QoS-enabled Large-scale Group Autonomy (ELASTIC)

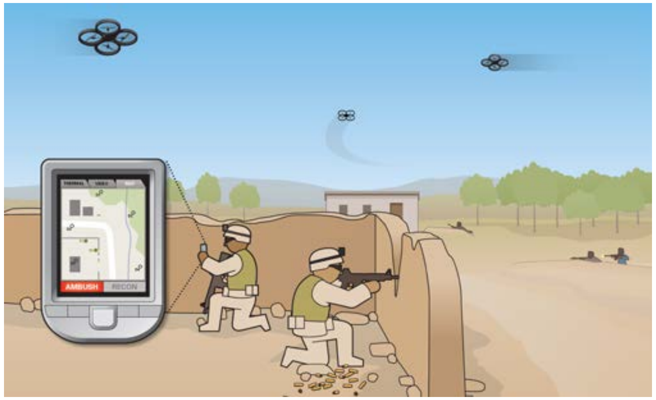
Presenter: James Edmondson (jredmondson@sei.cmu.edu)

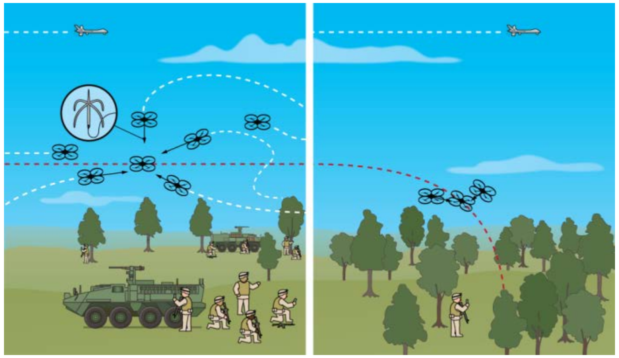
Date: February 18, 2015

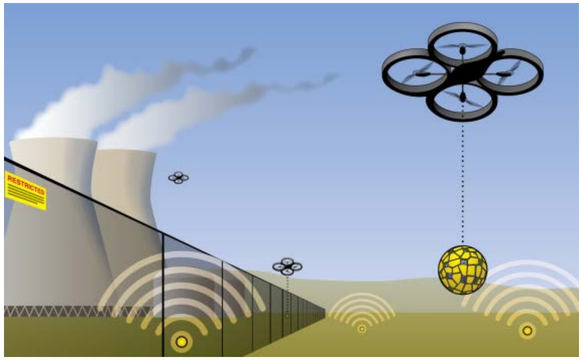
DM-0002124

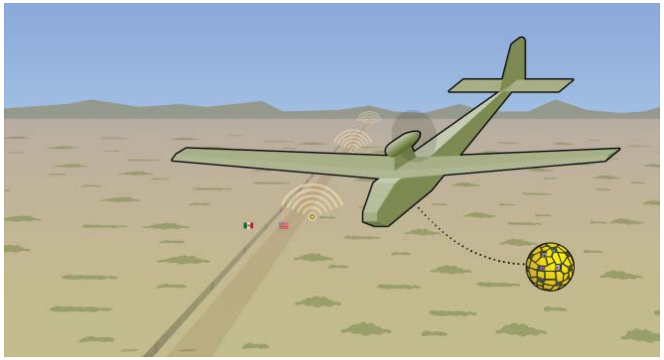
Problems facing large-scale group autonomy in real-world missions

1. Centralized controllers are too easy to attack, and suffer during natural disconnects in wireless environments
2. AI platforms tend to rely on blocking, reliable communication that creates brittle AI
3. Most AI focuses on self-interested, isolated agents in simple missions or preplanned missions that do not adapt well to group objectives
4. AI platforms tend to not be feature rich, do not provide configurable quality-of-service levels, and only support very limited types of AI, learning and platforms

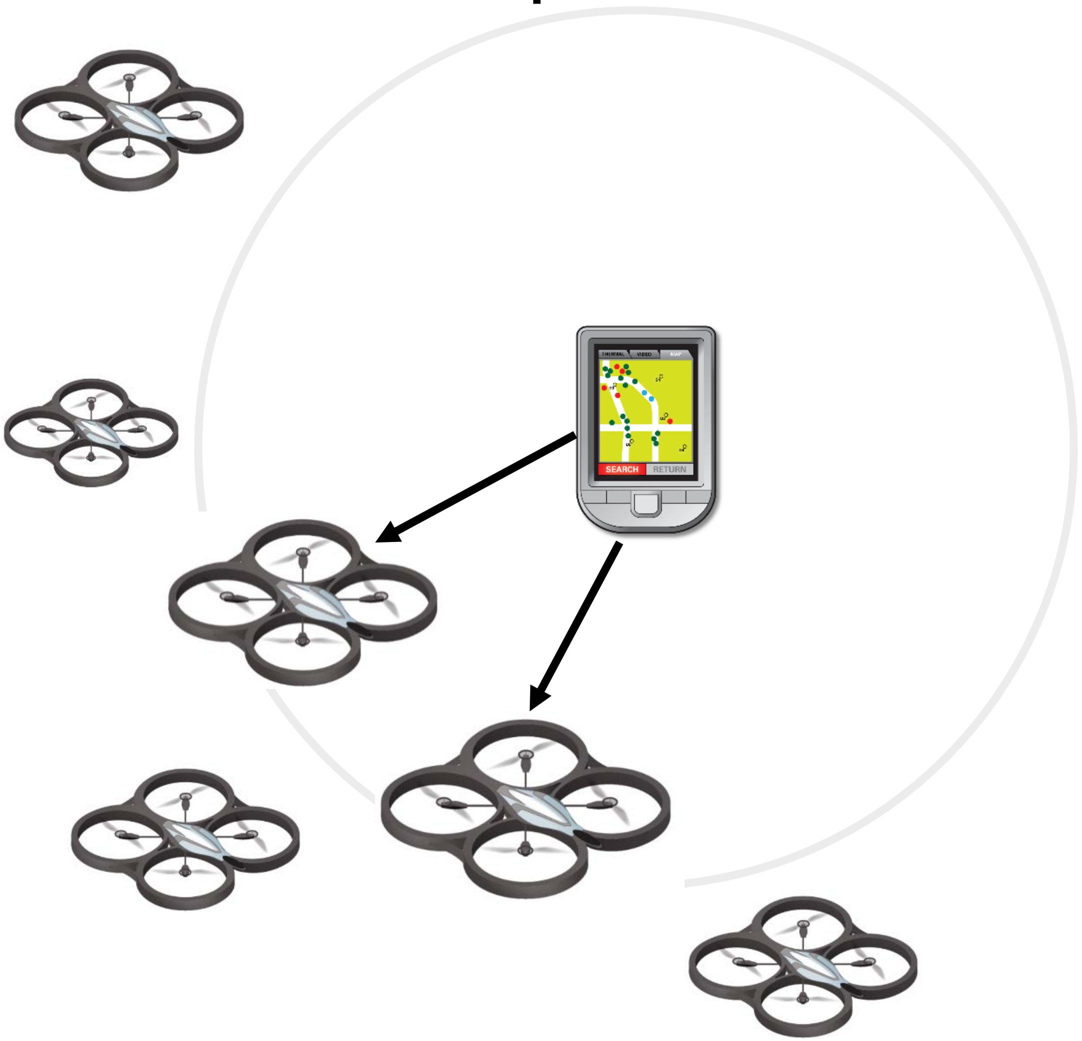






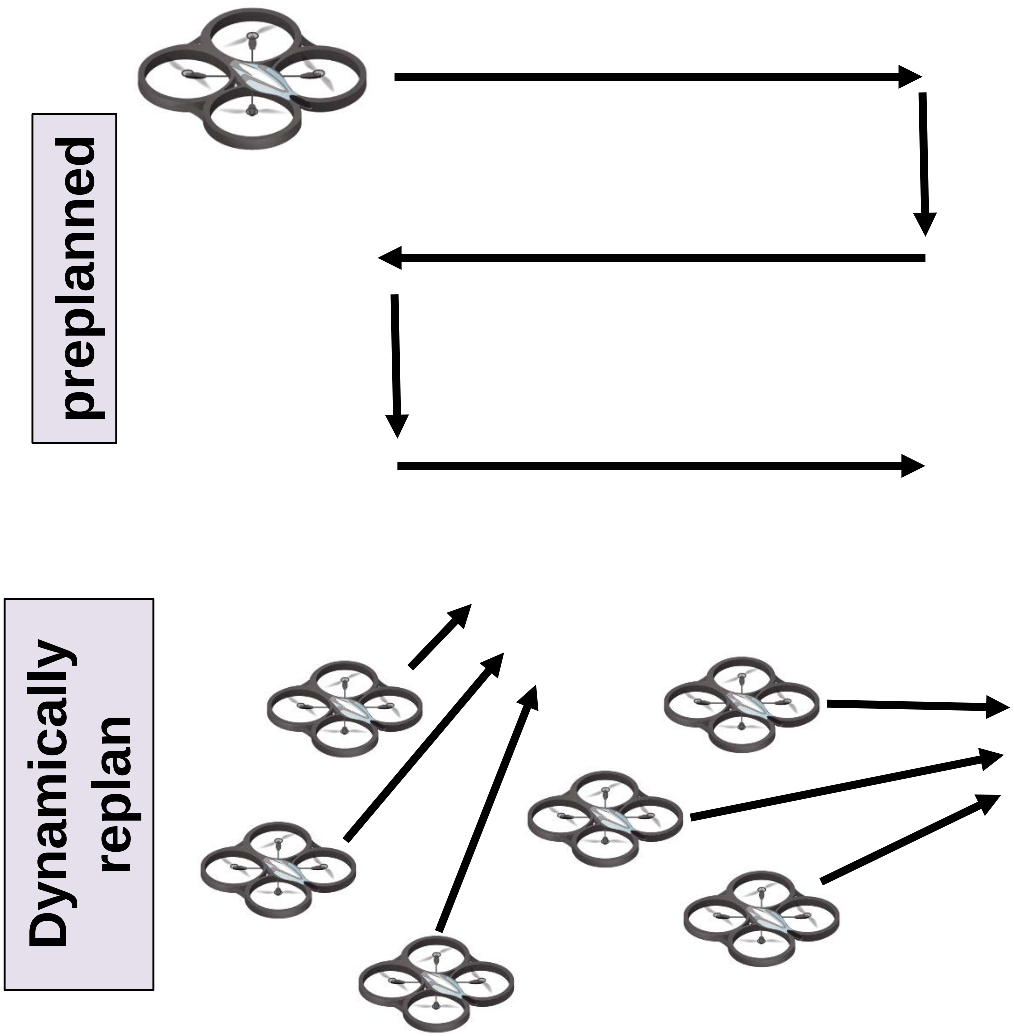


Challenge 1 & 2: A normal mission environment is very unforgiving toward centralized planners and controllers



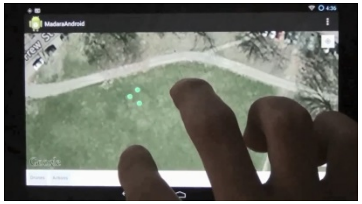
- Environment can have obstacles or natural interference in mission areas
- Adversaries can make this problem even worse with jamming
- TCP causes resends of old information, can block the AI loop, and causes AI to lag and eventually stall
- **Solution: Stop using centralized planners and blocking, reliable communication**
- **Solution: Focus on best effort with resends of important information between decentralized agents**

Challenge 3: Pre-planned AI does not deal well with adaptation. Self-interested agents don't always do well with group missions.



- Many research groups use pre-planned AI pushed to each individual agent before a task
- Adversaries can exploit this stale plan
- The mission itself may change and communication becomes a problem (see Challenge 1 and 2)
- **Solution: Focus on decentralized AI with best-effort communication that can be rebroadcasted by agents and allows important information to be resent periodically**
- **Solution: Provide mechanisms for dynamic adaptation and change via configurable middleware**

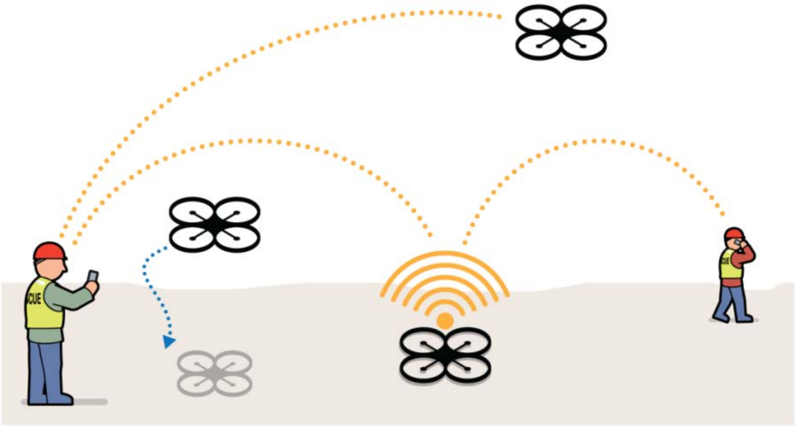
Challenge 4: AI platforms provide limited QoS, support limited hardware, architectures, and platforms



- A single bursting AI agent connected to networking middleware can overwhelm communication (see ROS)
- AI implementations are rarely hardware or platform agnostic
- **Solution: Build QoS into AI middleware as first class entity**
- **Solution: Be more portable (architecture, languages, simulators, etc.)**

Our Approach to Group Autonomy

1. Create a portable, open-sourced, decentralized operating environment for autonomous control and feedback. Focus on scalability, performance and extensibility
2. Design algorithms and tools to perform mission-oriented tasks like area coverage and multi-agent shielding of important assets
3. Integrate the operating environment into unmanned autonomous systems (UAS), simulators, platforms, smartphones, tablets, and other devices. Focus on portability.

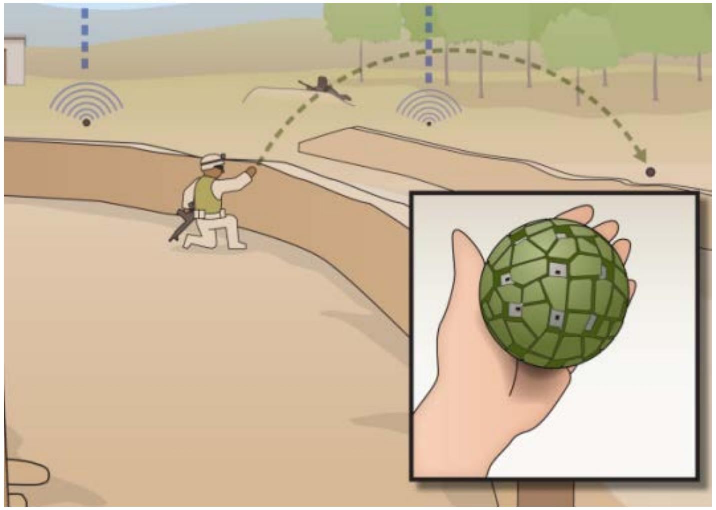


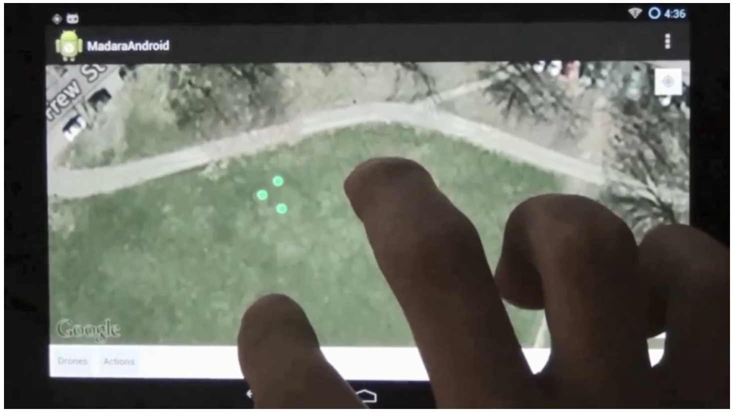
FY 2014 Technologies/Platforms

We investigated several platforms and collaborations in FY 2014, including:

- **UAVs (Parrot and 3D Robotics)**
- **Simulations (VREP)**
- **Smartphones, Tablets (Android)**
- **High precision and gps-denied positioning**

FY 2015 is focusing on autonomous swarms of 25+ boats (Platypus/CMU collaboration)





How our technologies are being used

FY 2013-2014 architecture for Drone-RK/CMU collaboration

- CMU Student Development
- SEI Staff Development

GAMS Platform for Parrot Ar.Drone 2.0

GAMS Controller

GAMS

MADARA

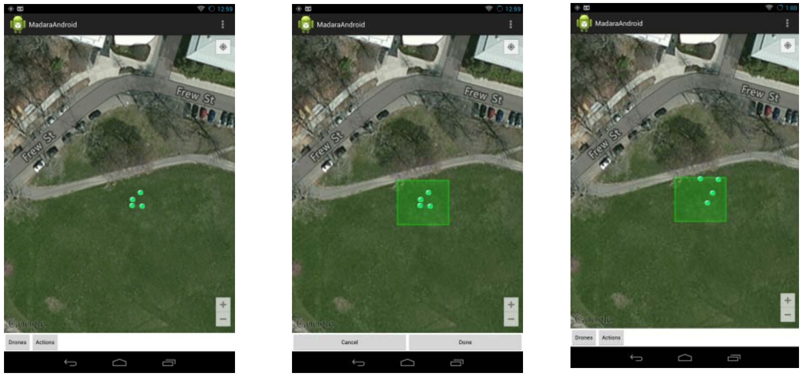
UDP Broadcast

Custom Commands

KaRL Interpreter

MADARA

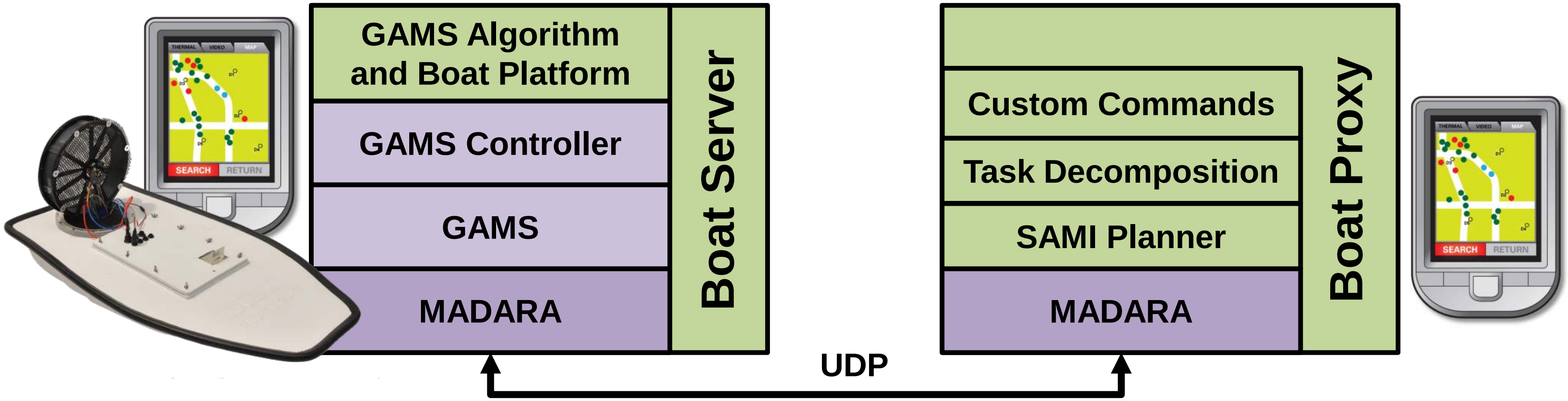
GUI/terminal



How our technologies are being used

FY 2015 architecture for Platypus/CMU collaboration

CMU Student Development
SEI Staff Development



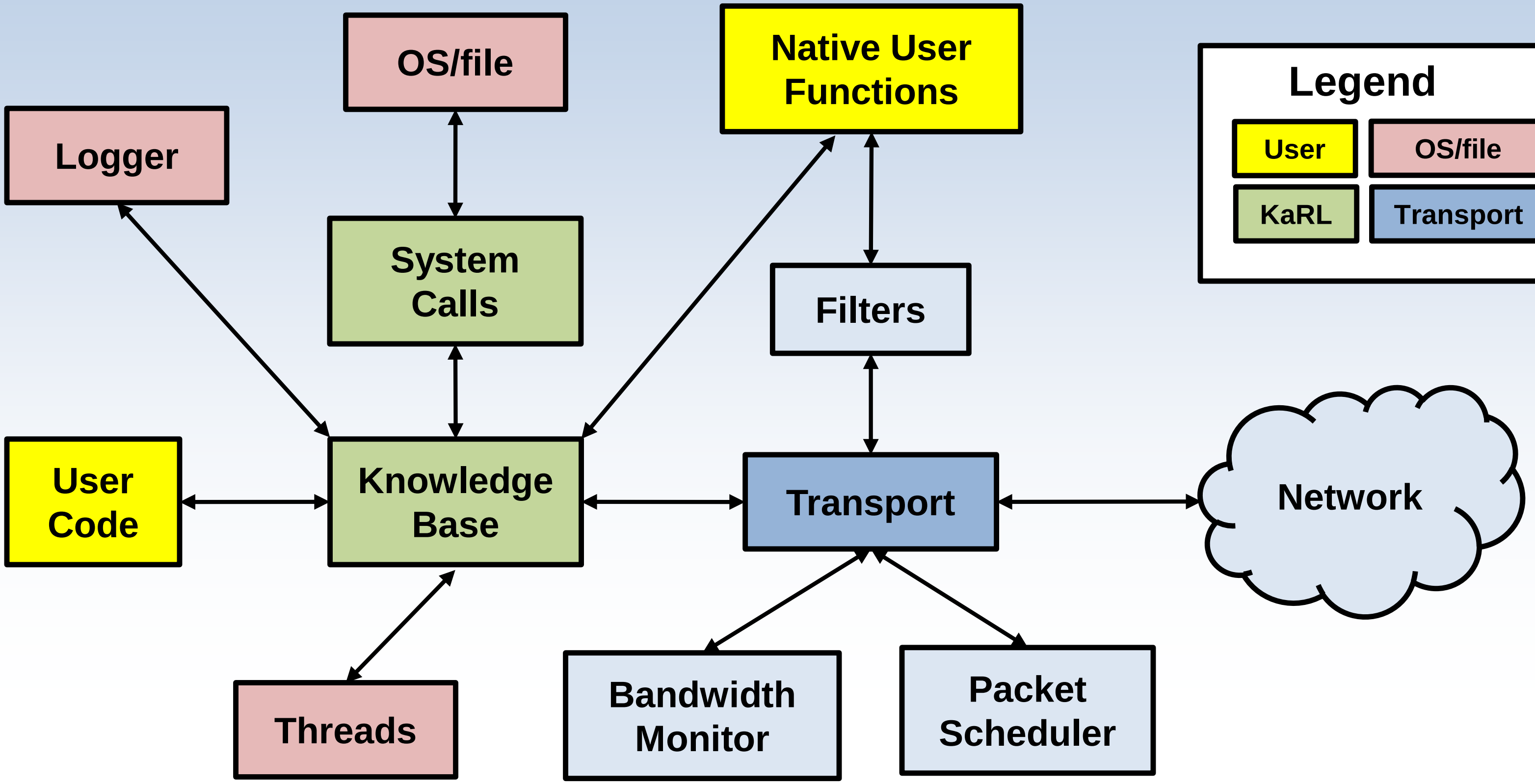
Boat Images © 2013-2014 Platypus LLC

Principles of our open-sourced middleware (MADARA and GAMS)

1. Be useful to application developers
2. Enable distributed, decentralized artificial intelligence including machine learning, reinforcement learning, fuzzy logic, and rule-based state machines
3. Be fast, small, and capable
4. Be portable to as many platforms relevant to UAS as possible
5. Provide configurable quality-of-service in all middleware features to enable developers to have more control over artificial intelligence and communication between agents
6. Be extensible to facilitate new transports, linking with external libraries, security, assurance, and consistency
7. Provide extensive documentation

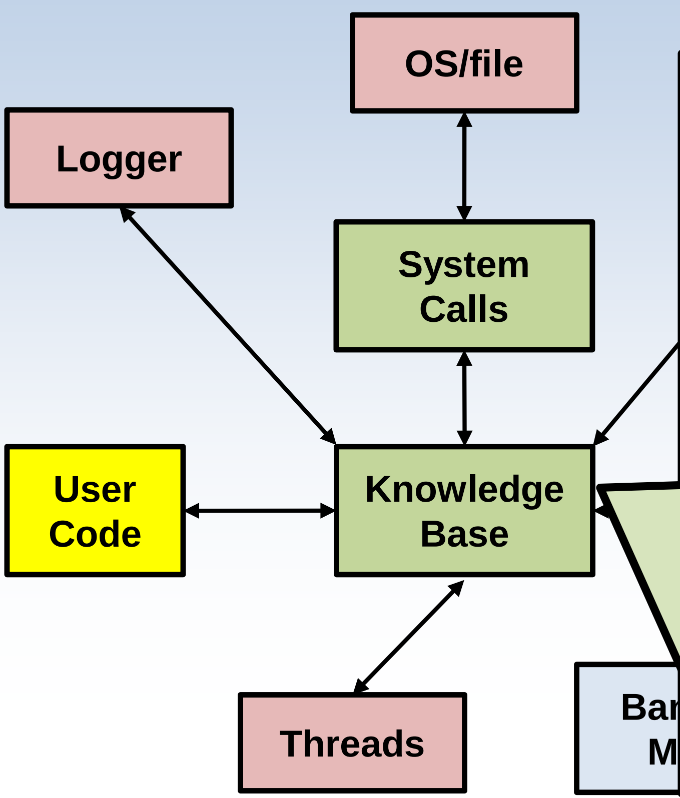
MADARA Architecture

More information, tutorials, and documentation at http://madara.googlecode.com



MADARA Architecture

More information, tutorials, and documentation at http://madara.googlecode.com

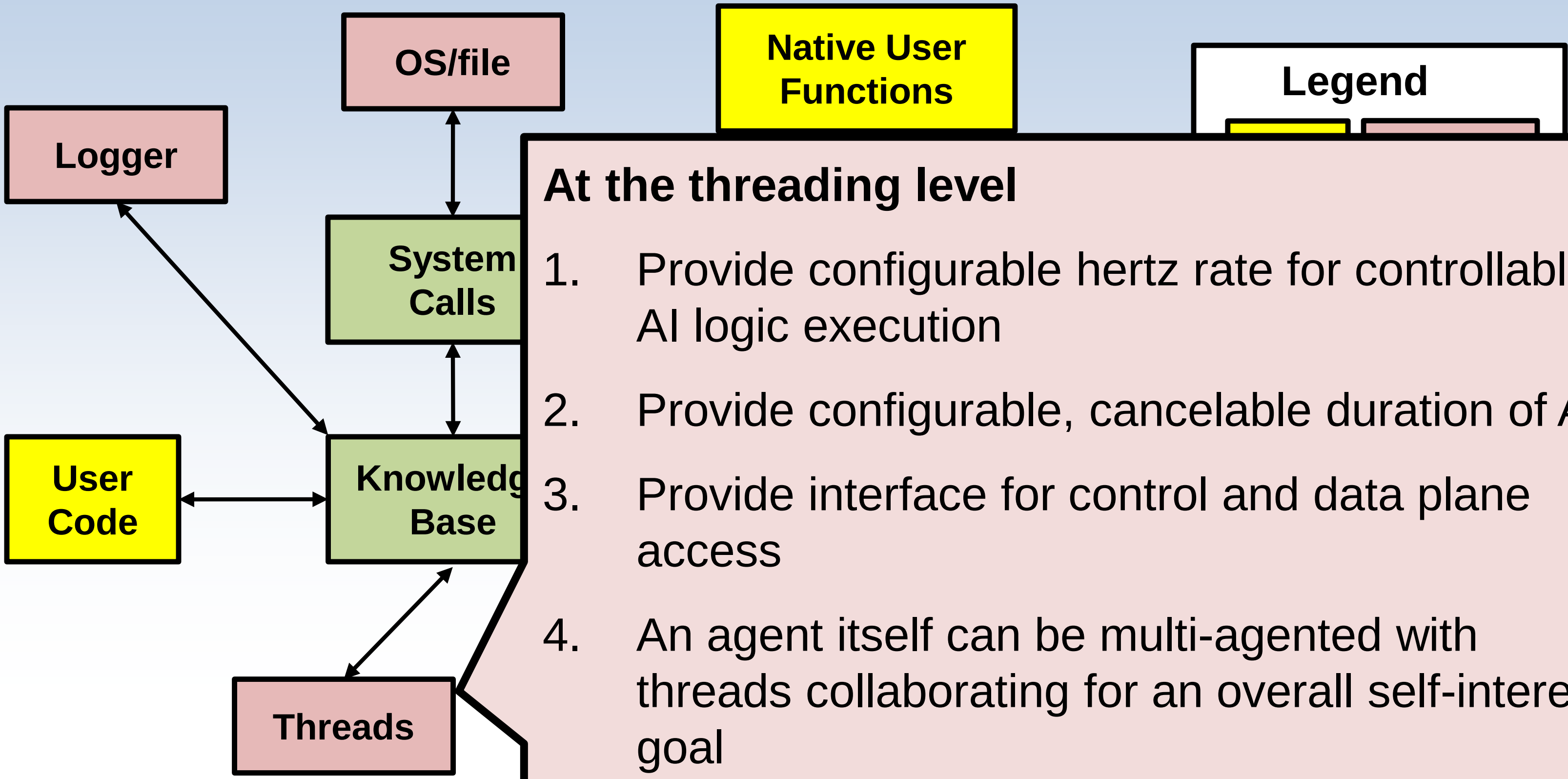


Features at the Knowledge Engine Level

1. O(1) access times via containers and knowledge references (nanosecond performance)
2. Allow dynamic changing of programs via real-time scripting language, multiple programming models (state-based and event-based)
3. Can wait for knowledge to be a certain condition, with periods and timeouts

MADARA Architecture

More information, tutorials, and documentation at http://madara.googlecode.com



At the threading level

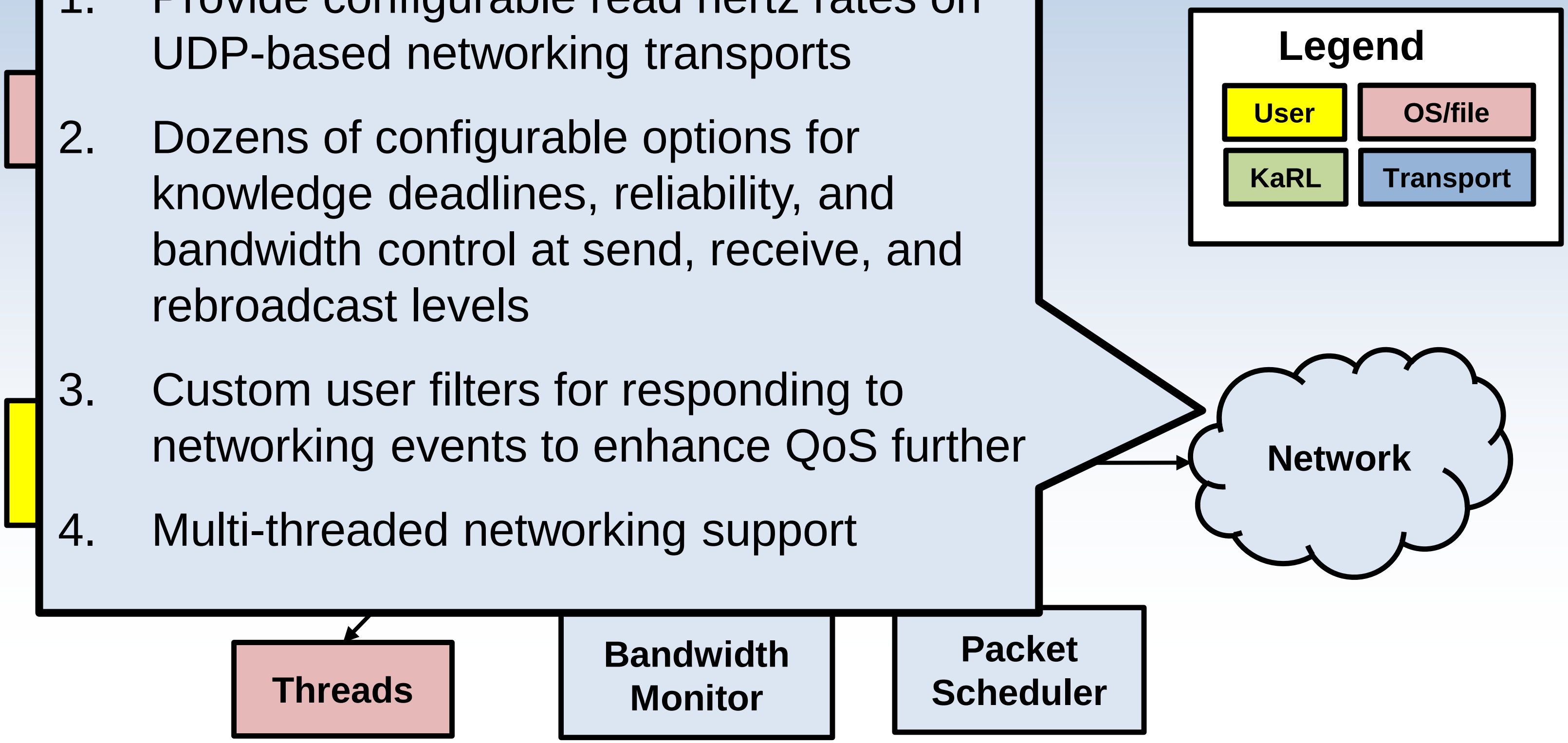
1. Provide configurable hertz rate for controllable AI logic execution
2. Provide configurable, cancelable duration of AI
3. Provide interface for control and data plane access
4. An agent itself can be multi-agented with threads collaborating for an overall self-interest goal

MADARA Architecture

Mo adara.googlecode.com

At the networking level

1. Provide configurable read hertz rates on UDP-based networking transports
2. Dozens of configurable options for knowledge deadlines, reliability, and bandwidth control at send, receive, and rebroadcast levels
3. Custom user filters for responding to networking events to enhance QoS further
4. Multi-threaded networking support



Potential Real World Questions

- How could I use MADARA for distributed machine learning?
  - The easiest method of doing this would be to connect your current learning functionality to the MADARA knowledge base by using knowledge containers

```
// create containers
containers::Integer danger (“agent.0.danger”, knowledge);
containers::Double classifier (“agent.0.utility”), knowledge);

// update knowledge base with results of machine learning
danger = learn_danger (current_state);
classifier = learn_classifier (current_state);

// aggregate updates to danger and classifier and send them
knowledge.send_modifieds ();
```

  - Another option would be to call learning functions/libraries when data is received

Potential Real World Questions

- How could I use MADARA for transfer learning?
  - The previous example does transfer the results of learning
  - A new agent could simply use another agent's learned state (e.g. danger and/or classifier in previous example)
  - A more complex solution could have a new agent use a heuristic to aggregate all agent learned state into the best known learned state

Potential Real World Questions

- How do I send reliable knowledge?
  - The default transports are best effort
  - Assumption is that important information gets resent until acknowledgments from the agents arrive
  - Reliability is unlikely to be necessary for all information (e.g., receipt of transfer learning) but very likely for commands or mission changes (e.g., enemy found, stop patrol, hide)
  - If all information is important, MADARA does support RTI and PrismTech DDS, which support reliable communication of all knowledge
    - Developers must understand what happens to a reliable transport (e.g., unbounded buffer growth) in a wireless disconnected environment

Key MADARA Features (2009-present)

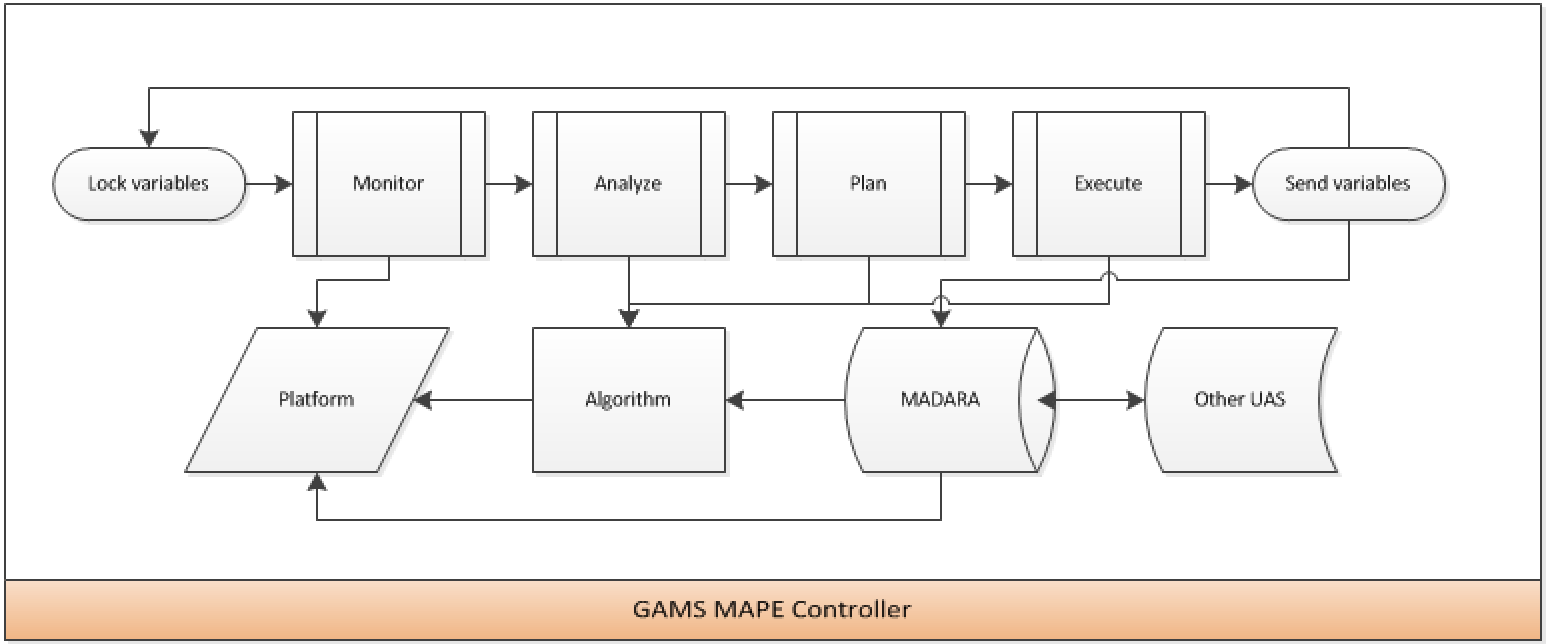
- Allows developers to write both state-based and event-based programs (or combinations of both) for distributed artificial intelligence
  - Programs can react to receive, send, or rebroadcast events
  - Programs can have deadline-enforced periodic executions, wait for certain state-based conditions to come true, or execute efficient, dynamic actions in KaRL (Knowledge and Reasoning Language)
- Provides object-oriented containers and threads as first class entities
- Supports C++, Java, Python, ARM, Intel, Windows, Linux, Android, iOS
- Supports IP multicast, broadcast, unicast, OMG DDS transports
- Enforces consistency of updates through Lamport clocks, priorities
- Extensible transport layer, filtering system, and callbacks
- Extensive documentation (guides, tutorials, doxygen)

How MADARA helps researchers and developers

- **Facilitates distributed and multithreaded programming**
  - Networking and threading is provided
  - Performance and scaling is exceptional
  - Language and architecture portability to prevent vendor lock-in and shorten transition timeframe
  - Open source. Free. Extensible.
- **Allows reseachers to focus on what is important to them**
  - Quickly code and experiment with multi-processed, multi-threaded, or multi-robot applications with dependable, portable code
  - Scale to thousands of collaborating entities in real-time

GAMS Architecture (FY 2014)

1. Built directly on top of MADARA
2. Utilizes MAPE loop (IBM autonomy construct)
3. Provides extensible platform, sensor, and algorithm support
4. Uses new MADARA feature called Containers, which support object-oriented programming of the Knowledge Base



How our technologies are being used

FY 2013-2014 architecture for Drone-RK/CMU collaboration

- CMU Student Development
- SEI Staff Development

GAMS Platform for Parrot Ar.Drone 2.0

GAMS Controller

GAMS

MADARA

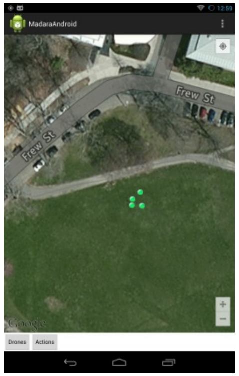
UDP Broadcast

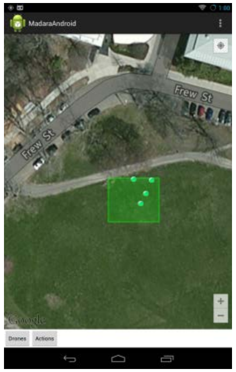
Custom Commands

KaRL Interpreter

MADARA

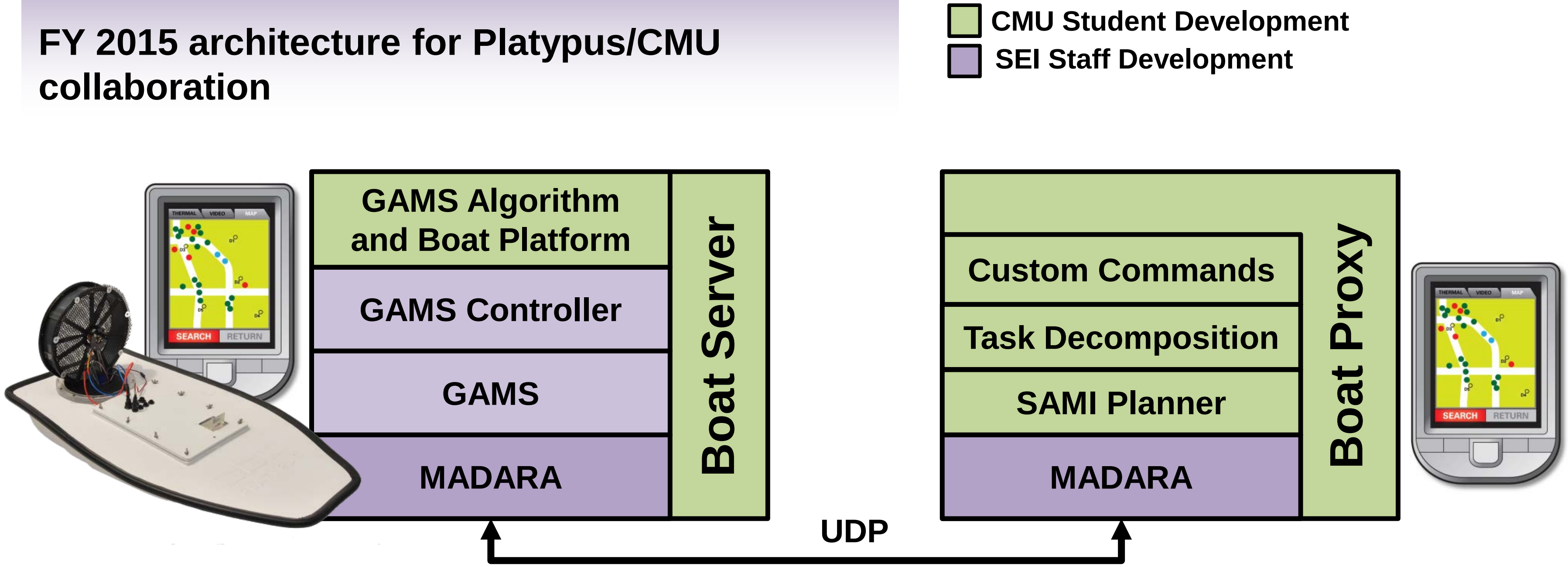
GUI/terminal





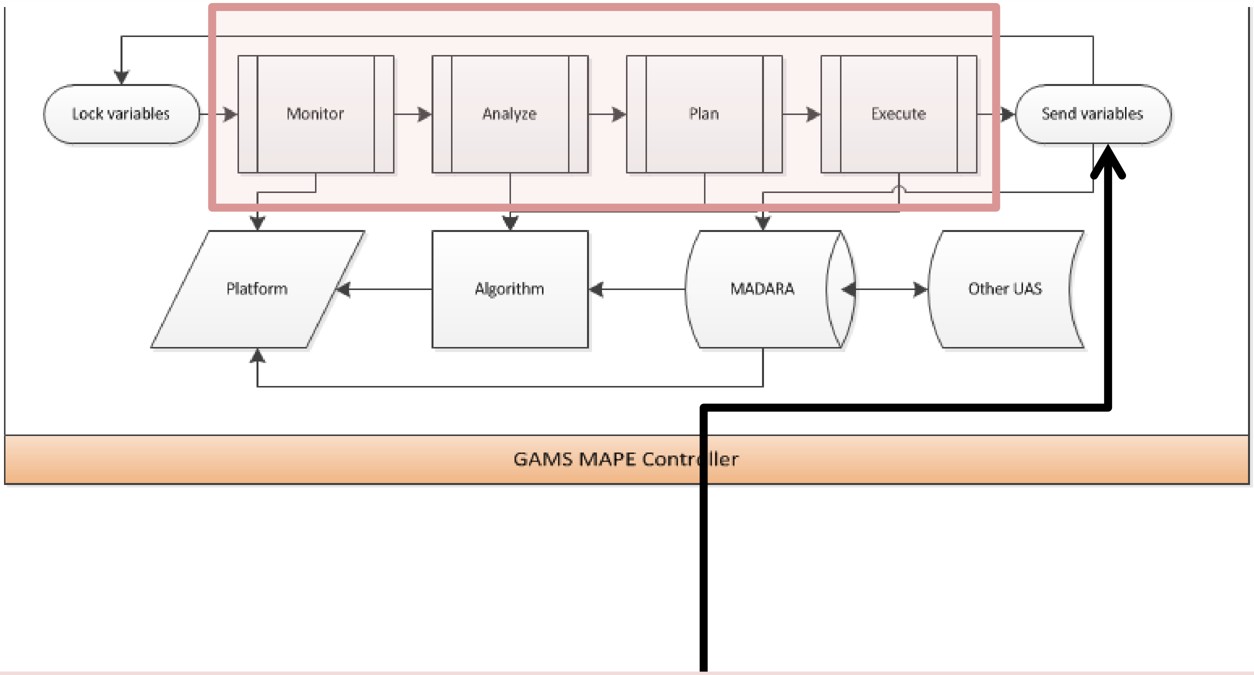
How our technologies are being used

FY 2015 architecture for Platypus/CMU collaboration



Boat Images © 2013-2014 Platypus LLC

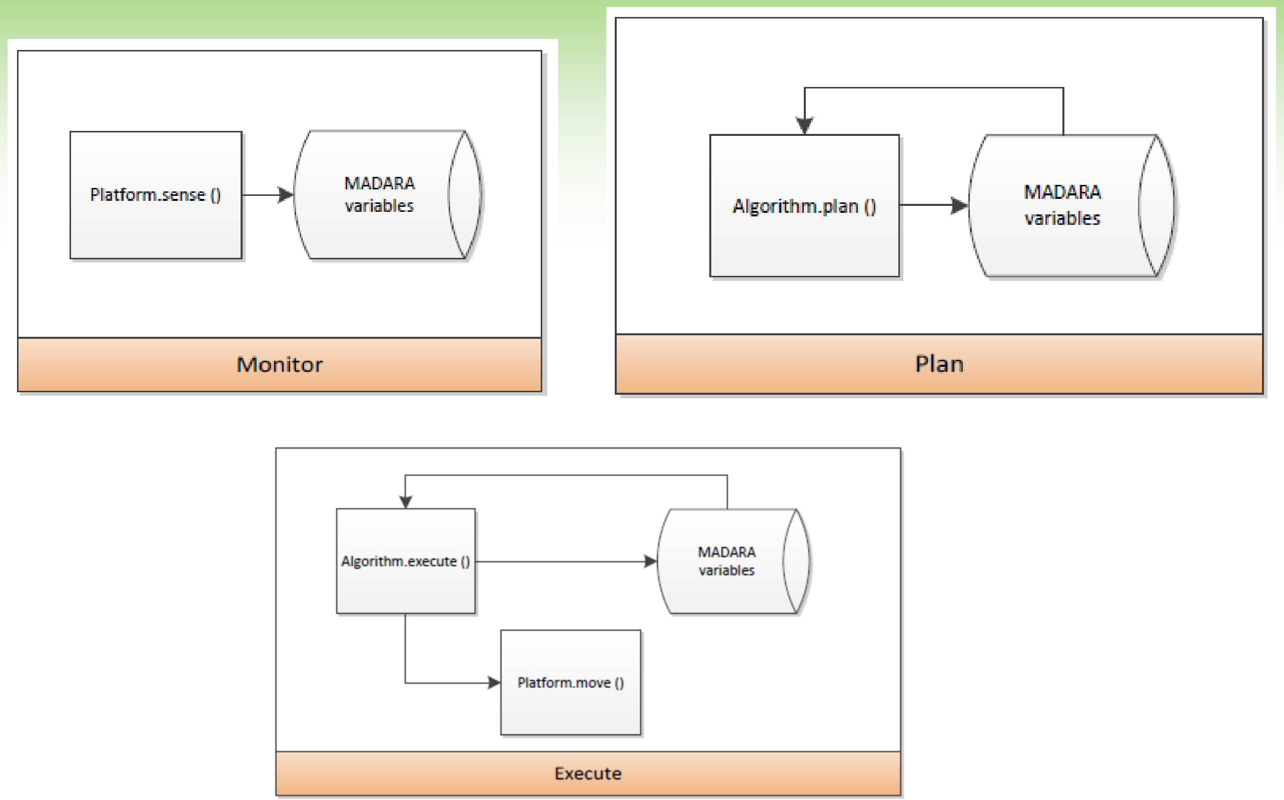
GAMS Architecture (FY 2014)



Key points:

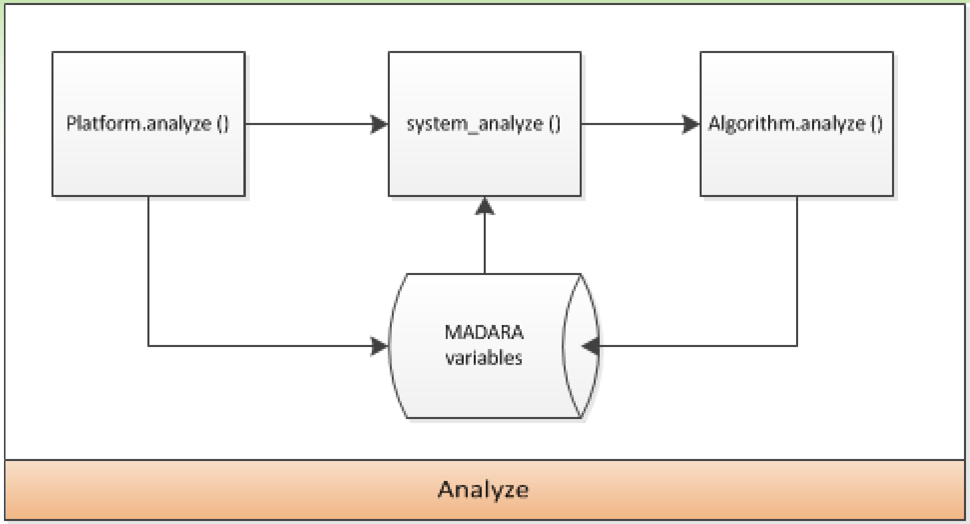
- **During the MAPE loop, the context is locked from external updates**
- **At the end of the MAPE loop, all global variable changes are aggregated together and sent to other UAS participating in the mission**

GAMS Platform and Algorithm Interactions



The Monitor, Plan, and Execute phases are pretty straight-forward

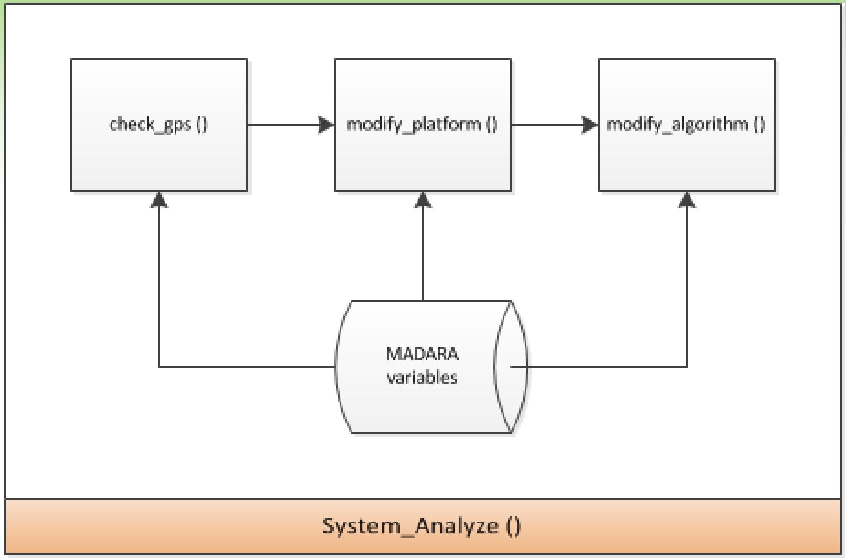
GAMS Platform and Algorithm Interactions



During the analyze phase:

1. **The platform analyzes its state and informs the rest of the GAMS system via MADARA variables**
2. **The system analyzes the platform and environment for algorithm changes**
3. **The algorithm then analyzes its state and sets appropriate MADARA variables.**

GAMS Platform and Algorithm Interactions



About system_analyze ():

1. **The platform can inform the control loop of gps-spoofing, if it has capabilities**
2. **Check_gps () is also intended to implement gps-spoof checking in software**
3. **Environmental or platform characteristics can result in changes to the platform (e.g., an arm is damaged) or algorithm (e.g., the UAS should return home)**

How to use GAMS with new platforms and algorithms

1. Extend the platform base class
   - Implement move (), land (), takeoff (), or other functions
   - Implement sense ()
   - Implement analyze ()
2. Extend the algorithm base class
   - Implement analyze ()
   - Implement plan ()
   - Implement execute ()
3. Extend the base controller class (optional)
   - Override MAPE methods
4. Use the parameterized Mape_Loop class (optional)
   - Use the define_monitor, define_analyze, etc. methods with MADARA functions

What exactly are we solving?

1. MADARA is a bit expansive in its capabilities and developers can find themselves pulled in many different directions when thinking of autonomy to implement. **GAMS provides an interface for algorithms and platforms to be added and utilized within a wireless environment**

2. **GAMS provides mechanisms for tracking platform and algorithm states and characteristics of distributed applications**, such as detection of GPS-spoofing, blocked/deadlocked conditions within algorithms, low battery, degraded sensors, etc.

3. While MADARA may support any type of distributed artificial intelligence paradigm, **GAMS provides a stable, consistent framework for group autonomous behaviors and may prove beneficial to standardization efforts for group autonomy**

New Features in FY 2015

1. **Tighter and more feature rich MADARA interactions**
   - GAMS can now be directly ran inside of MADARA thread library
   - GAMS can now run at multiple hertz speeds for sampling sensors at varying rates
   - GAMS may have separate sampling and sending hertz rates
2. **Multiple platform support in VREP and real-world**
   - VREP: Quadcopter, Ant, and possibly Boat models and platforms (Q2-3 2015)
   - Real World: Drone-RK quadcopter and Platypus boat
3. **Even more focus on scale and reliability**
4. **Distributed mission-focused algorithms that respond to environment and mission objective changes**

FY 2015 Goals and Objectives (ELASTIC Project)

1. Showcase GAMS and MADARA on 25+ real, collaborating robots
   - Focusing on Paul Scerri's robotic boats (Platypus/CMU)
   - Focus on dynamic adaptation in contested/deprived environments
2. Facilitate transition of GAMS/MADARA into DARPA or DoD Labs
3. Quantify scalability limitations
4. Identify best practices for developing distributed mission-focused autonomy applications

Closing remarks

In this talk, we've discussed

- A **distributed reasoning engine called MADARA** that provides portable, fast reasoning services for distributed artificial intelligence
- An **extensible framework called GAMS for distributed algorithms and platforms** that enables Monitor-Analyze-Plan-Execute-based distributed autonomous systems

Carnegie Mellon University

Software Engineering Institute
Carnegie Mellon

FY 2014 Open Source Release

The algorithms, tools, and middleware created at SEI are released via BSD-style licenses through the following projects:

- Multi-Agent Distributed Adaptive Resource Allocation (MADARA) for the distributed OS layer: http://madara.sourceforge.net/
- Group Autonomy for Mobile Systems (GAMS) for the algorithms and UIs: http://gams-cmu.googlecode.com
- Model Checking for Distributed Applications (MCDA) http://mcda.googlecode.com
- Drone-RK for the UAV device drivers: http://www.drone-rk.org
- Contact: jredmondson@sei.cmu.edu

SEI Project Members

James Edmondson

Sagar Chaki

David Kyle

HCCPS/DART Group

CMU Project Members

Paul Scerri

Nate Brooks

Christopher Tomaszewski

Vanderbilt Students

Anton Dukeman (CS)

Carnegie Mellon University

Software Engineering Institute
Carnegie Mellon