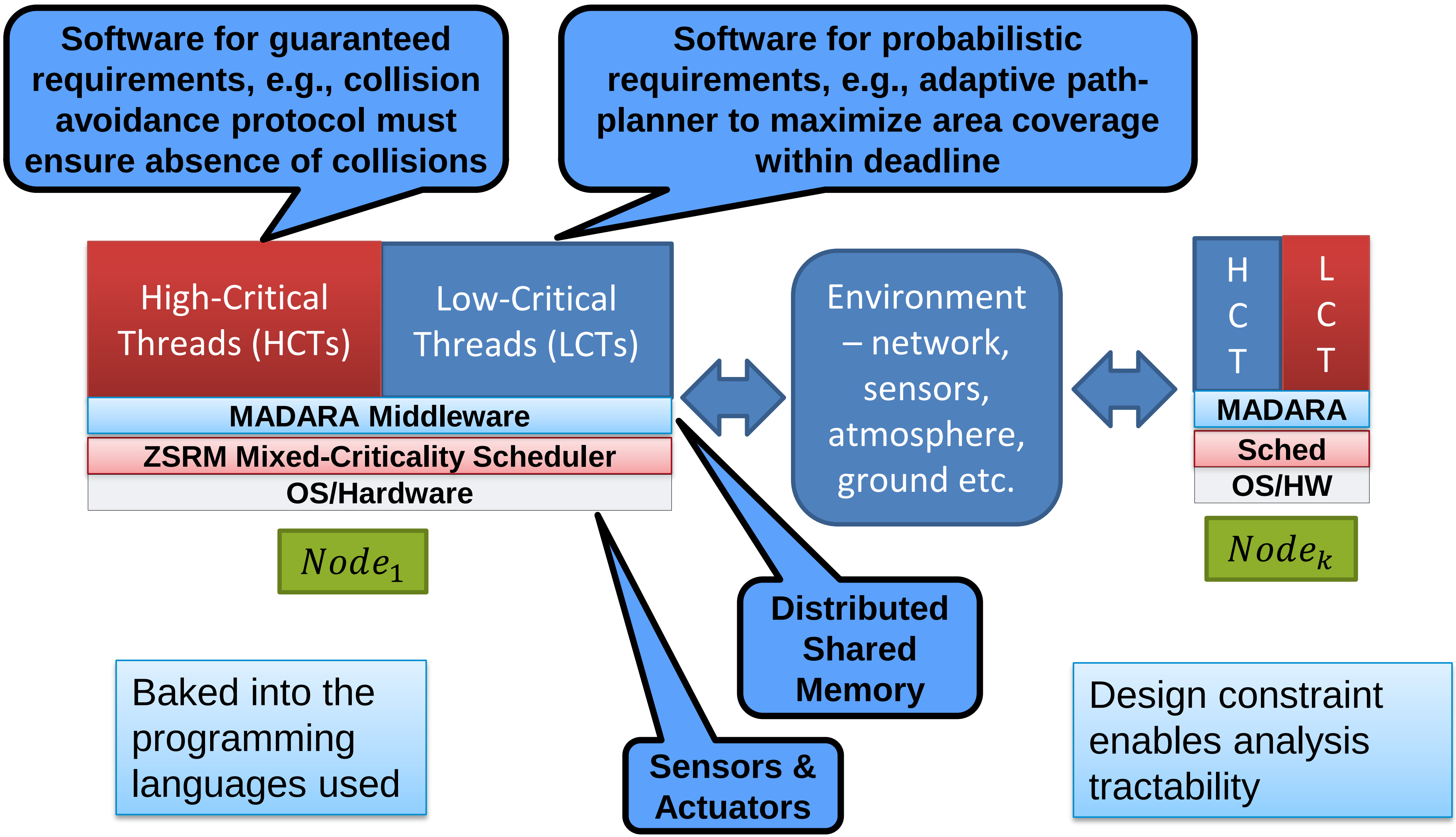
Program DART systems and specify properties in a precise manner



Efficient middleware provides distributed shared variables with well-defined data consistency

Use probabilistic model checker to repeatedly compute optimal adaptation strategies with bounded lookahead

Evaluate adaptation strategy quality over mission lifetime


Software for guaranteed requirements, e.g., collision avoidance protocol must ensure absence of collisions
Software for probabilistic requirements, e.g., adaptive path-planner to maximize area coverage within deadline
High-Critical Threads (HCTs)
Low-Critical Threads (LCTs)
MADARA Middleware
ZSRM Mixed-Criticality Scheduler
OS/Hardware
$Node_1$
Environment – network, sensors, atmosphere, ground etc.
HCT
LCT
MADARA
Sched
OS/HW
$Node_k$
Distributed Shared Memory
Sensors & Actuators
Baked into the programming languages used
Design constraint enables analysis tractability

DART Modeling and Programming Language (DMPL)

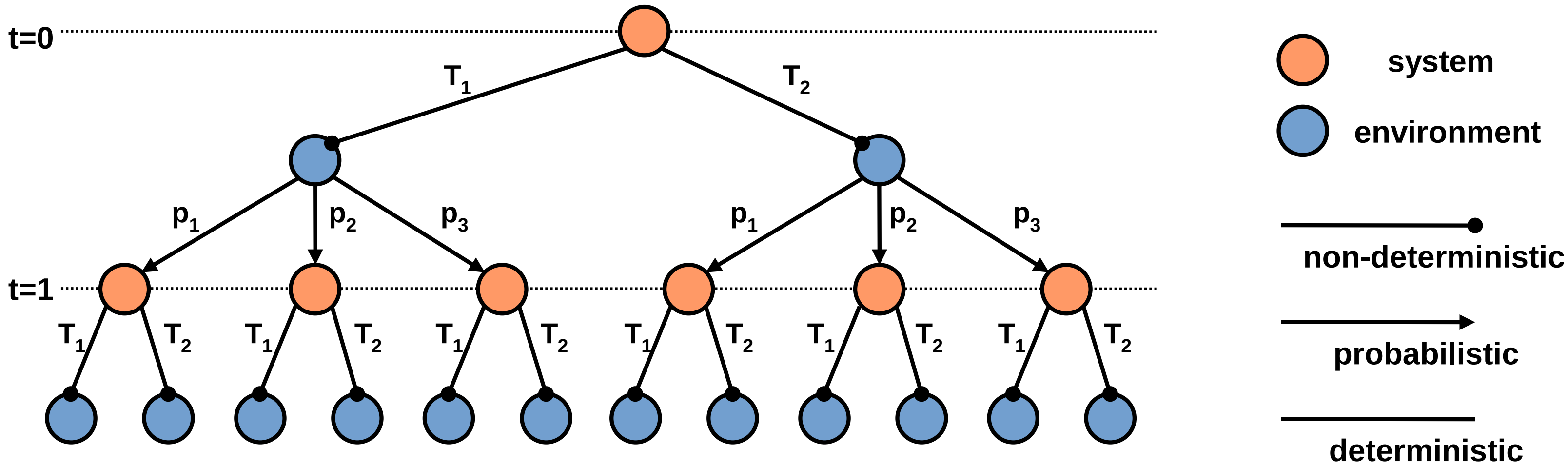
C-like language that can express distributed, real-time systems

- 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++

Open Source Release on Github

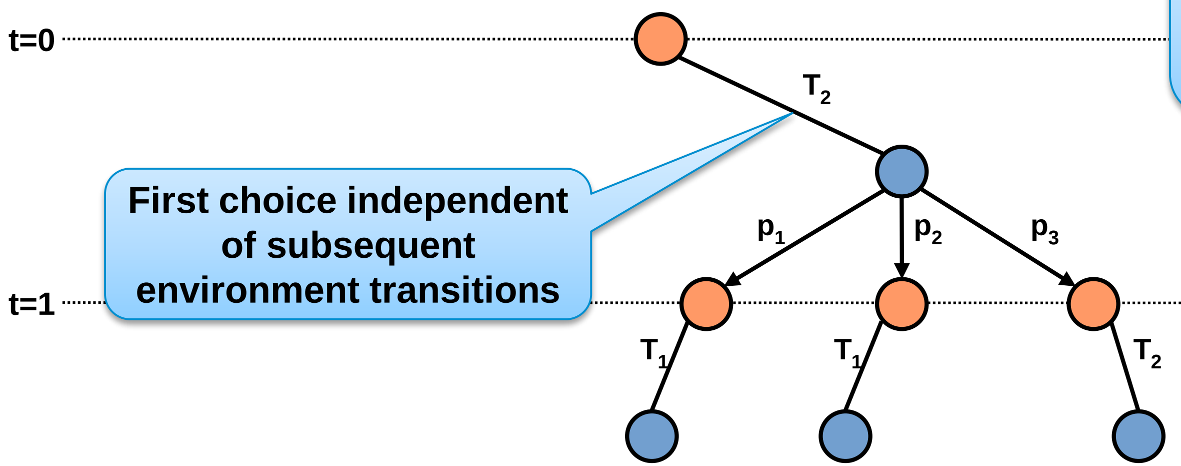
Developed syntax, semantics, and compiler (dmplc)

DMPL supports the right level of abstraction to formally reason about DART systems



PRISM
strategy synthesis

Resolves nondeterministic choices to maximize expected value of objective function



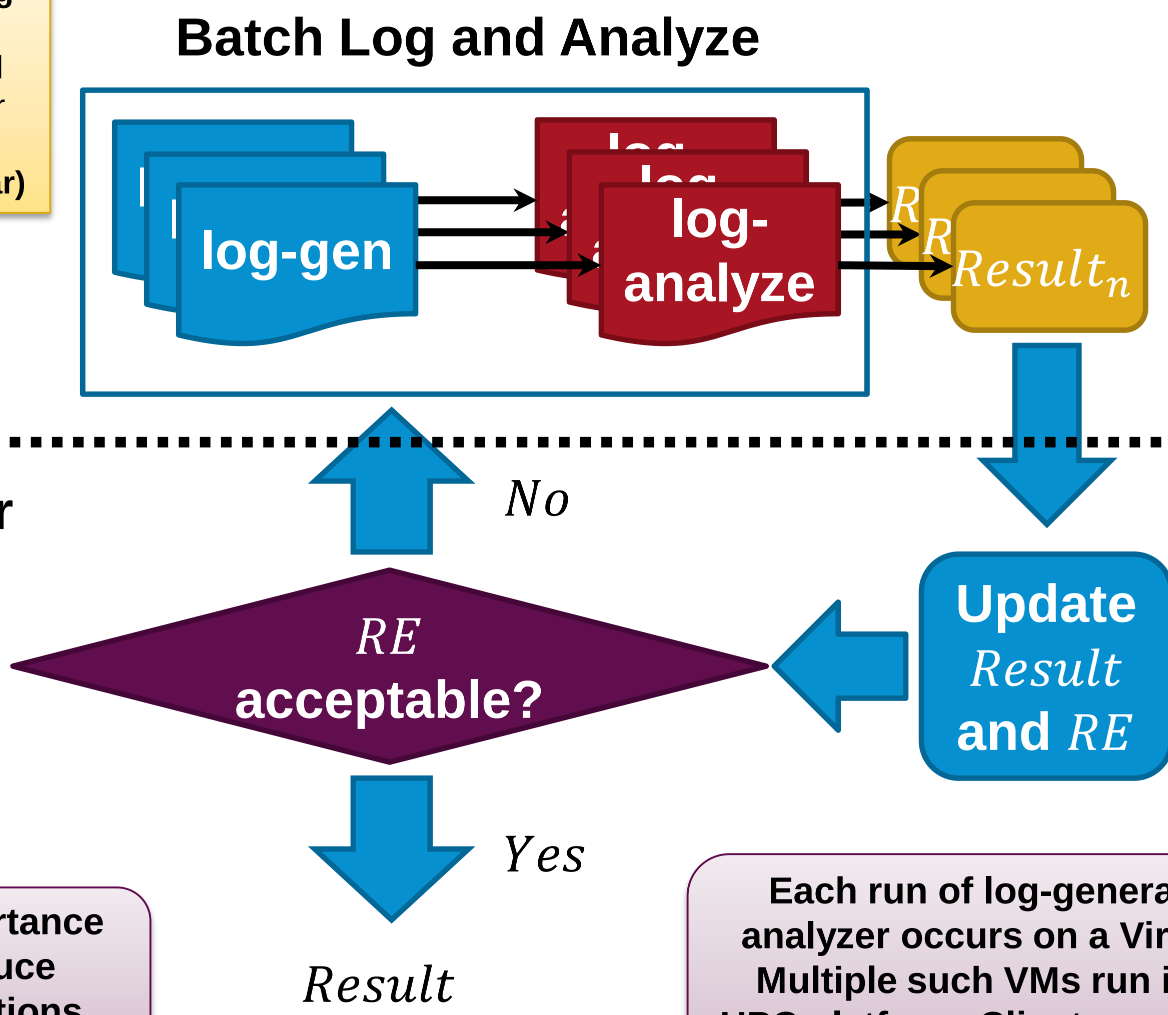
First choice independent of subsequent environment transitions

Ongoing work: replace probabilistic model checking with dynamic programming for speed.

Statistical Model Checking of Distributed Adaptive Real-Time Software. David Kyle, Jeffery Hansen, Sagar Chaki. **In Proc. of Runtime Verifcation 2015 (to appear)**

SMC Client

SMC Aggregator



Future Work: Importance Sampling to reduce number of simulations needed for "rare" events.

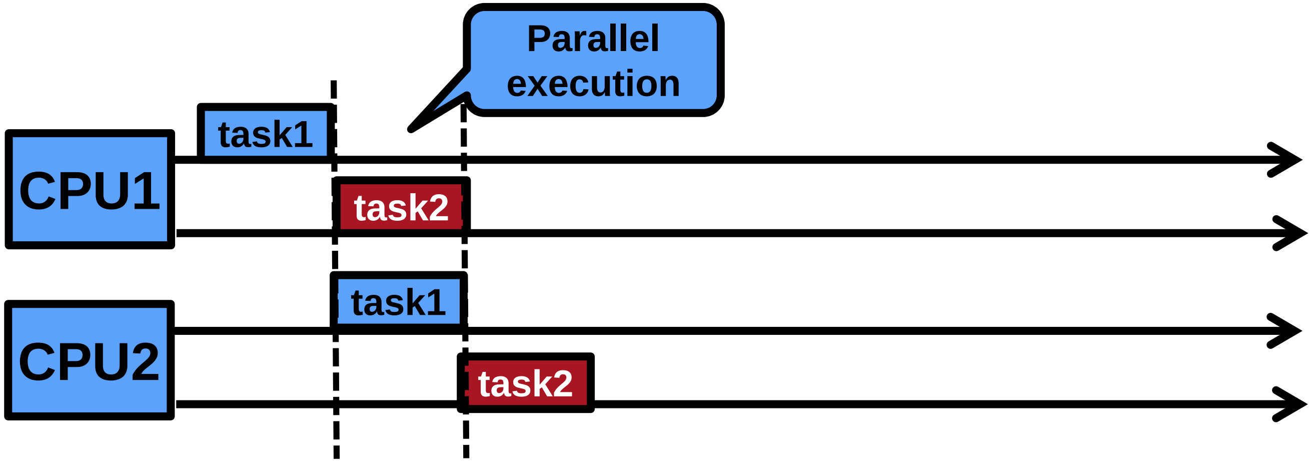
Each run of log-generator and log-analyzer occurs on a Virtual Machine. Multiple such VMs run in parallel on HPC platform. Clients can be added and removed on-the-fly.

Zero-Slack Rate Monotonic (ZSRM) software stack

- ZSRM Schedulability Analysis as AADL/OSATE Plugin
- ZSRM Scheduler as Linux Kernel Module
- ZSRM Priority & Criticality Ceiling Mutexes

End-to-end Zero-Slack Scheduling

- Based on pipelines that allows parallel execution of multiple tasks in different stages.
- Avoids assuming all tasks start together in all stages
- Reduces the end-to-end response time and improves utilization
- Working on submission to RTSS'15



Combining model checking of collision-avoidance protocol with reachability analysis of control algorithms via assume-guarantee reasoning

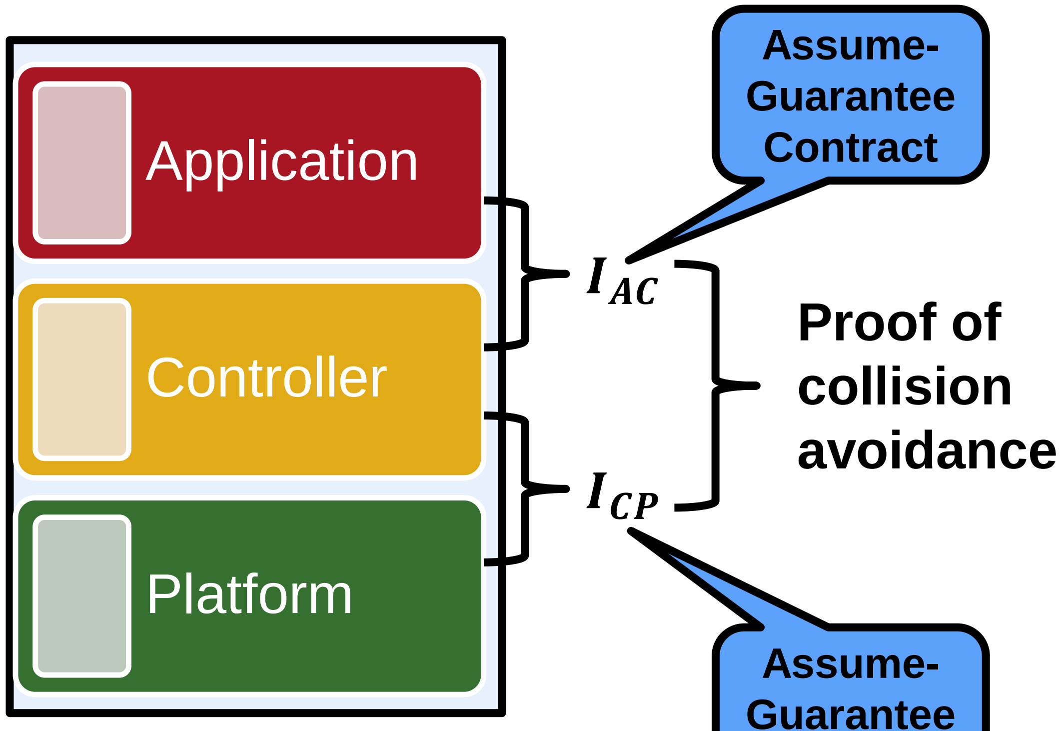
Prove application-controller controller contract for unbounded time

- **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



DART Node

Challenges and Future Work

Transition and application to realistic systems

Logical Isolation between Verified and Unverified Code

Big Trusted Computing Base (Compilers)

Discovered more complexity and nuances about mixed-criticality scheduling (end-to-end)

Importance sampling for distributed systems

Longer term: Fault-Tolerance, Runtime Assurance, Security

Summary

Distributed Adaptive Real-Time (DART) systems promise to revolutionize several areas of DoD capability (e.g., autonomous systems). We want to create a sound engineering approach for producing high-assurance software for DART Systems, and demonstrate on stakeholder guided examples.

Team

Bjorn Andersson
Bud Hammons
Gabriel Moreno
Jeffery Hansen
Scott Hissam
Sagar Chaki
Mark Klein
Arie Gurfinkel
David Kyle
James Edmondson
Dionisio de Niz

https://github.com/cps-sei/dart

QUESTIONS?

Contact Information

Sagar Chaki

Senior MTS

SSD/CSC

Telephone: +1 412-268-1436

Email: chaki@sei.cmu.edu

U.S. Mail

Software Engineering Institute

Customer Relations

4500 Fifth Avenue

Pittsburgh, PA 15213-2612

USA

Web

www.sei.cmu.edu

www.sei.cmu.edu/contact.cfm

Customer Relations

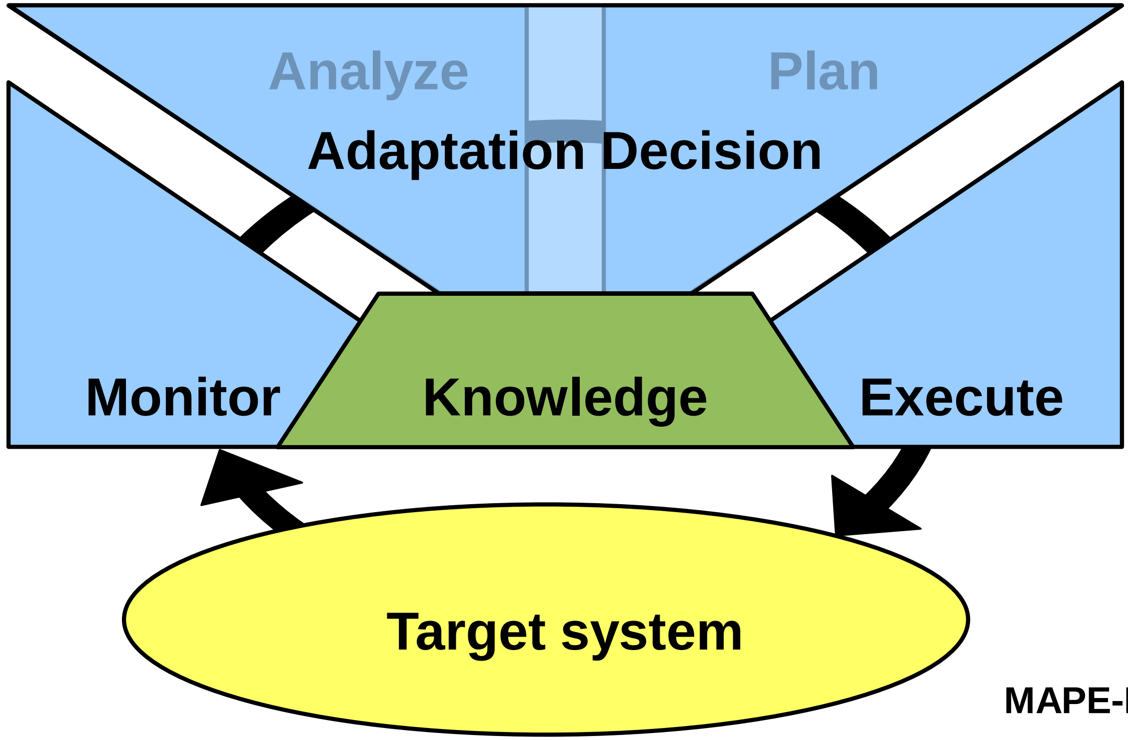
Email: info@sei.cmu.edu

Telephone: +1 412-268-5800

SEI Phone: +1 412-268-5800

SEI Fax: +1 412-268-6257

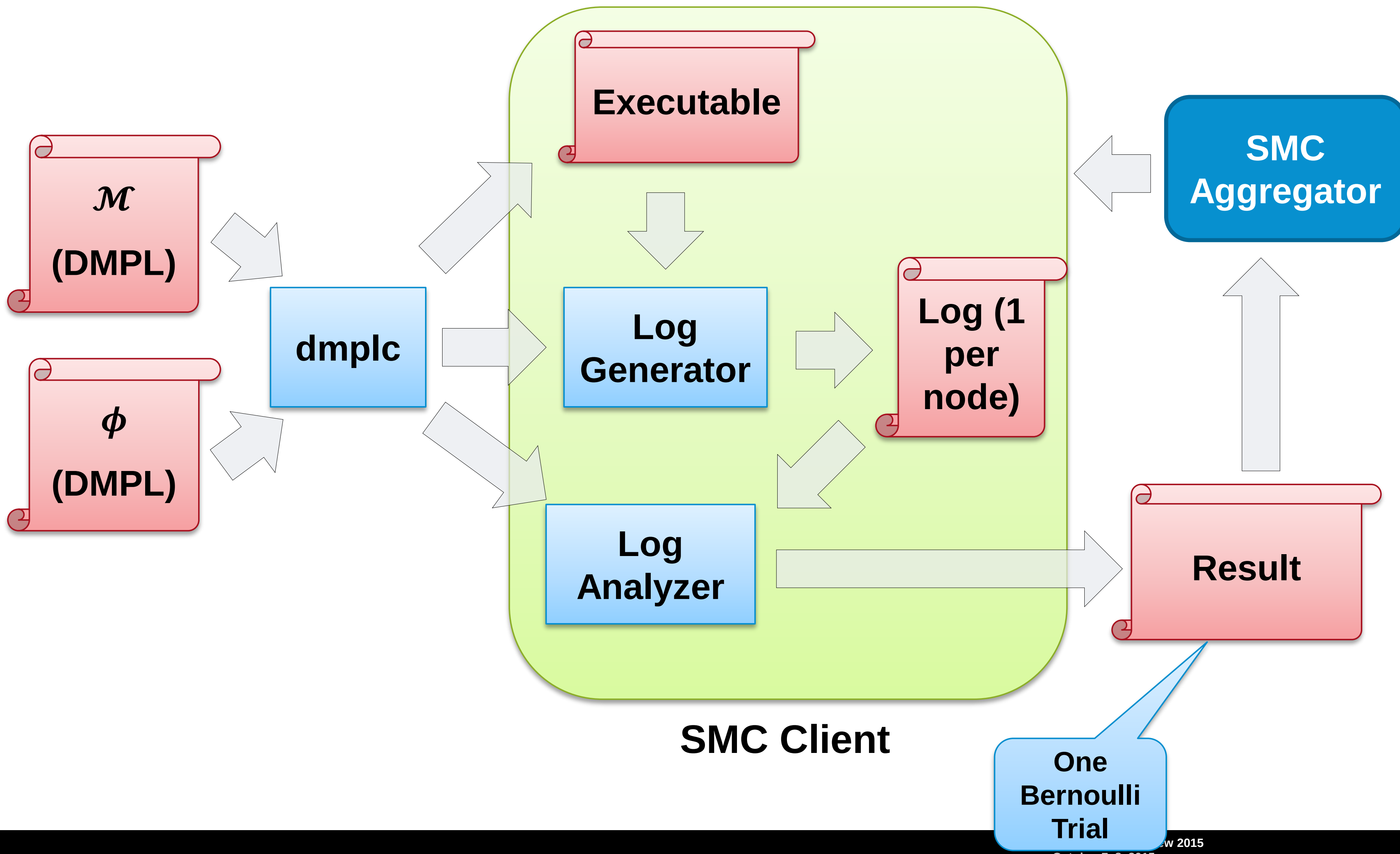
Backup Slides



MAPE-K [Kephart 2003]

Implemented proactive self-adaptation manager in a multi-UAS coordinated protection DART example. Manager adapts by changing system formation to tradeoff between energy consumption and protection provided to a mothership.

Paper presented at ACM/SIGSoft FSE’15: Gabriel Moreno, Javier Camara, David Garlan and Bradley Schmerl, "Proactive Self-Adaptation under Uncertainty: a Probabilistic Model Checking Approach".


$\mathcal{M}$
(DMPL)
ϕ
(DMPL)
dmplc
Executable
Log Generator
Log (1 per node)
Log Analyzer
SMC Aggregator
Result
SMC Client
One Bernoulli Trial