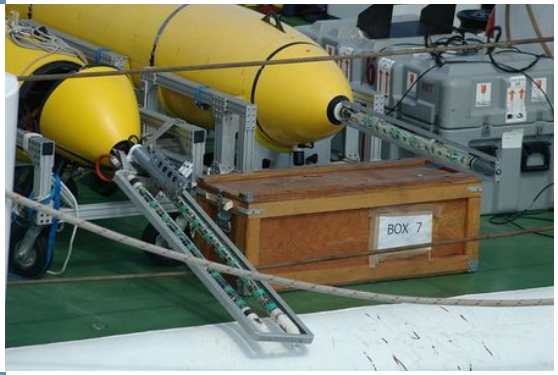


New Features and Applications in MOOS-IvP 13.5

Michael R. Benjamin
MIT Dept. of Mechanical Engineering
Computer Science and Artificial Intelligence Lab
Laboratory for Autonomous Marine Sensing Systems

mikerb@mit.edu

Acknowledgements

- Office of Naval Research (ONR 311) has sponsored MOOS-IvP for nearly 10 years.
  Dr. Don Wagner, Dr. Behzad Kamgar-Parsi, Dr. Wen Masters.
- Battelle has sponsored, beginning in 2010, MOOS-IvP, MOOS, the development of our autonomy course at MIT, the purchase of the Kingfisher platforms, and the build-out of our new Charles River facilities, and the development of our regression testing code.
  Dr. Rob Carnes

Faculty collaborators:

Prof. Henrik Schmidt
MIT MechE

Prof. John Leonard
MIT MechE/CSAIL

Prof. Paul Newman
Oxford

My group:

Alon Yaari
MIT Research Scientist

LT Arthur Anderson
MIT Navy PhD student

LT Kyle Woerner
MIT Navy MS student

Michael Novitzky
Georgia Tech Visiting PhD

Objectives and Motivations

- Algorithms / Software Functionality
- Documentation
- Verification and Validation
- Competitions
- Lab sequences
- Example Missions
- Online Tutorials / Lectures

Objectives and Motivations

- Algorithms / Software Functionality (Stand-by Helm, uField Toolbox, AppCasting)
- Documentation (www.moos-ivp.org/download)
- Verification and Validation
- Competitions (Hazard Search Competition, Hunter-Prey Competition)
- Lab sequences (http://oceanai.mit.edu/2680)
- Example Missions (www.moos-ivp.org/download)
- Online Tutorials / Lectures (http://oceanai.mit.edu/2680)

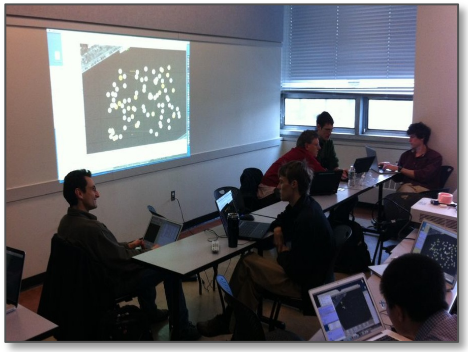
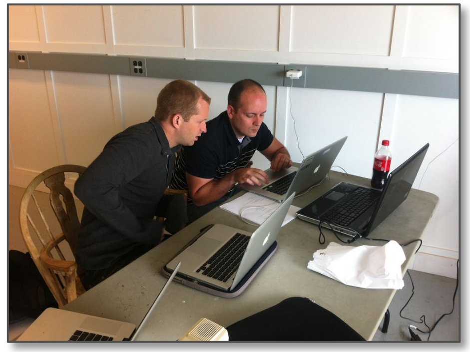
2.680 Marine Autonomy, Sensing and Communications

(Funded by Battelle)

- Effective autonomy can compensate for limits in sensing and communications.
- Effective communications can compensate for limits in sensing and autonomy.

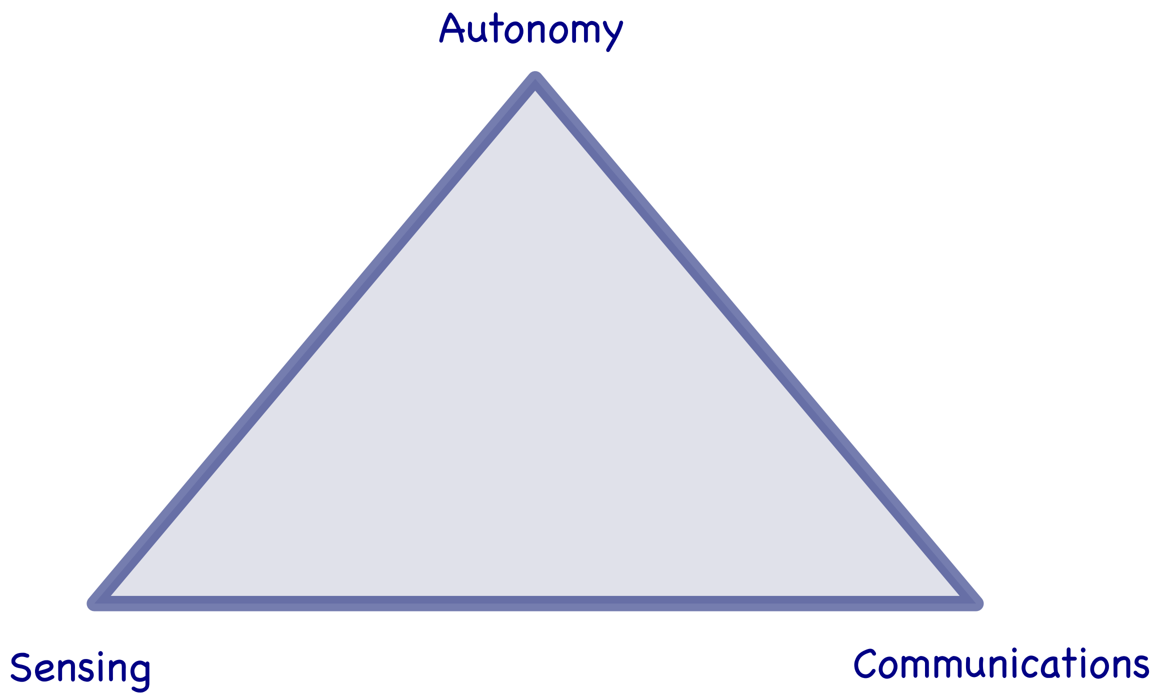
- Understanding the ocean with marine robotic platforms.
- Educational focus is on autonomous decision making for marine robotic platforms.
- Students use an extensive MIT-developed autonomy and simulation codebase.
- On-board sensor-processing, inter-vehicle communications.



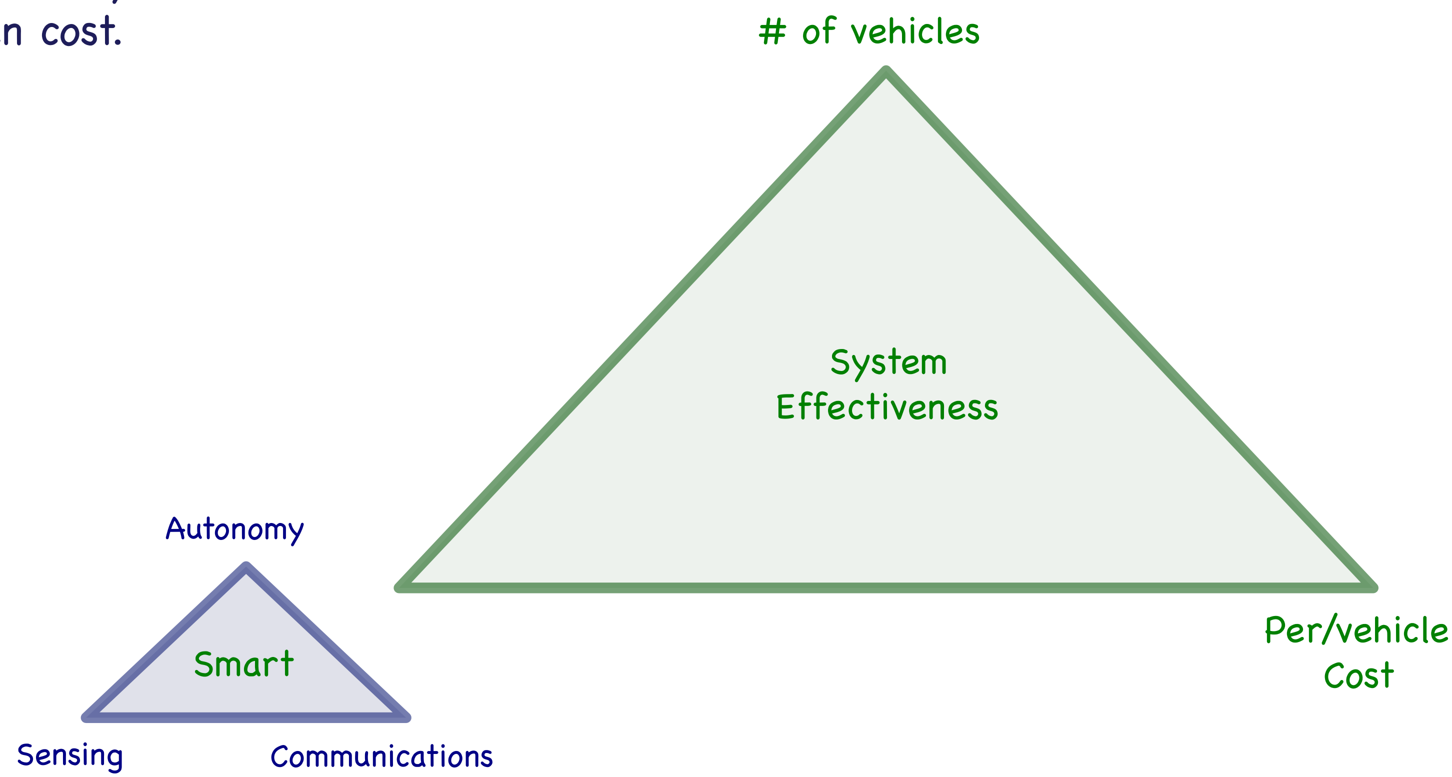

Class focus: Autonomy, Sensing and Communications

- Effective autonomy can compensate for limits in sensing and communications.
- Effective communications can compensate for limits in sensing and autonomy.



- A "smarter" vehicle means more can be done with fewer vehicles.
- A "smarter" vehicle means more can be done with a less capable (cheaper) vehicle.

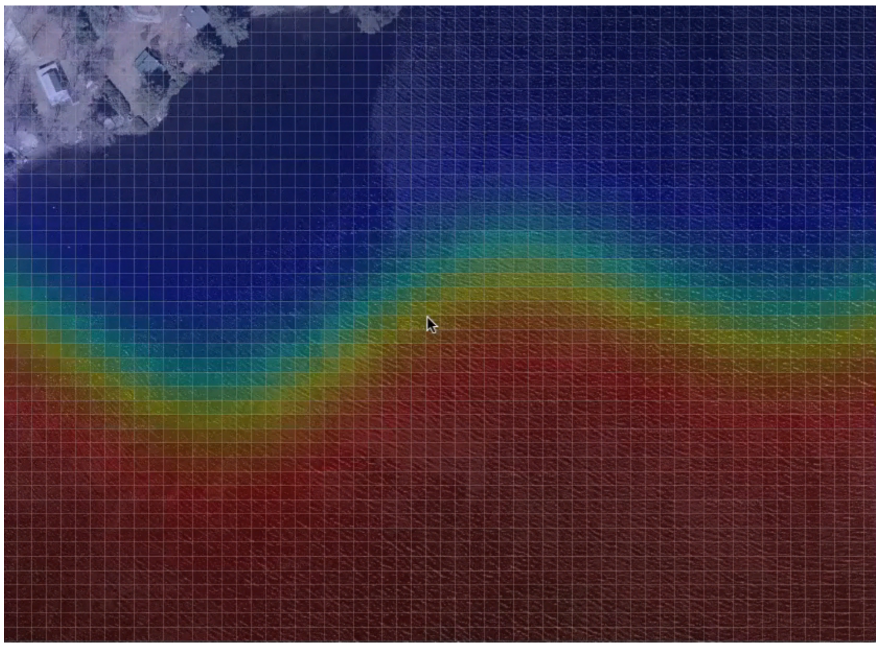
Bottom line is system effectiveness for a given cost.



In-Class Competitions

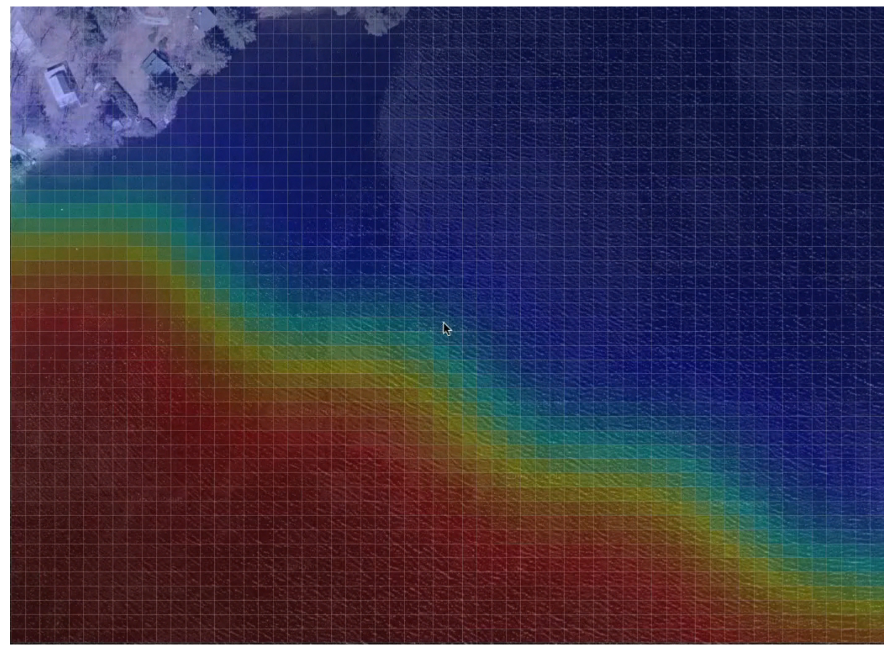
Autonomous Front Detection

Objective: Detect and characterize a moving temperature gradient
Output: Parameters of the gradient: amplitude, period, angle, wavelength, offset, alpha, beta, temperature-north, temperature-south.
Given: A simulated CTD sensor and simulated annealer for parameter estimation.
Assignment: Build one or more vehicle behaviors for maneuvering the vehicle to collect CTD measurements.



```
offset = -90
angle  = 5
amplitude = 20
period = 200
wavelength = 200
```

```
alpha = 400
beta = 20
temp_north = 20
temp_south = 25
```



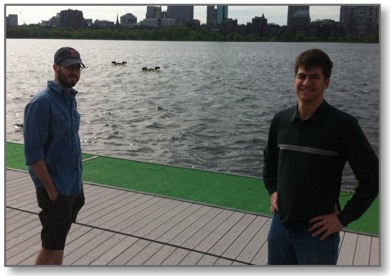
```
offset = -90
angle  = -25
amplitude = 7
period = 75
wavelength = 100
```

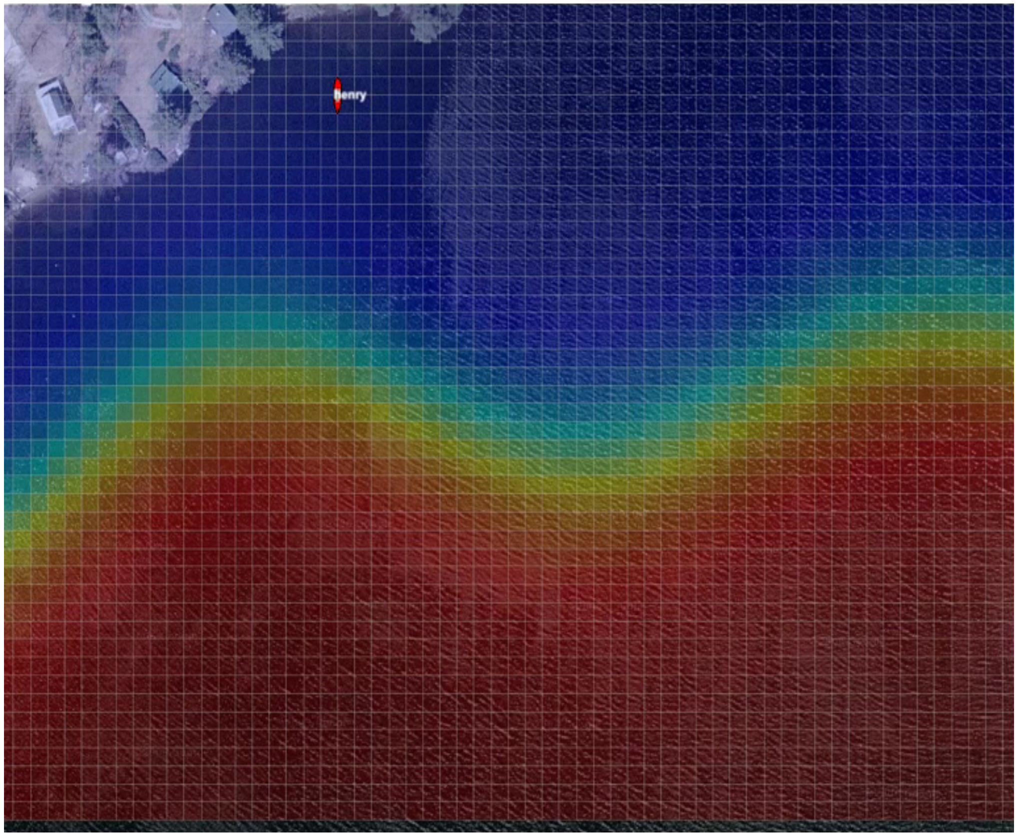
```
alpha = 400
beta = 20
temp_north = 20
temp_south = 25
```

Autonomous Front Detection

Student Competition

Team Entry:
Rob Truax,
Isaac Evans





• Detection using three coordinated behaviors
• Combined using multi-objective optimization

Behavior #1: BHV_GoSideways
- Desired heading along decision line, whichever direction is closer.

Behavior #2: BHV_WaveFollow
- Constant desired speed
- Heading toward hot or cold with small offset
- Bang-bang setpoint controller to crisscross wavefront.

Behavior #3: BHV_RubberBand
- Holds vehicle inside the operation area

Autonomous Front Detection

Student Competition

Team Entry:
Rob Truax,
Isaac Evans

- Detection using three coordinated behaviors
- Combined using multi-objective optimization

Behavior #1: BHV_GoSideways
- Desired heading along decision line, whichever direction is closer.

Behavior #2: BHV_WaveFollow
- Constant desired speed
- Heading toward hot or cold with small offset
- Bang-bang setpoint controller to crisscross wavefront.

Behavior #3: BHV_RubberBand
- Holds vehicle inside the operation area

MIT Autonomy Lab

At the MIT Sailing Pavilion

On-campus, nearly year-round facility

- Used by MIT 2.680 students, and graduate research for several faculty
- Yearly fees support by MIT Mechanical Engineering and Battelle
- Lab Equipment and vehicles funded by Battelle

MIT Autonomy Lab

At the MIT Sailing Pavilion

On-campus, nearly year-round facility

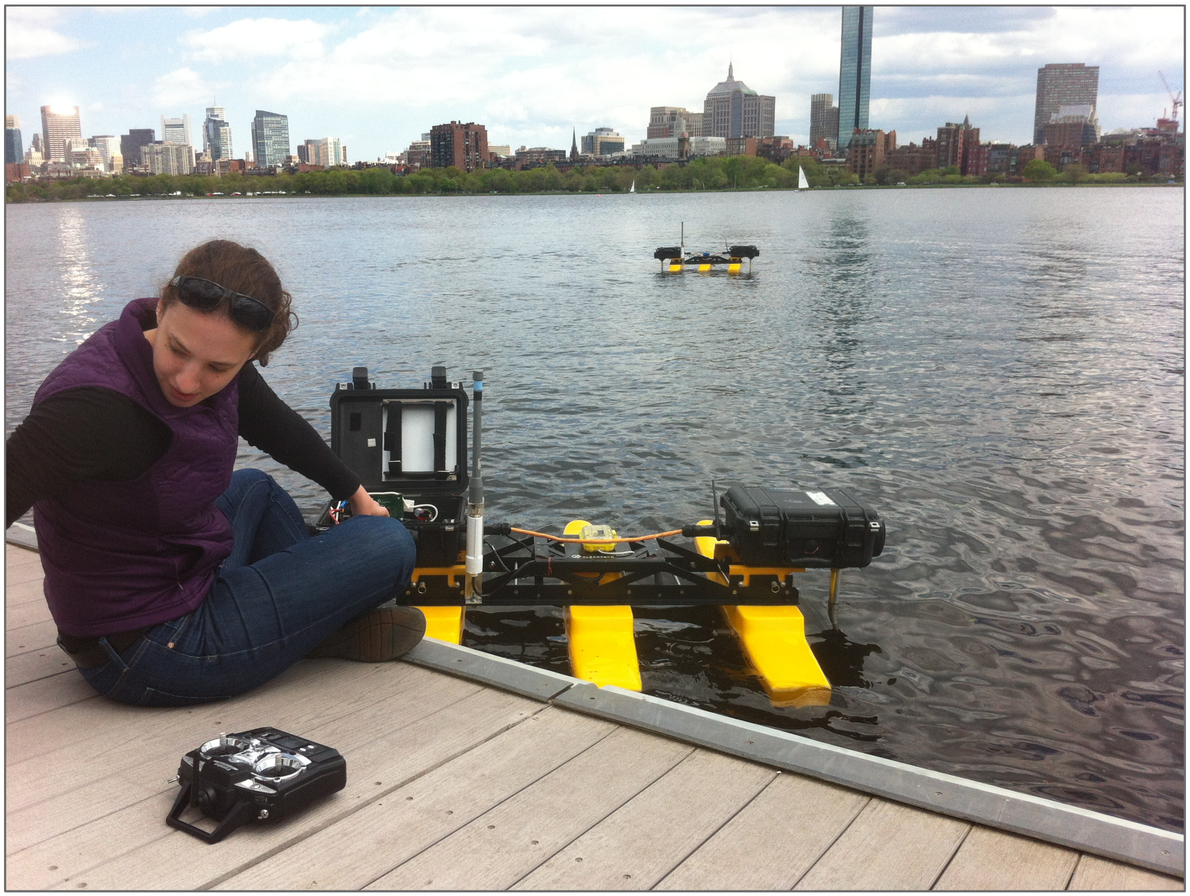
- Used by MIT 2.680 students, and graduate research for several faculty
- Yearly fees support by MIT Mechanical Engineering and Battelle
- Lab Equipment and vehicles funded by Battelle

MIT Autonomy Lab

At the MIT Sailing Pavilion

On-campus, nearly year-round facility

- Used by MIT 2.680 students, and graduate research for several faculty
- Yearly fees support by MIT Mechanical Engineering and Battelle
- Lab Equipment and vehicles funded by Battelle



MIT Autonomy Lab

At the MIT Sailing Pavilion

On-campus, nearly year-round facility

- Used by MIT 2.680 students, and graduate research for several faculty
- Yearly fees support by MIT Mechanical Engineering and Battelle
- Lab Equipment and vehicles funded by Battelle

MIT Autonomy Lab

At the MIT Sailing Pavilion

On-campus, nearly year-round facility

- Used by MIT 2.680 students, and graduate research for several faculty
- Yearly fees support by MIT Mechanical Engineering and Battelle
- Lab Equipment and vehicles funded by Battelle

Outline

- Objectives and Motivations
- The Marine Autonomy Courseware

  Lectures, Labs, Documentation at http://oceanai.mit.edu/2680
- The uField Toolbox
- AppCasting
- Changes / Additions to the Helm

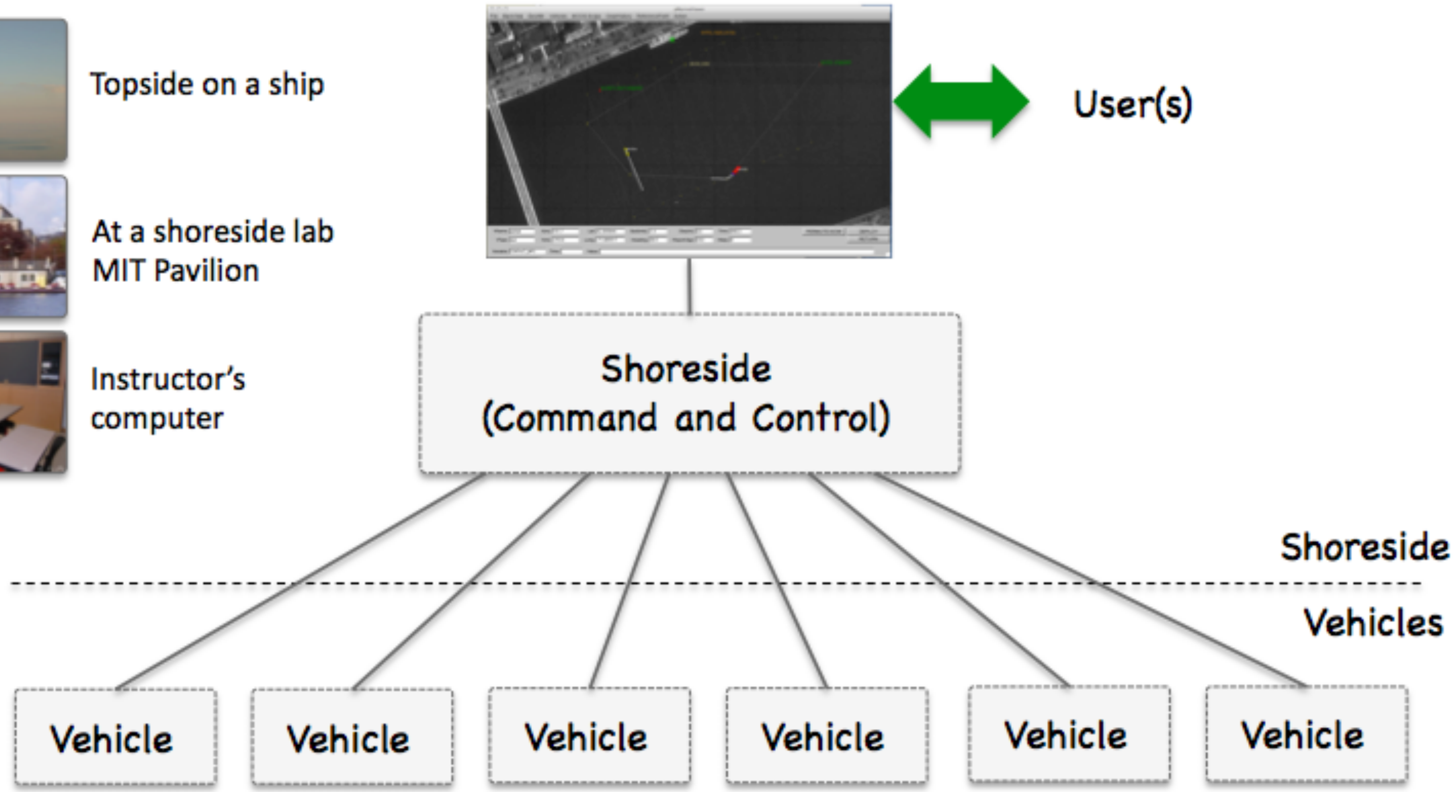
The Shoreside/Vehicle Topology

"Shoreside" could be:

Topside on a ship

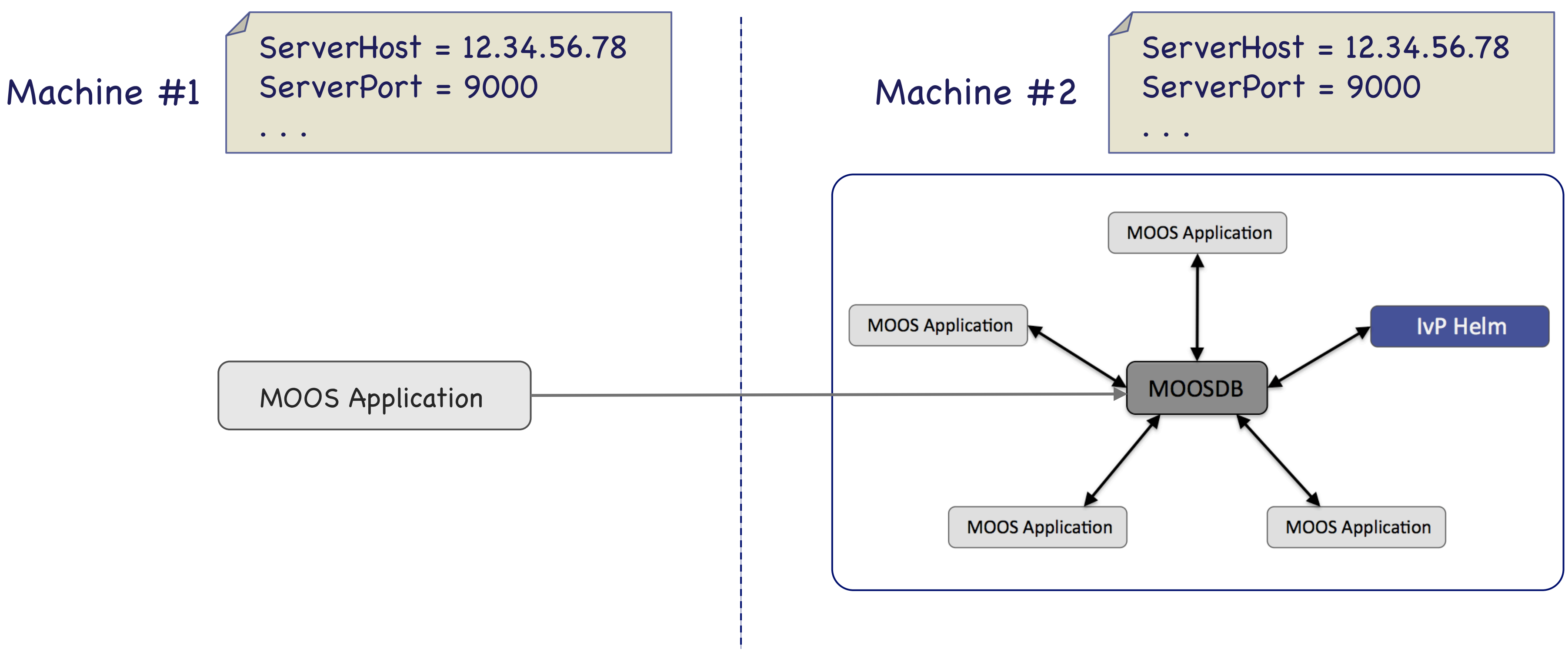
At a shoreside lab
MIT Pavilion

Instructor's
computer



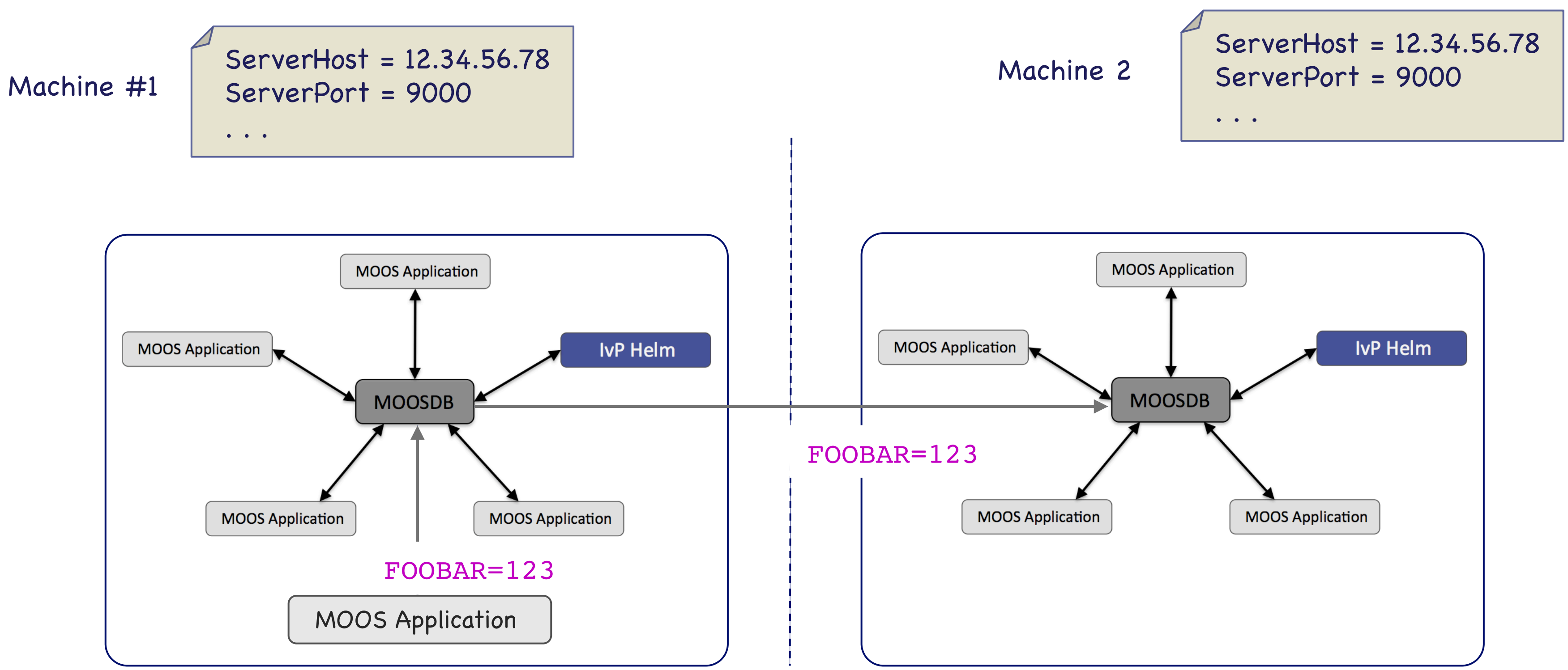
Communications Between Machines / Vehicles

We've seen in our labs that MOOS apps do not necessarily have to be on the same physical machine running the MOOSDB.



Communications Between Machines / Vehicles

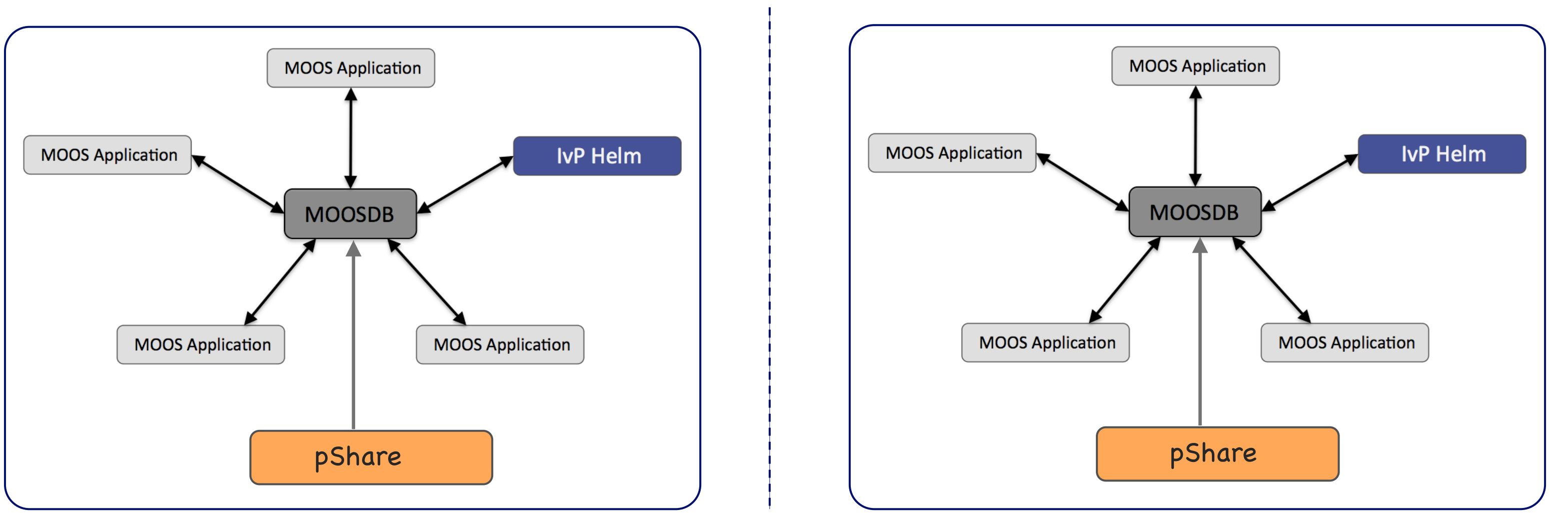
How do we get two MOOSDB's (communities) to talk to each other?



When the two machines are on the same network, we can use pShare.

Inter MOOSDB Communications with pShare

We use pShare for communications between two MOOS communities on the same network.



The pShare app is launched on *both* machines as part of their respective communities.

pShare Configuration

We use pShare for communications between two MOOS communities on the same network.

The port *we* are listening on

Name on target machine

```
ProcessConfig = pShare
{
  input  = route=localhost:9201

  output = src_name=VIEW_POLYGON, route=localhost:9202, dest_name=POLYGON
}
```

Name of variable locally

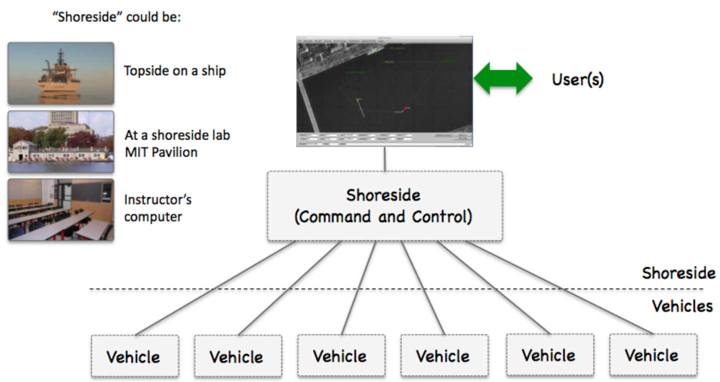
IP address of target machine

Port on which pShare is listening in the target community.

The uField Toolbox
(Overview)

The uField Toolbox is:

- A collection of about a dozen MOOS applications, each a Utility for Fielding multiple vehicles with a shoreside/topside command-and-control MOOS Community.



- All applications are documented in the MOOS-IvP Tools document, online.
  http://oceanai.mit.edu/moos-ivp-pdf/moosivp-tools.pdf

The uField Toolbox is comprised of three general capabilities:

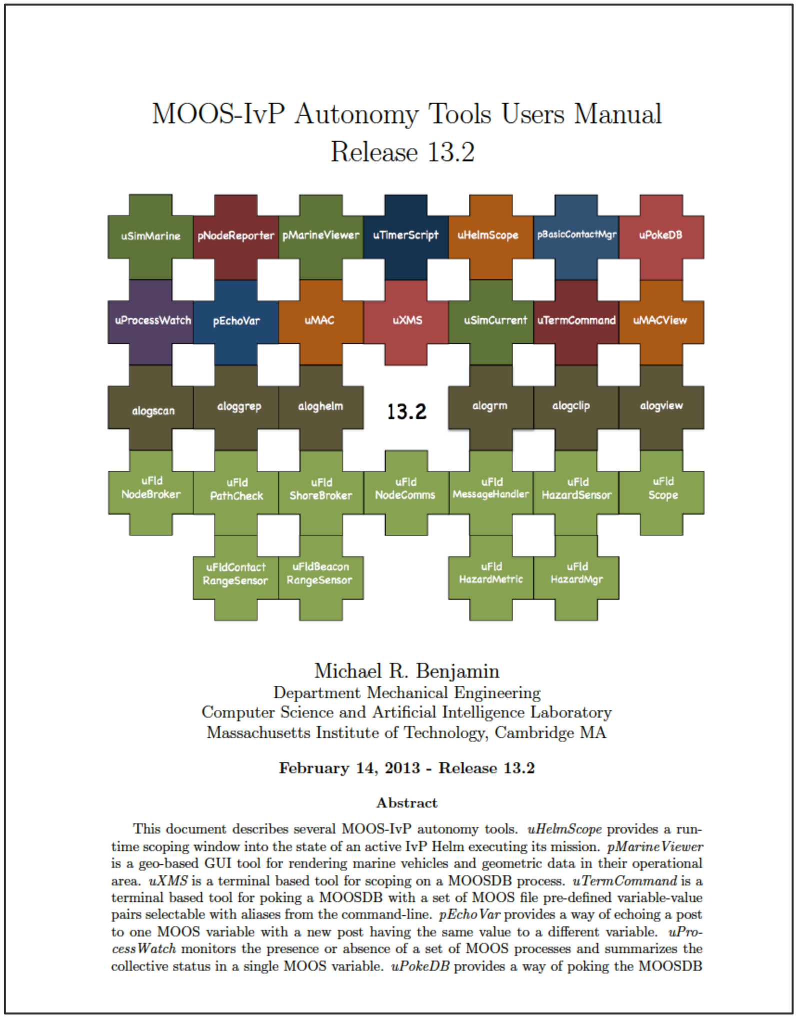
1. Facilitation of Inter MOOSDB Share configuration
2. Simulation of Inter-Vehicle Messaging
3. Sensor Simulation

The uField Toolbox

(Overview)

All applications are documented in the MOOS-IvP Tools document, online.

http://oceanai.mit.edu/moos-ivp-pdf/moosivp-tools.pdf



MOOS-IvP Autonomy Tools Users Manual
Release 13.2

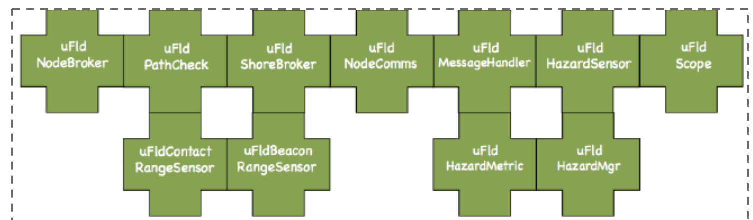
Michael R. Benjamin
Department Mechanical Engineering
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology, Cambridge MA

February 14, 2013 - Release 13.2

Abstract

This document describes several MOOS-IvP autonomy tools. *uHelmScope* provides a run-time scoping window into the state of an active IvP Helm executing its mission. *pMarineViewer* is a geo-based GUI tool for rendering marine vehicles and geometric data in their operational area. *uXMS* is a terminal based tool for scoping on a MOOSDB process. *uTermCommand* is a terminal based tool for poking a MOOSDB with a set of MOOS file pre-defined variable-value pairs selectable with aliases from the command-line. *pEchoVar* provides a way of echoing a post to one MOOS variable with a new post having the same value to a different variable. *uProcessWatch* monitors the presence or absence of a set of MOOS processes and summarizes the collective status in a single MOOS variable. *uPokeDB* provides a way of poking the MOOSDB

(uField Toolbox Apps)



The uField Toolbox

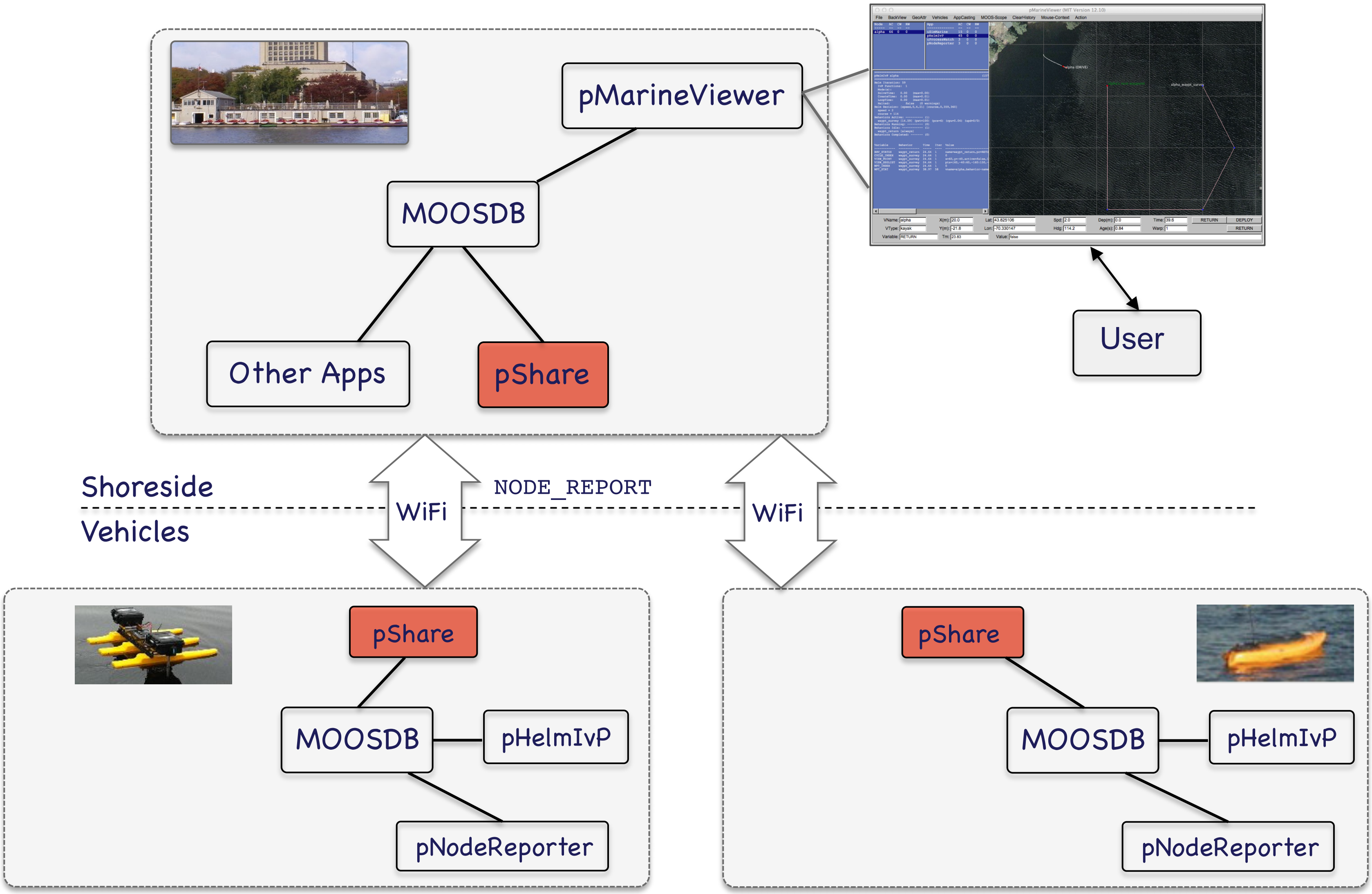
(Overview)

The uField Toolbox is comprised of three general capabilities:

1. Facilitation of Inter MOOSDB Share configuration
   - pHostInfo
   - uFldNodeBroker
   - uFldShoreBroker
2. Simulation of Inter-Vehicle Messaging
   - uFldNodeComms
   - uFldMessageHandler
3. Sensor Simulation
   - uFldHazardSensor
   - uFldHazardMgr
   - uFldHazardMetric
   - uFldContactRangeSensor
   - uFldBeaconRangeSensor

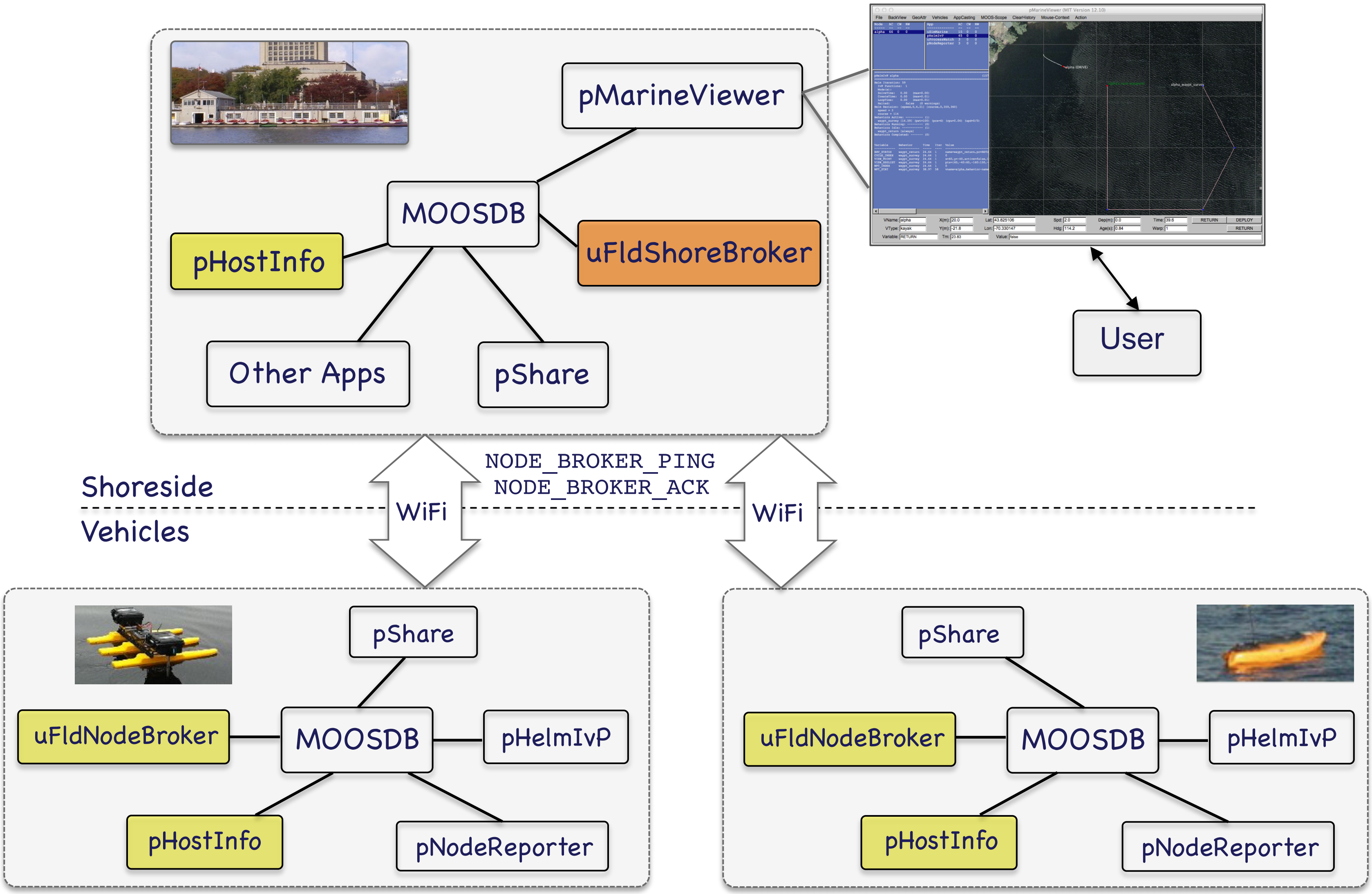
The uField Toolbox

Inter-MOOSDB sharing needs to be configured:



The uField Toolbox

We want this to be as automatic as possible.

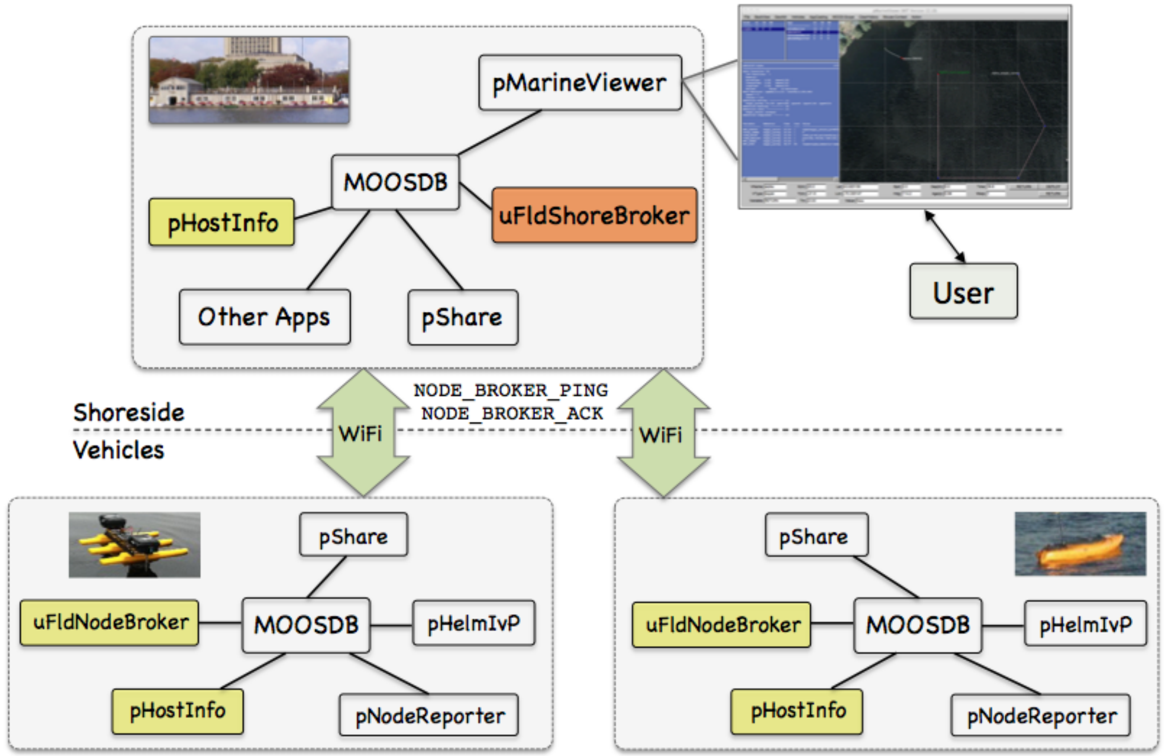


The uField Toolbox

We want this to be as automatic as possible.

pHostInfo

A MOOS app for automatically determining the local machines IP address, and publishing it to the MOOSDB

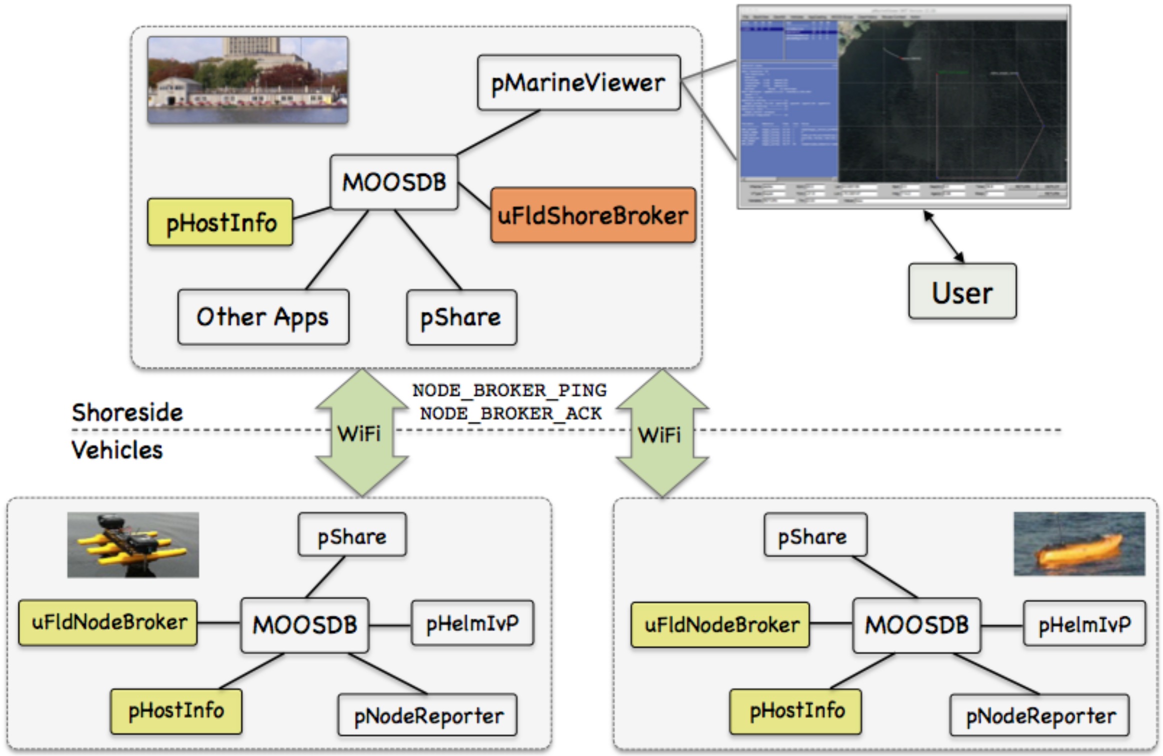


The uField Toolbox

We want this to be as automatic as possible.

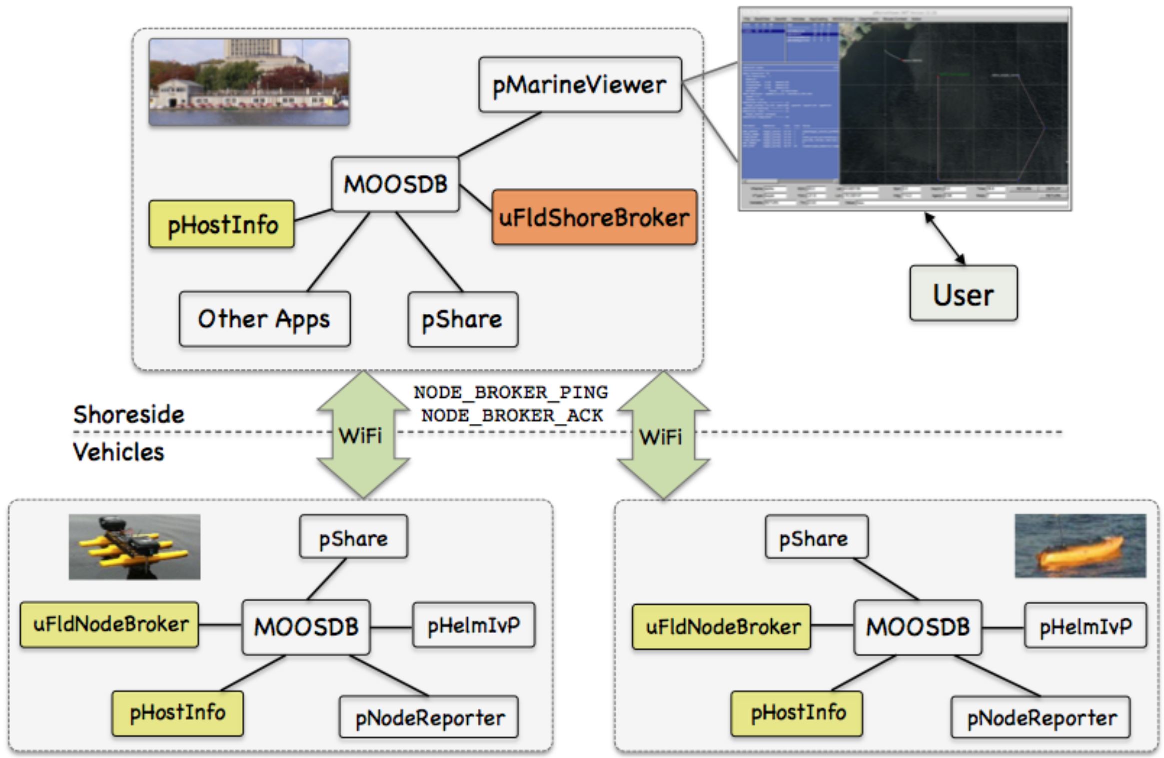
uFldNodeBroker

A MOOS app for
- finding a shore side,
- determining it's IP address and pShare input route,
- Auto-configuring its own local pShare outgoing route



The uField Toolbox

We want this to be as automatic as possible.



uFldShoreBroker

A MOOS app for

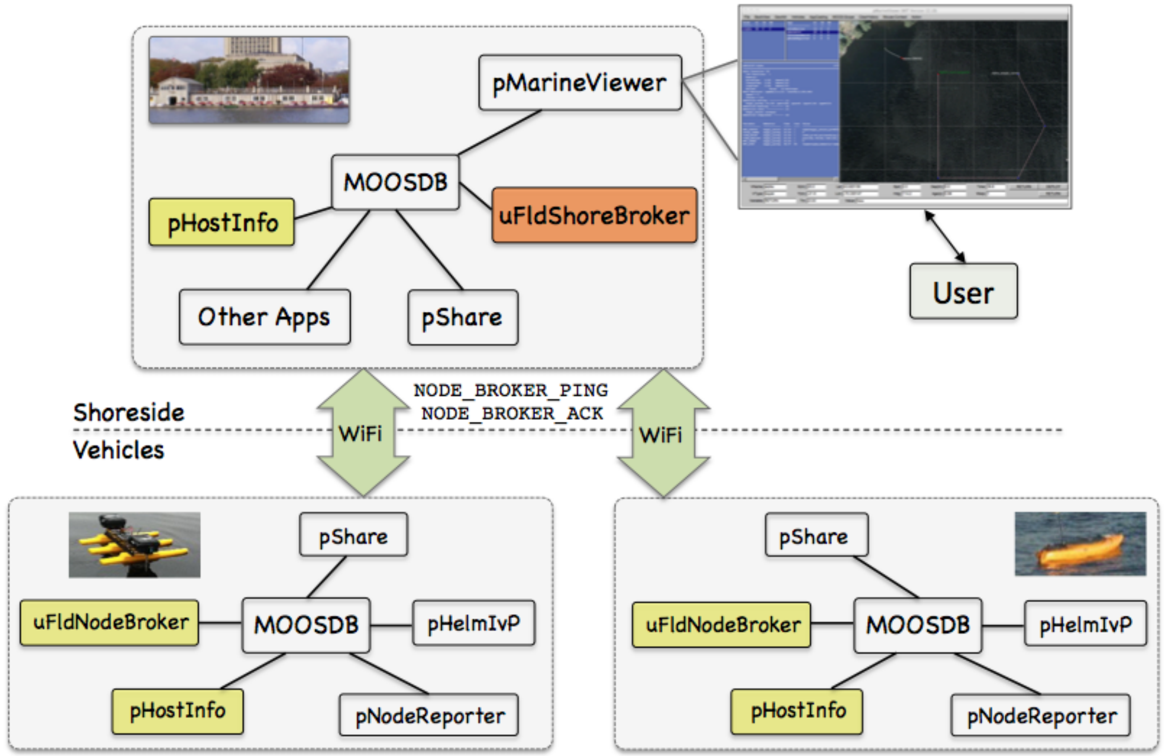
- Listening for incoming nodes
- Notifying the nodes of the shoreside IP address and pShare input route,
- Auto-configuring its own local pShare outgoing route

The uField Toolbox

We want this to be as automatic as possible.

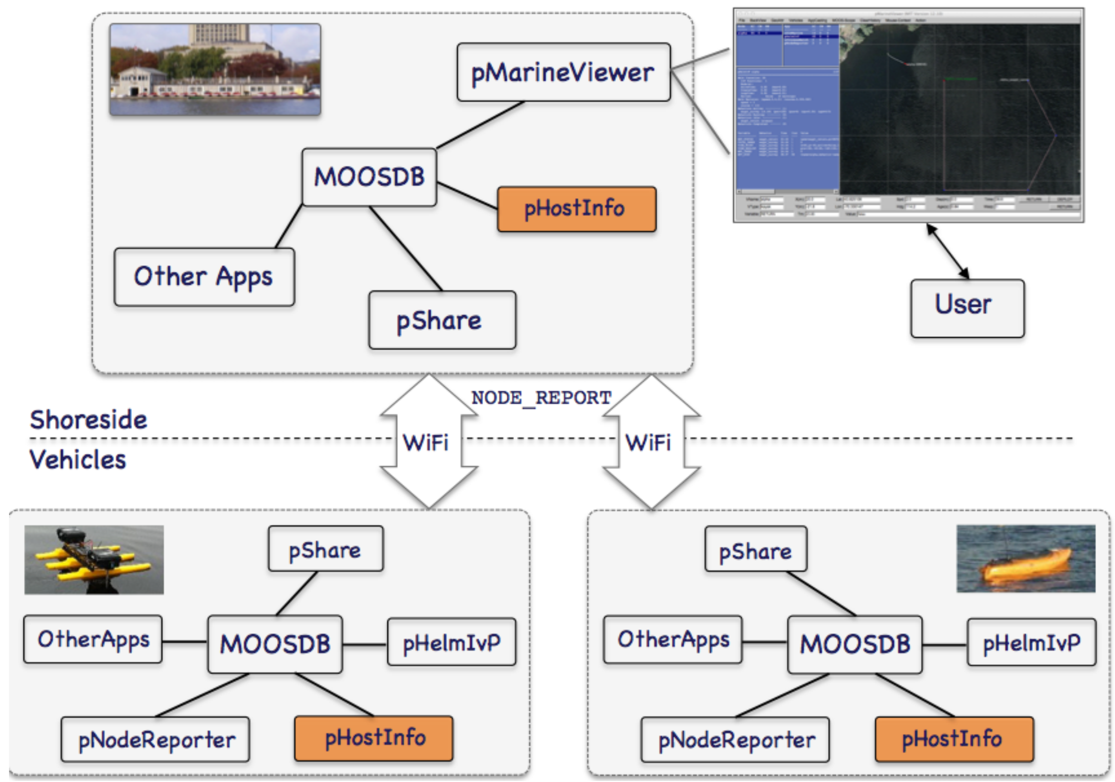
pHostInfo

A MOOS app for automatically determining the local machines IP address, and publishing it to the MOOSDB



The pHostInfo Utility

Purpose: Determine the IP address of the machine.
Publish the result in `PHI_HOST_IP`



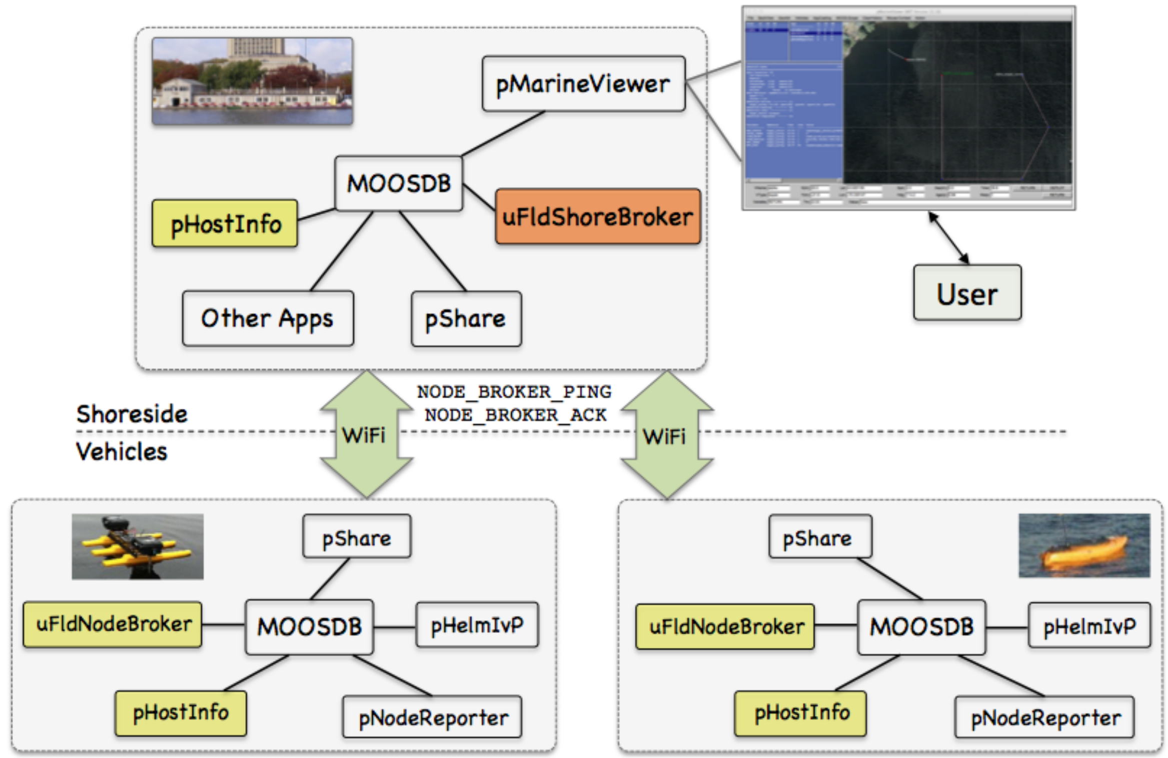

```
        PHI_HOST_IP = 118.10.24.23
    PHI_HOST_IP_ALL = 118.10.24.23,169.224.126.40
PHI_HOST_IP_VERBOSE = OSX_ETHERNET2=118.10.24.23,OSX_AIRPOT=169.224.126.40
```

The uField Toolbox

uFieldNodeBroker

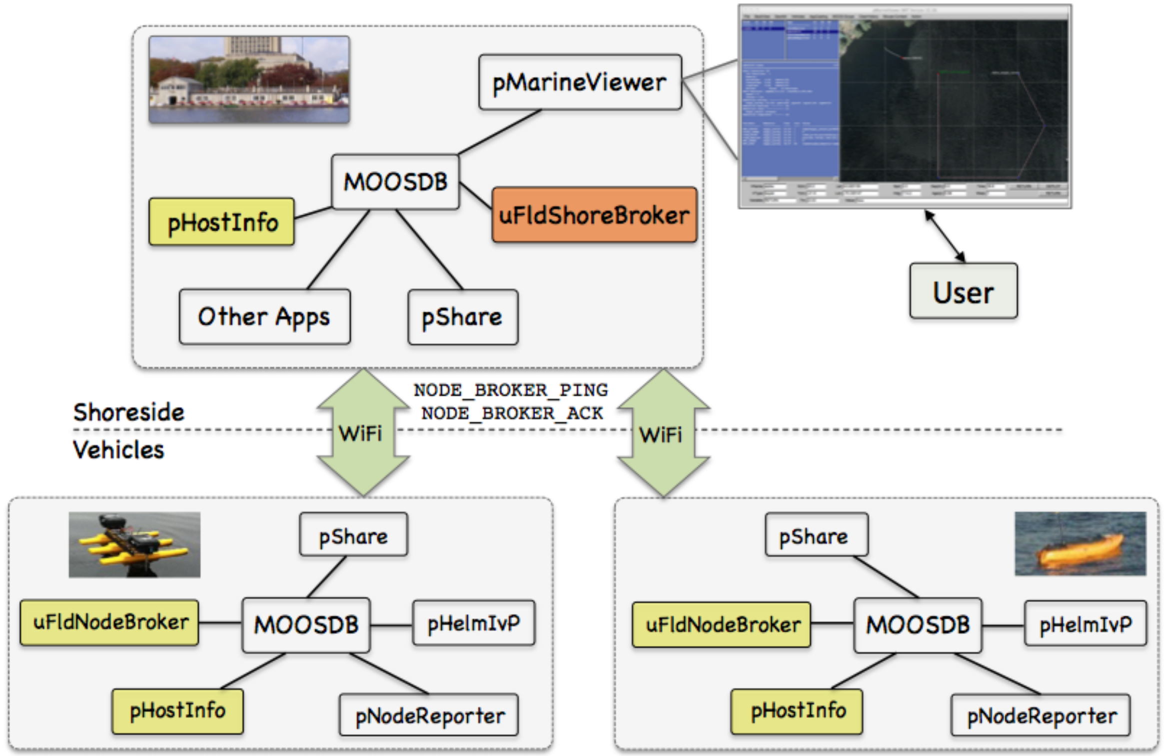
uFldNodeBroker

A MOOS app for
- finding a shore side,
- determining it's IP address and pShare input route,
- Auto-configuring its own local pShare outgoing route



The uField Toolbox

uFieldNodeBroker



uFldNodeBroker

A MOOS app for
- finding a shore side,
- determining it's IP address and pShare input route,
- Auto-configuring its own local pShare outgoing route

- Gets local host IP information from pHostInfo.
- Pings a candidate shoreside community with information about itself

```
NODE_BROKER_PING = community=henry,hostip=192.168.1.1,port_db=9000,
 pshare_iroutes=192.168.1.1:9200,timewarp=8
```

- Receives reply from shoreside with information about the shoreside community.

```
NODE_BROKER_ACK = community=shoreside,hostip=192.168.1.199,port_db=9000,
     pshare_iroutes=192.168.1.199:9300,timewarp=8,status=ok
```

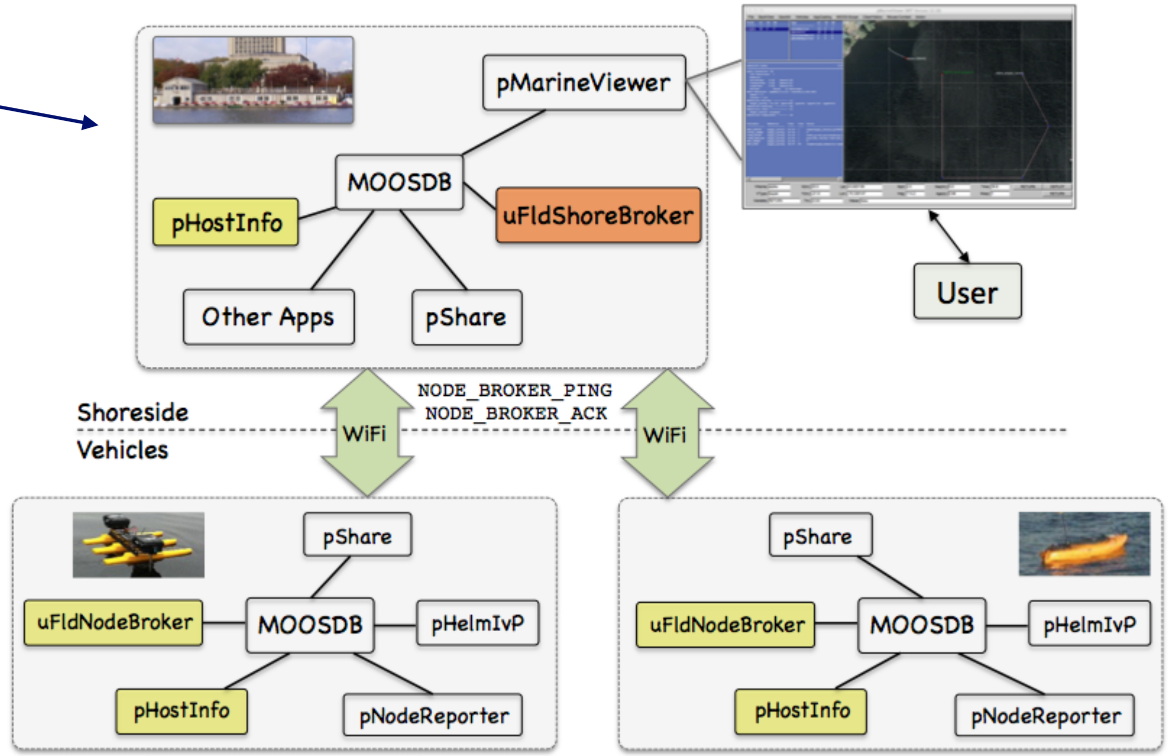
- Augments the local pShare configuration

```
PSHARE_CMD = src_name=NODE_REPORT_LOCAL, dest_name=NODE_REPORT, route=192.68.1.199:9300
```

The uField Toolbox

uFldShoreBroker

Runs in the shoreside community



uFldShoreBroker

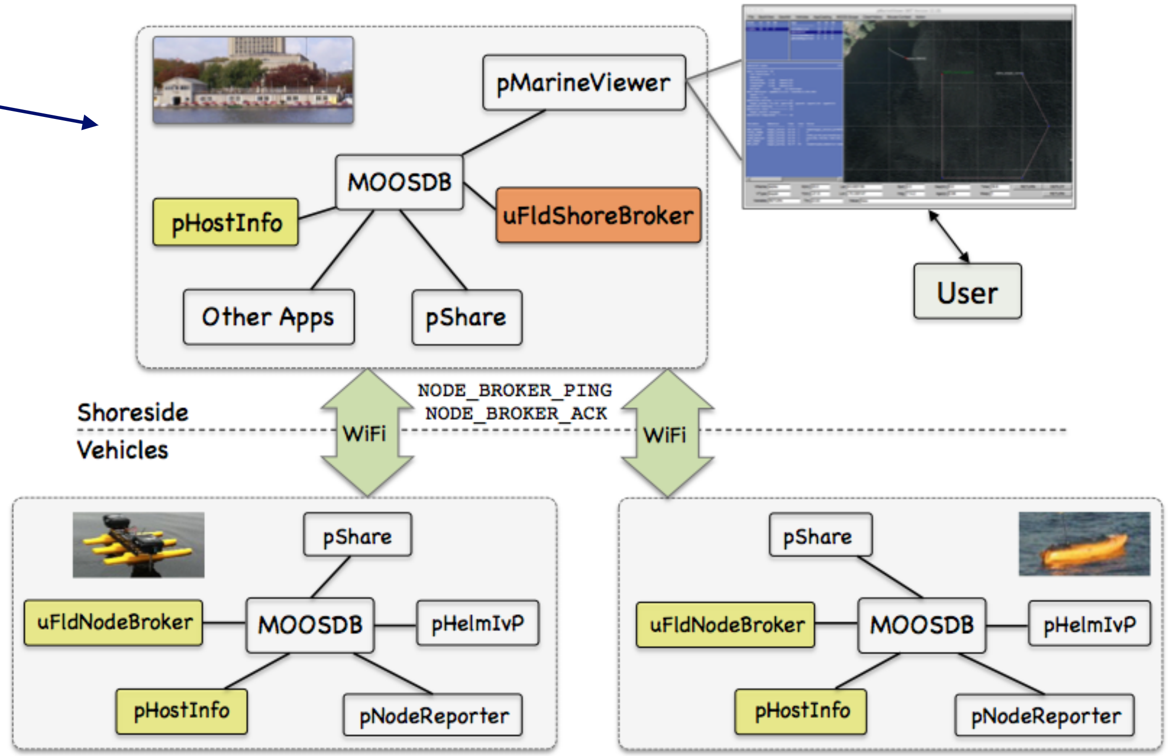
A MOOS app for

- Listening for incoming nodes
- Notifying the nodes of the shoreside IP address and pShare input route,
- Auto-configuring its own local pShare outgoing route

The uField Toolbox

uFldShoreBroker

Runs in the shoreside community



uFldShoreBroker

A MOOS app for

- Listening for incoming nodes
- Notifying the nodes of the shoreside IP address and pShare input route,
- Auto-configuring its own local pShare outgoing route

- Gets local host IP information from pHostInfo.
- Receives a ping from a candidate shoreside community with information about a vehicle.

```
NODE_BROKER_PING = community=henry,hostip=192.168.1.1,port_db=9000,
pshare_iroutes=192.168.1.1:9200,timewarp=8
```

- Sends reply from shoreside to vehicle with information about the shoreside community.

```
NODE_BROKER_ACK = community=shoreside,hostip=192.168.1.199,port_db=9000,
    pshare_iroutes=192.168.1.199:9300,timewarp=8,status=ok
```

- Augments the local pShare configuration

```
PSHARE_CMD = src_name=NODE_REPORT_LOCAL, dest_name=NODE_REPORT, route=192.68.1.199:9300
```

The uField Toolbox

(Overview)

The uField Toolbox is comprised of four general capabilities:

1. Facilitation of Inter MOOSDB Share configuration
   - pHostInfo
   - uFldNodeBroker
   - uFldShoreBroker
2. Simulation of Inter-Vehicle Messaging
   - uFldNodeComms
   - uFldMessageHandler
3. Sensor Simulation
   - uFldHazardSensor
   - uFldHazardMgr
   - uFldHazardMetric
   - uFldContactRangeSensor
   - uFldBeaconRangeSensor

The uFldMessageHander App

Inter-vehicle messaging

Vehicle alpha (source vehicle)

(Some MOOS App)
Publishes:

```
NODE_MESSAGE_LOCAL =
   "src_node=alpha,dest_node=bravo,
   var_name=STATUS,string_var=searching"
```

Vehicle bravo (dest vehicle)

uFldMessageHandler
Subscribes/Handles:
Publishes:

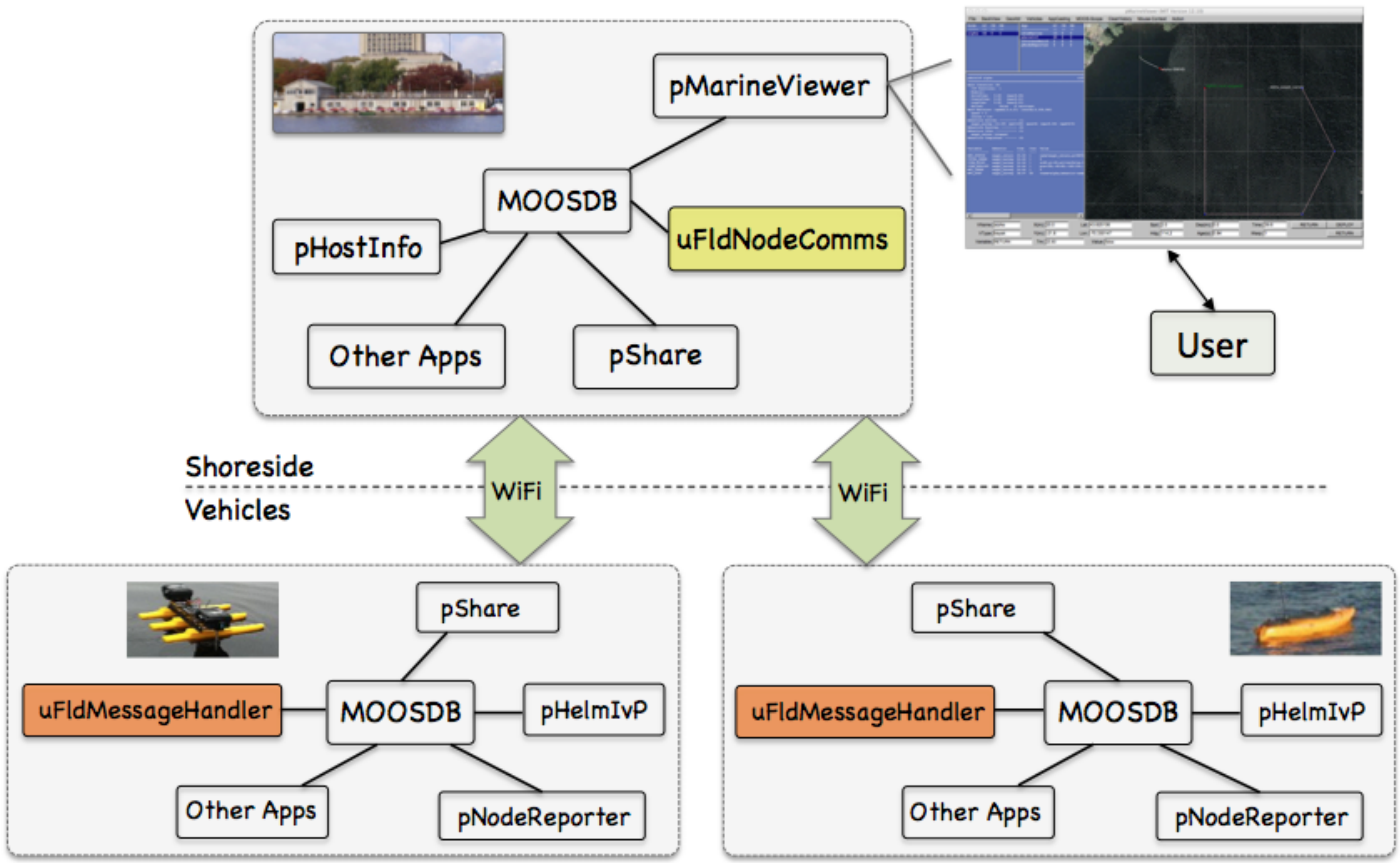
```
NODE_MESSAGE =
   "src_node=alpha,dest_node=bravo,
   var_name=STATUS,string_var=searching"
```

```
STATUS = "searching"
```

The uFldMessageHander App

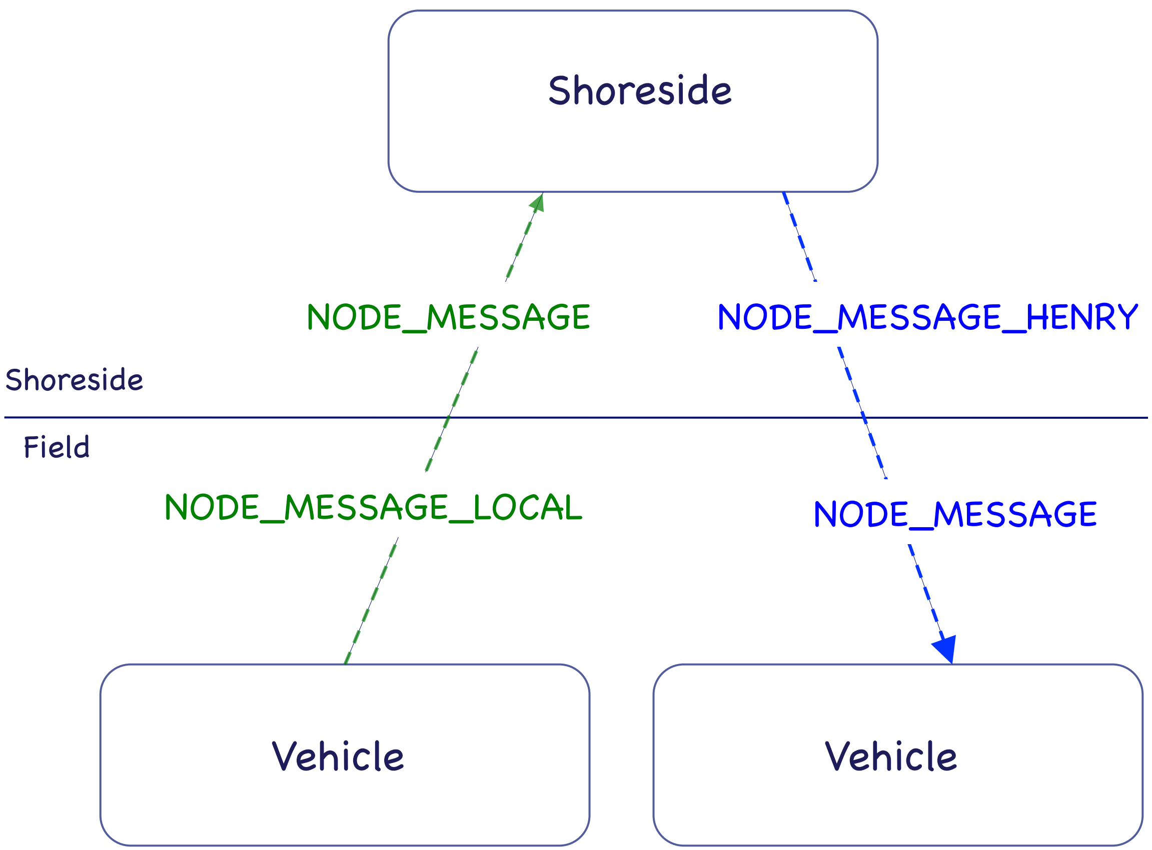
Typical Topology

The uFldMessageHandler app is running on all vehicles wishing to receive messages.



Message Routing

Message routing is handled on the shoreside

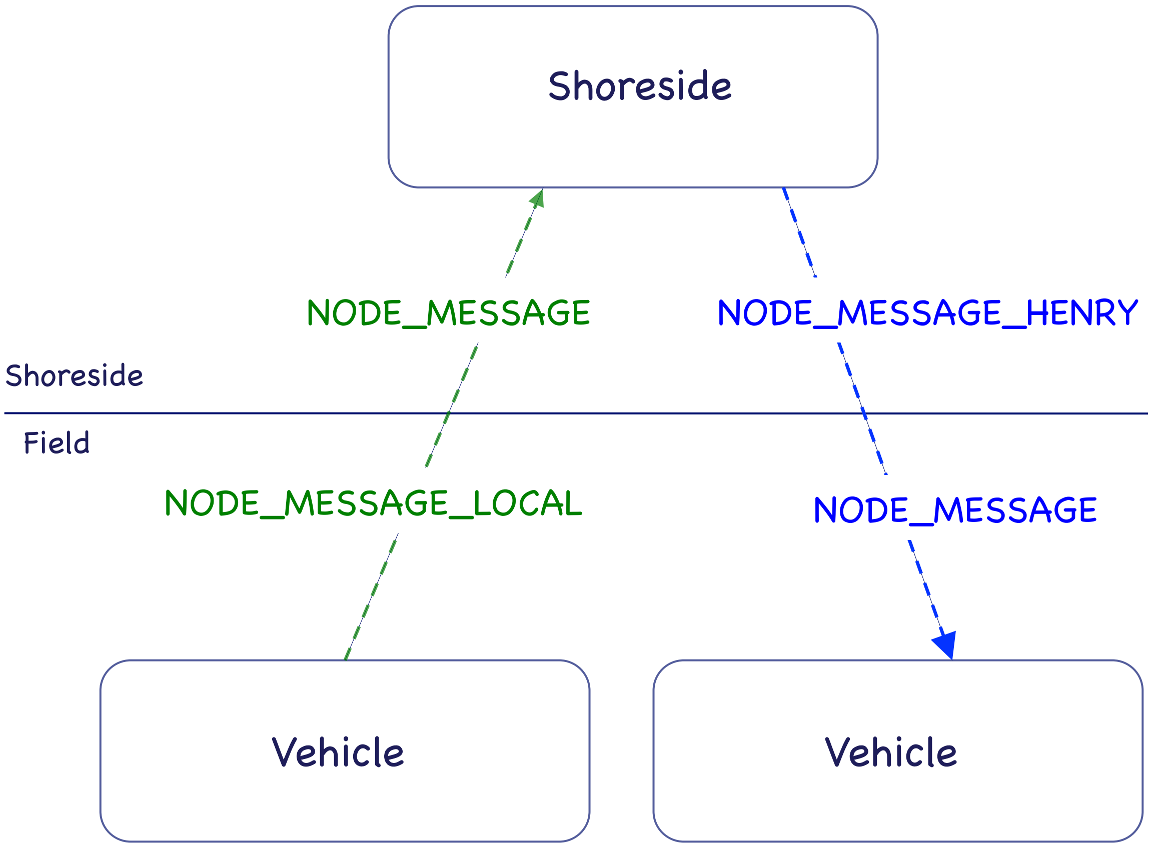



```
ProcessConfig = uFieldShoreBroker
{
   QBRIDGE = NODE_MESSAGE
}
```

```
ProcessConfig = uFieldNodeBroker
{
   BRIDGE = src=NODE_MESSAGE_LOCAL
            alias=NODE_MESSAGE
}
```

uField Message Routing

Message routing is handled on the shoreside
But it's not the case that all messages make it through
They are handled by uFldNodeComms.



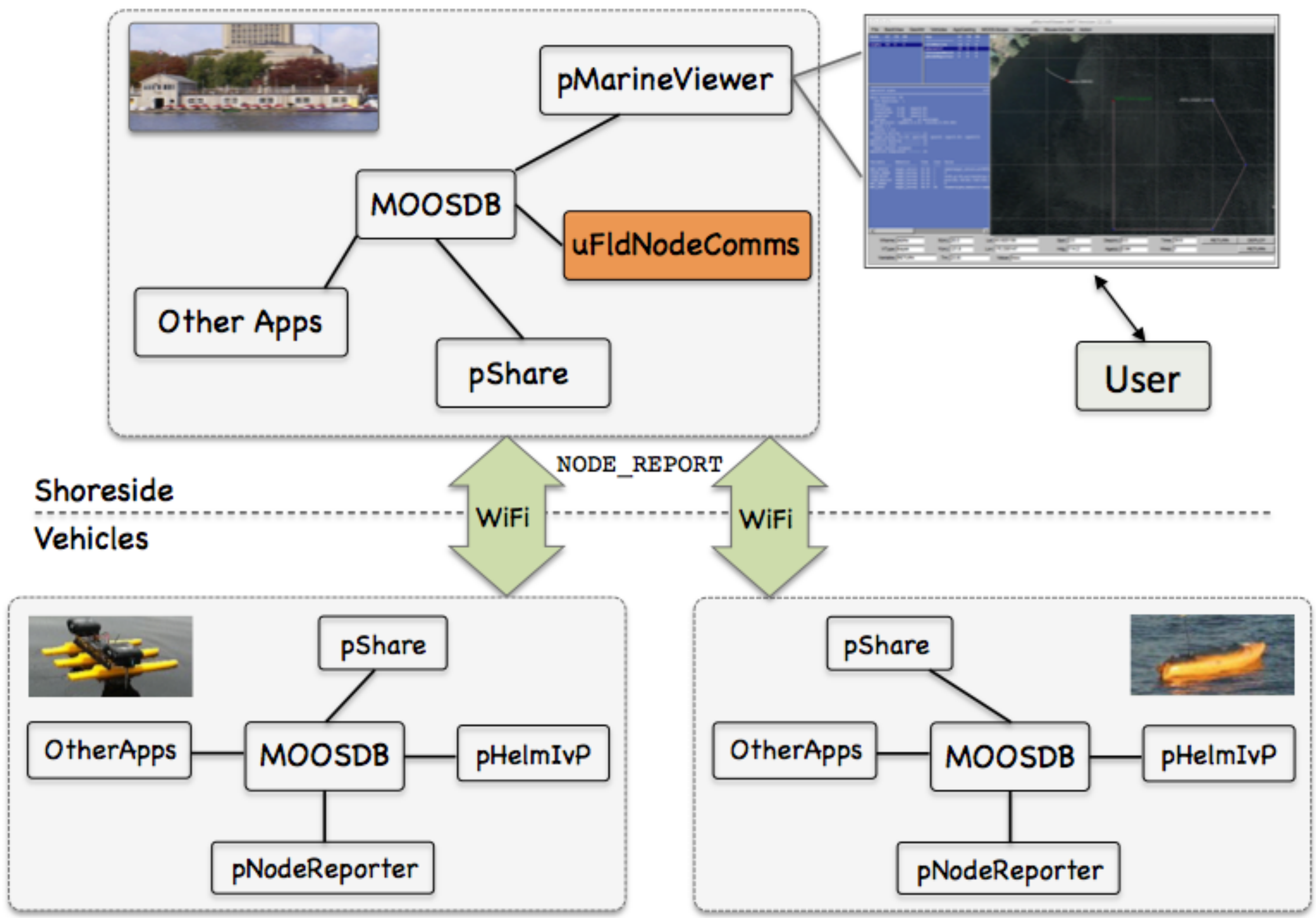

```
ProcessConfig = uFieldShoreBroker
{
   QBRIDGE = NODE_MESSAGE
}
```

```
ProcessConfig = uFieldNodeBroker
{
   BRIDGE = src=NODE_MESSAGE_LOCAL
            alias=NODE_MESSAGE
}
```

The uFldNodeComms App

Typical Topology

The uFldNodeComms app runs on the shoreside, limits intervehicle messaging.



The uFldNodeComms App

Typical Application Topology

The uFldNodeComms configuration parameters:

```
ProcessConfig = uFieldNodeComms
{
   COMMS_RANGE       = 200
   MIN_MSG_INTERVAL = 60
   MAX_MSG_LENGTH    = 100
}
```

Distance in meters between vehicles

Min time between messages from a vehicle

Max chars in a string message

The uField Toolbox

(Sensor Simulation)

The uField Toolbox is comprised of four general capabilities:

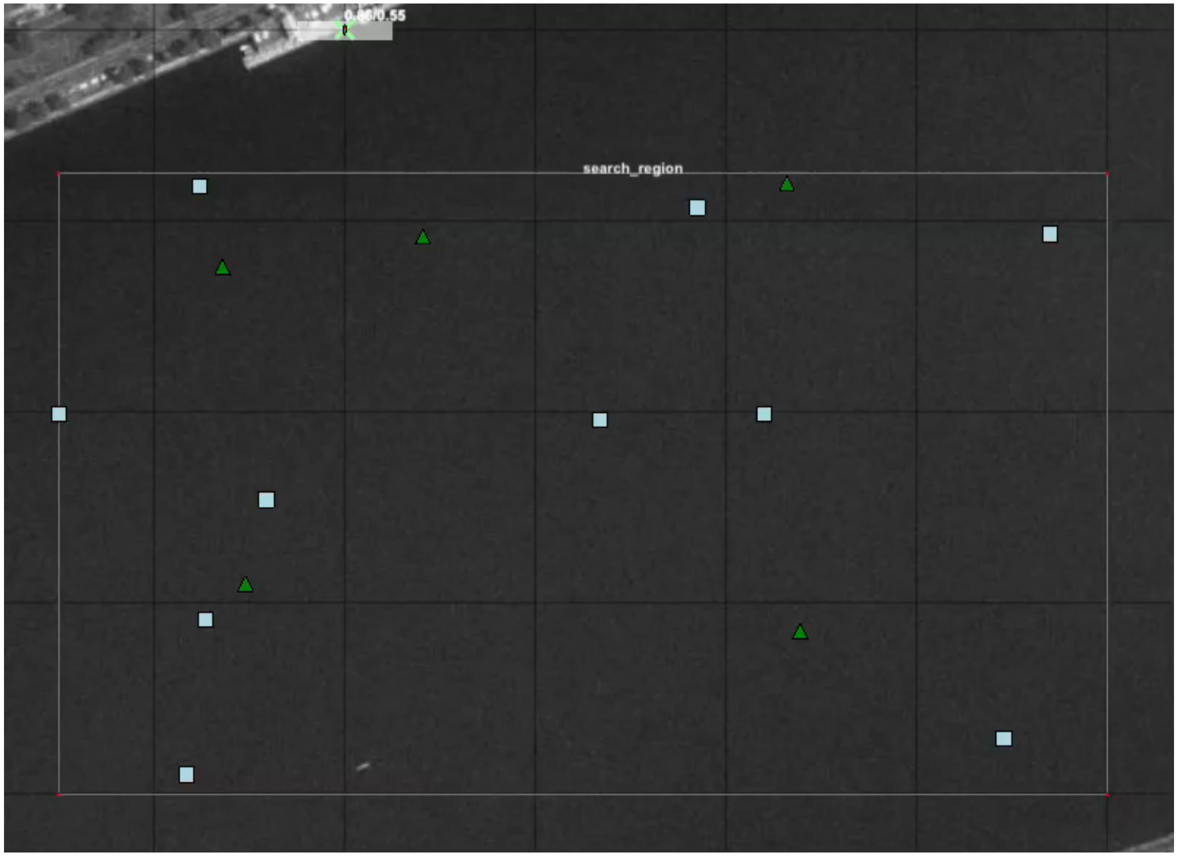
1. Facilitation of Inter MOOSDB Share configuration
   - pHostInfo
   - uFldNodeBroker
   - uFldShoreBroker
2. Simulation of Inter-Vehicle Messaging
   - uFldNodeComms
   - uFldMessageHandler
3. Sensor Simulation
   - uFldHazardSensor
   - uFldHazardMgr
   - uFldHazardMetric
   - uFldContactRangeSensor
   - uFldBeaconRangeSensor

The uField Toolbox

(Sensor Simulation)

The uField Toolbox is comprised of four general capabilities:

1. Facilitation of Inter MOOSDB Share configuration
   - pHostInfo
   - uFldNodeBroker
   - uFldShoreBroker
2. Simulation of Inter-Vehicle Messaging
   - uFldNodeComms
   - uFldMessageHandler
3. Sensor Simulation
   - uFldHazardSensor
   - uFldHazardMgr
   - uFldHazardMetric
   - uFldContactRangeSensor
   - uFldBeaconRangeSensor

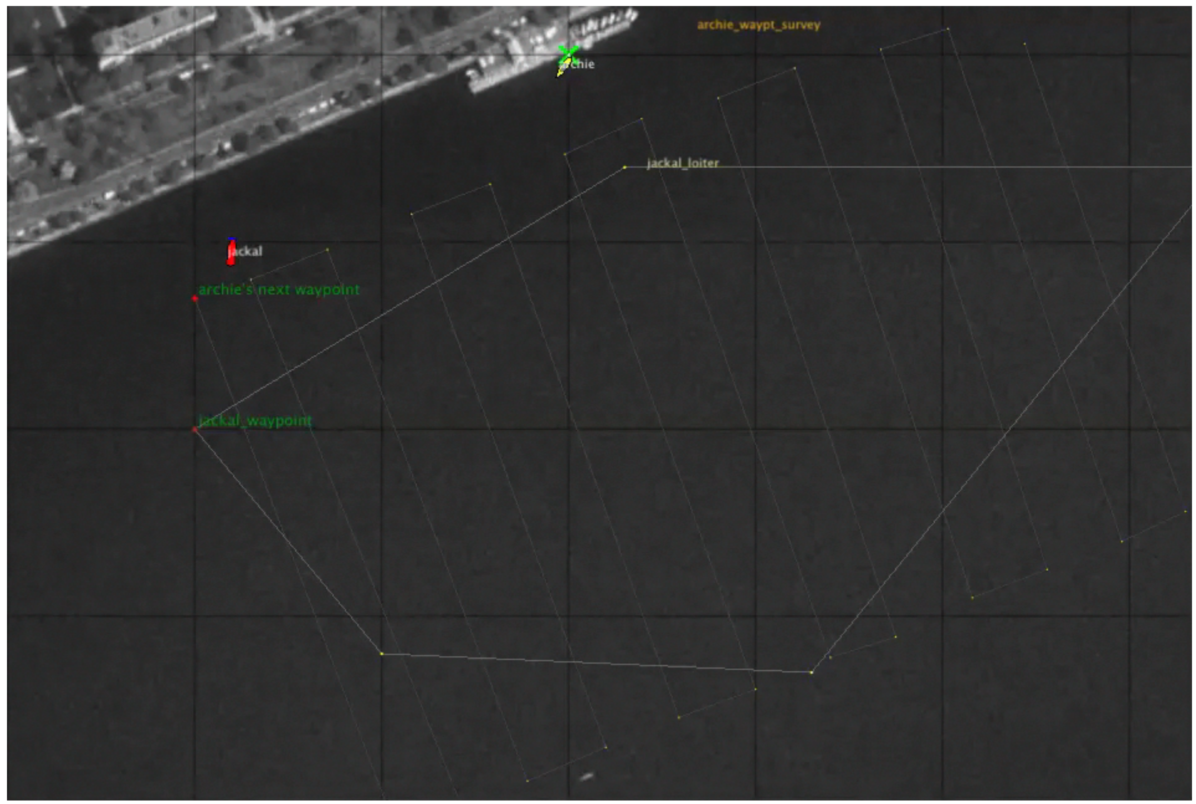


The uField Toolbox

(Sensor Simulation)

The uField Toolbox is comprised of four general capabilities:

1. Facilitation of Inter MOOSDB Share configuration
   - pHostInfo
   - uFldNodeBroker
   - uFldShoreBroker
2. Simulation of Inter-Vehicle Messaging
   - uFldNodeComms
   - uFldMessageHandler
3. Sensor Simulation
   - uFldHazardSensor
   - uFldHazardMgr
   - uFldHazardMetric
   - uFldContactRangeSensor
   - uFldBeaconRangeSensor

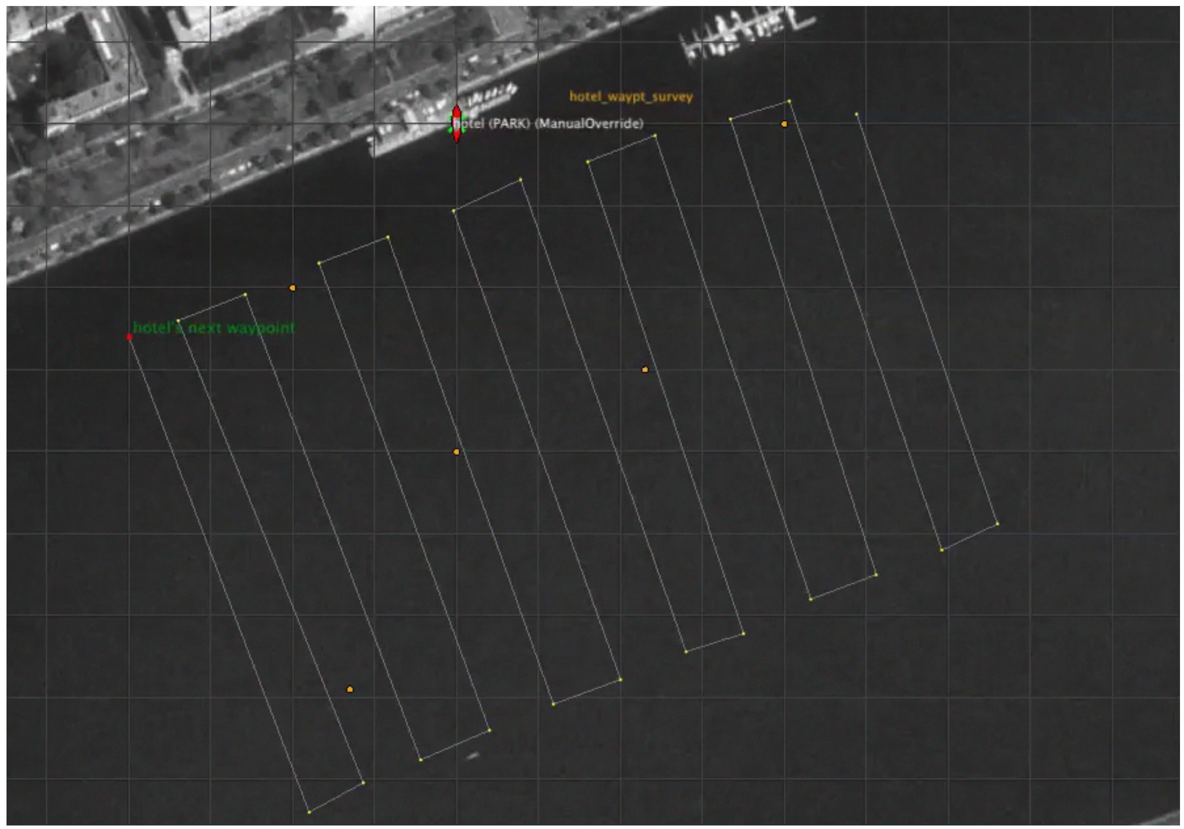


The uField Toolbox

(Sensor Simulation)

The uField Toolbox is comprised of four general capabilities:

1. Facilitation of Inter MOOSDB Share configuration
   - pHostInfo
   - uFldNodeBroker
   - uFldShoreBroker
2. Simulation of Inter-Vehicle Messaging
   - uFldNodeComms
   - uFldMessageHandler
3. Sensor Simulation
   - uFldHazardSensor
   - uFldHazardMgr
   - uFldHazardMetric
   - uFldContactRangeSensor
   - uFldBeaconRangeSensor



Outline

- Objectives and Motivations
- The Marine Autonomy Courseware
- The uField Toolbox
- AppCasting
- Changes / Additions to the Helm
- Ongoing / Future Efforts

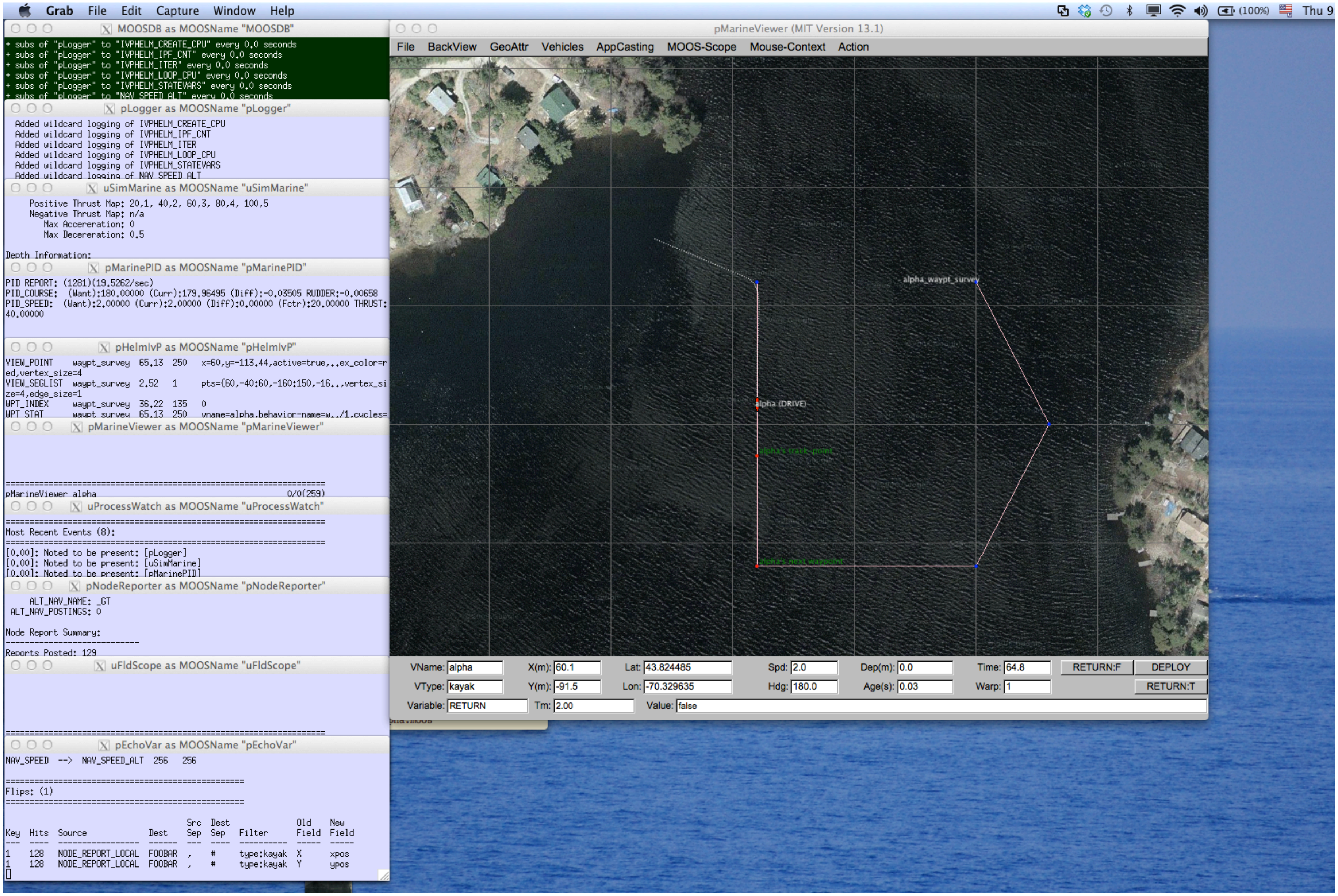
AppCasting
in MOOS

AppCasting was motivated by a few observations:

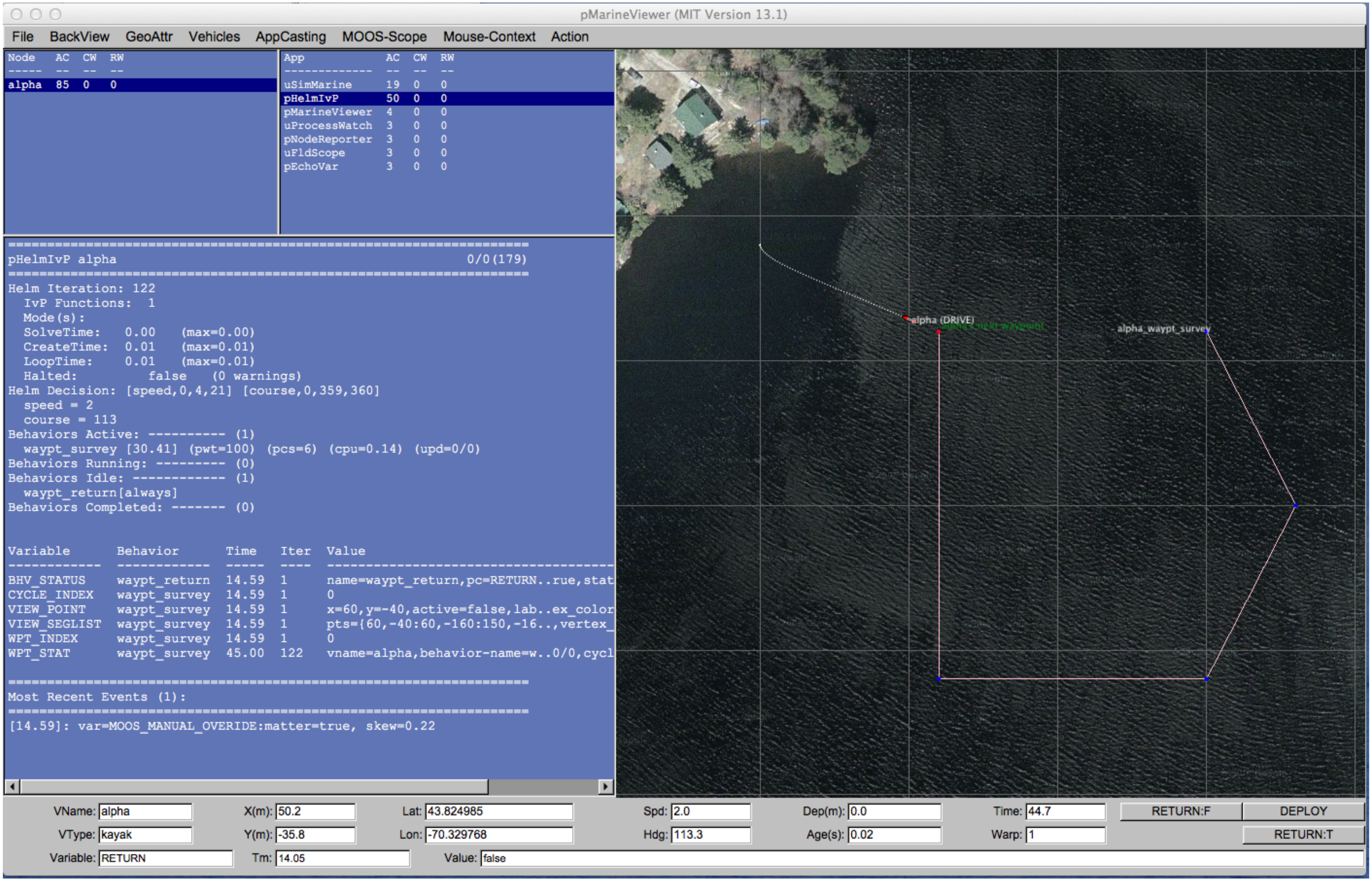
- The biggest headache of users new to MOOS (students in MIT 2.680) was the derailment of a mission due to an unnoticed configuration or runtime error.
- Debugging typically involves re-launching with app terminal windows open and analyzing expected vs. observed output.
- Deploying multiple vehicles each with multiple MOOS Apps means a lot of terminal windows are open.
- On a remotely deployed vehicle, one cannot ssh in and see any application terminal output at all!
- Since terminal output is rarely viewable for the above practical reasons, apps are rarely designed with much thought put into their terminal output.

... Introducing AppCasting in MOOS

Without AppCasting

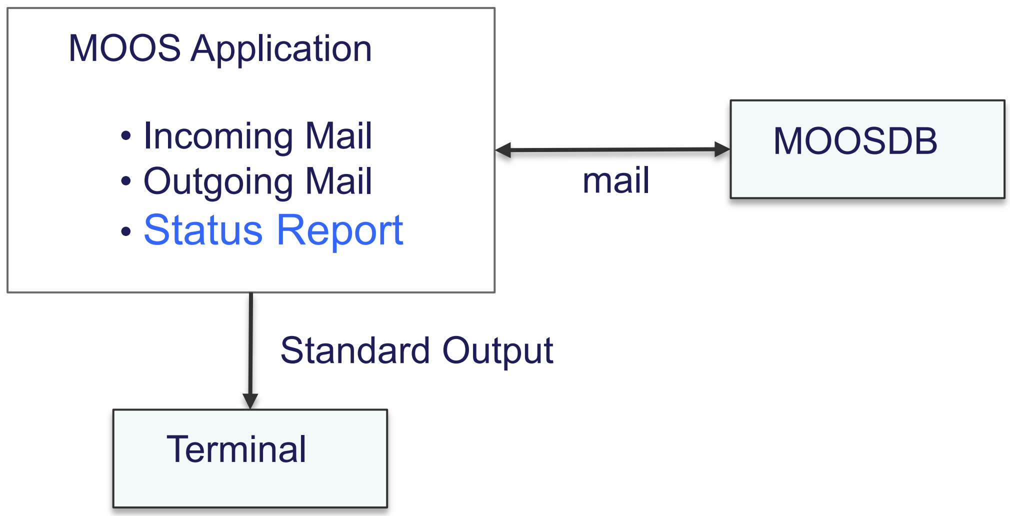


With AppCasting



MOOS Application I/O

- A typical MOOS application interacts by way of mail and the MOOSDB.



- Most applications also produce debugging/status info to the terminal.
- Often this format is an afterthought.
- Often this content is out of sight, if a terminal is not open.

Typical Terminal Output

Typical terminal output of a MOOSApp will show:

- Startup summary and health status,
- A simple heart beat character or other simple health indicator.

pLogger

Terminal — pLogger — 87×22 — ⌘4

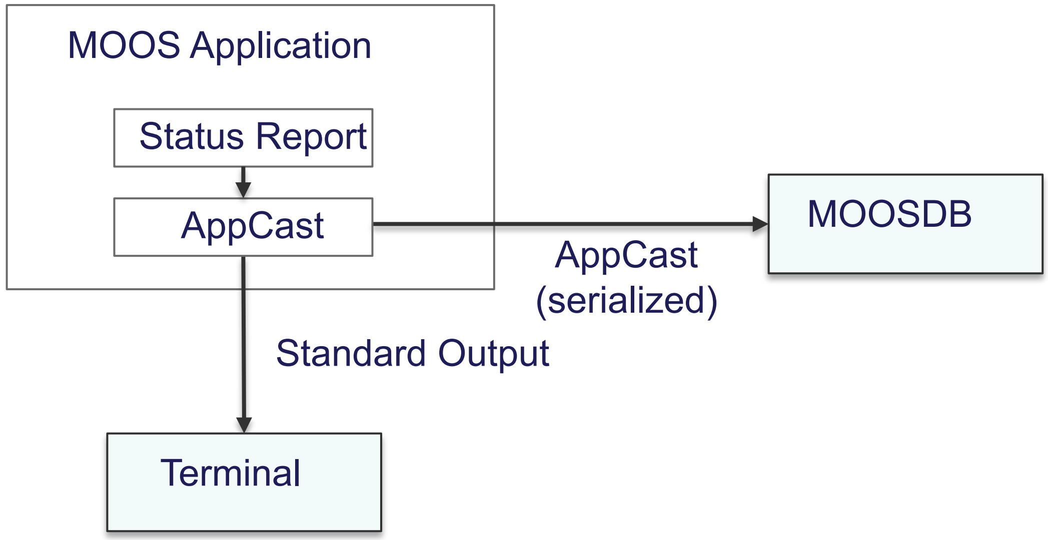
pLogger

```
****************************************************
*                                                  *
*       This is MOOS Client                        *
*       c. P Newman 2001                           *
*                                                  *
****************************************************

---------------MOOS CONNECT-----------------------
  contacting a MOOS server localhost:9000 -  try 00001
  Contact Made
  Handshaking as "pLogger"
  Handshaking Complete
  Invoking User OnConnect() callback...ok
--------------------------------------------------

Warning:
        neither "::GlobalLogPath" or "Path" are specified
        Will Log to ./
pLogger is Running:
         AppTick    @ 4.0 Hz
         CommsTick @ 4 Hz
```

Introducing AppCasting



An AppCast-Enabled MOOS App:

- Generates an AppCast representing its status report.
- The AppCast is sent to the terminal standard output.
  (From the user's perspective it looks like any other MOOS application.)
- The AppCast is also serialized and sent to the MOOSDB.

An Example AppCast

From the uProcessWatch MOOSApp

```
==================================================================
uProcessWatch henry                                          (160)
==================================================================
Summary: All Present

Antler List: pBasicContactMgr,pHelmIvP,pHostInfo,pLogger,pMarinePID
             pNodeReporter,pShare,uFldMessageHandler,uFldNodeBroker
             uSimMarine,uXMS

ProcName              Watch Reason   Status
------------------    ------------   ------
pBasicContactMgr      ANT        DB   OK
pHelmIvP              ANT WATCH DB   OK
pHostInfo             ANT        DB   OK
pLogger               ANT        DB   OK
pMarinePID            ANT WATCH DB   OK
pNodeReporter         ANT WATCH DB   OK
pShare                ANT        DB   OK
uFldMessageHandler    ANT        DB   OK
uFldNodeBroker        ANT        DB   OK
uSimMarine            ANT WATCH DB   OK

==================================================================
Most Recent Events (8):
==================================================================
[4.01]: Resurrected: [uFldMessageHandler]
[2.01]: PROC_WATCH_EVENT: Process [uFldMessageHandler] is missing.
[0.00]: Noted to be present: [pShare]
[0.00]: Noted to be present: [pLogger]
[0.00]: Noted to be present: [pBasicContactMgr]
[0.00]: Noted to be present: [pHostInfo]
[0.00]: Noted to be present: [uFldNodeBroker]
[0.00]: Noted to be present: [uFldMessageHandler]
```

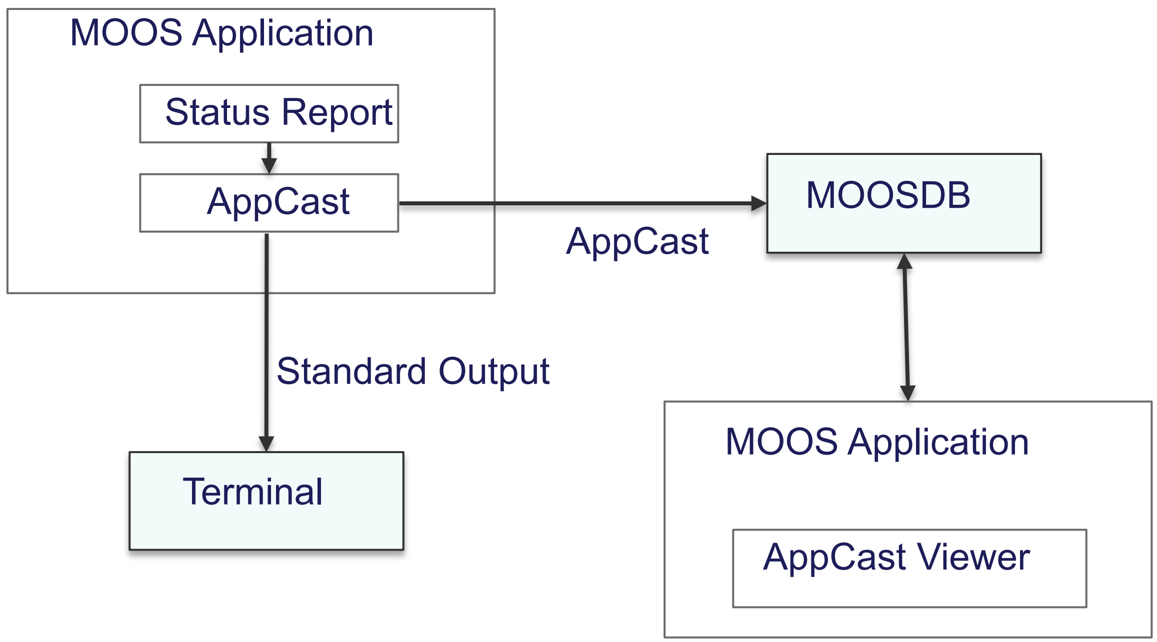
Application Iteration Counter

List of Strings

List of Events (Limited)

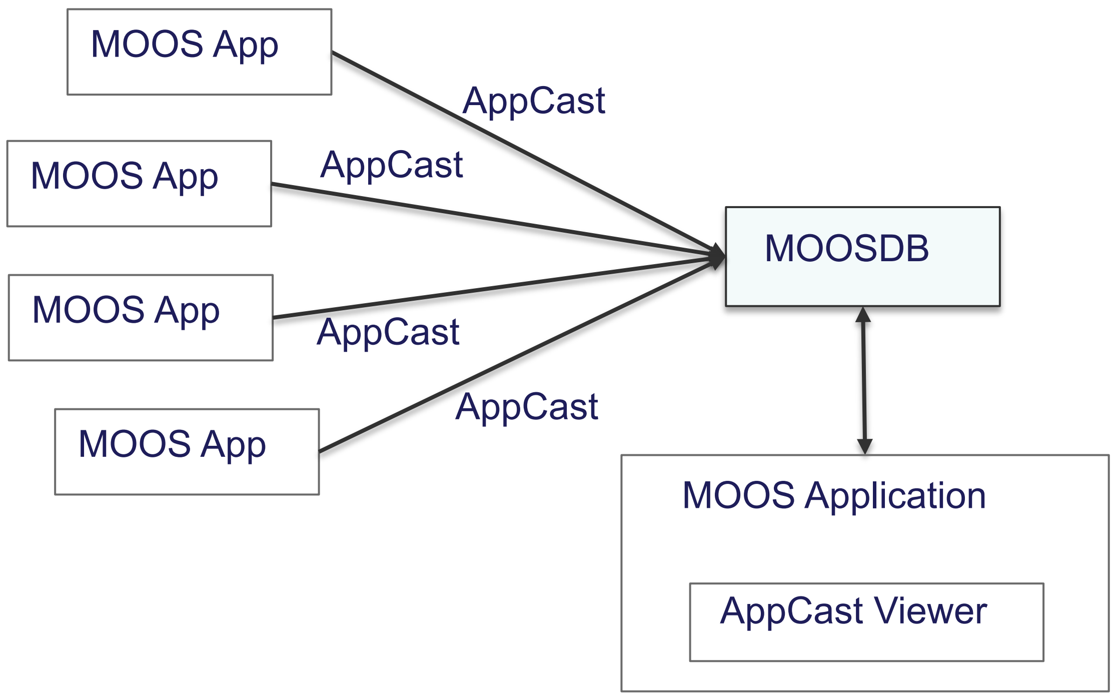
AppCast Viewing

- A separate MOOS utility application may be run to view AppCasts from any AppCast-enabled application.



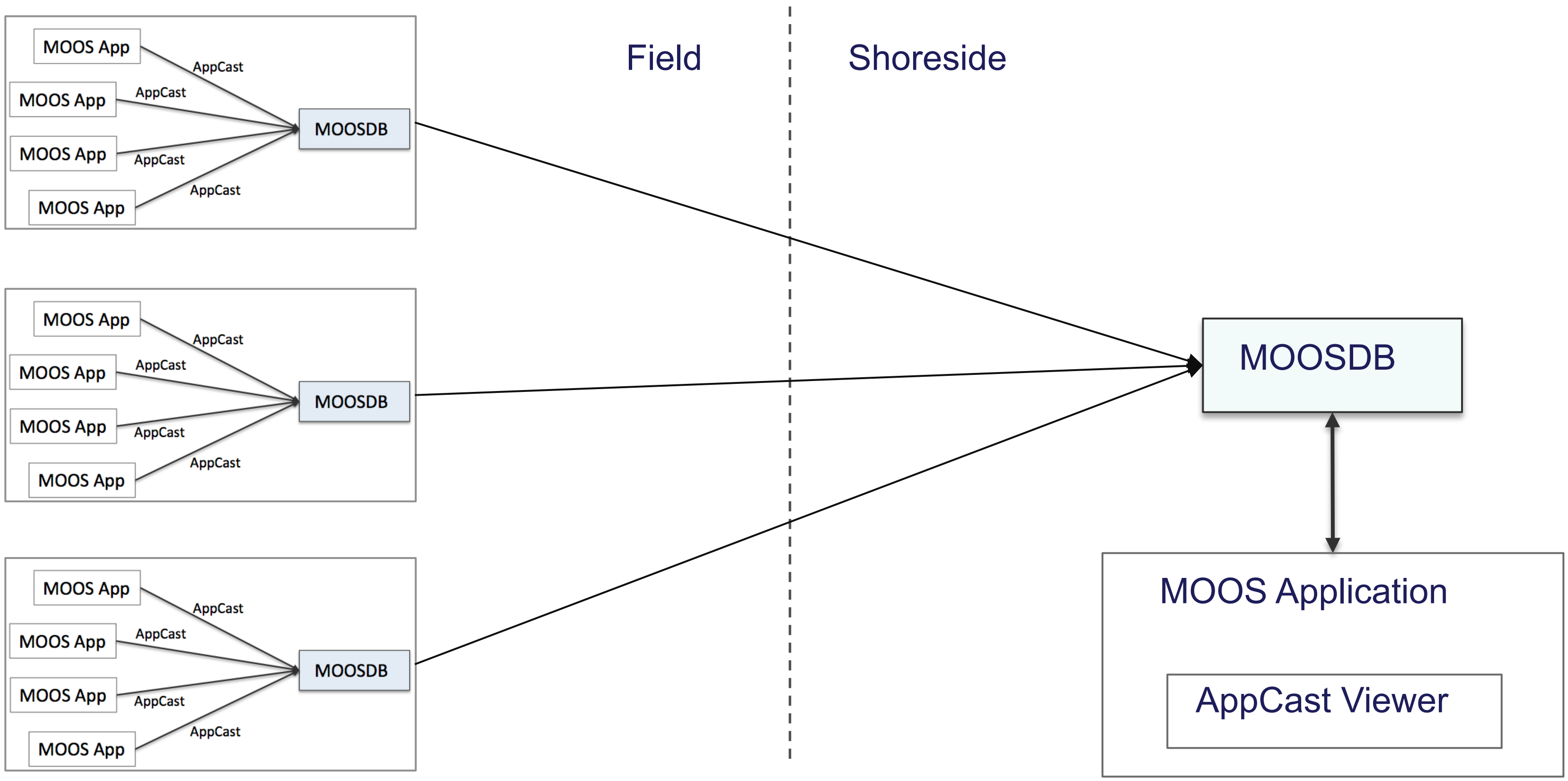
- Now a user can see application output even if an app initially was sending terminal output to `/dev/null`.

AppCast Viewing



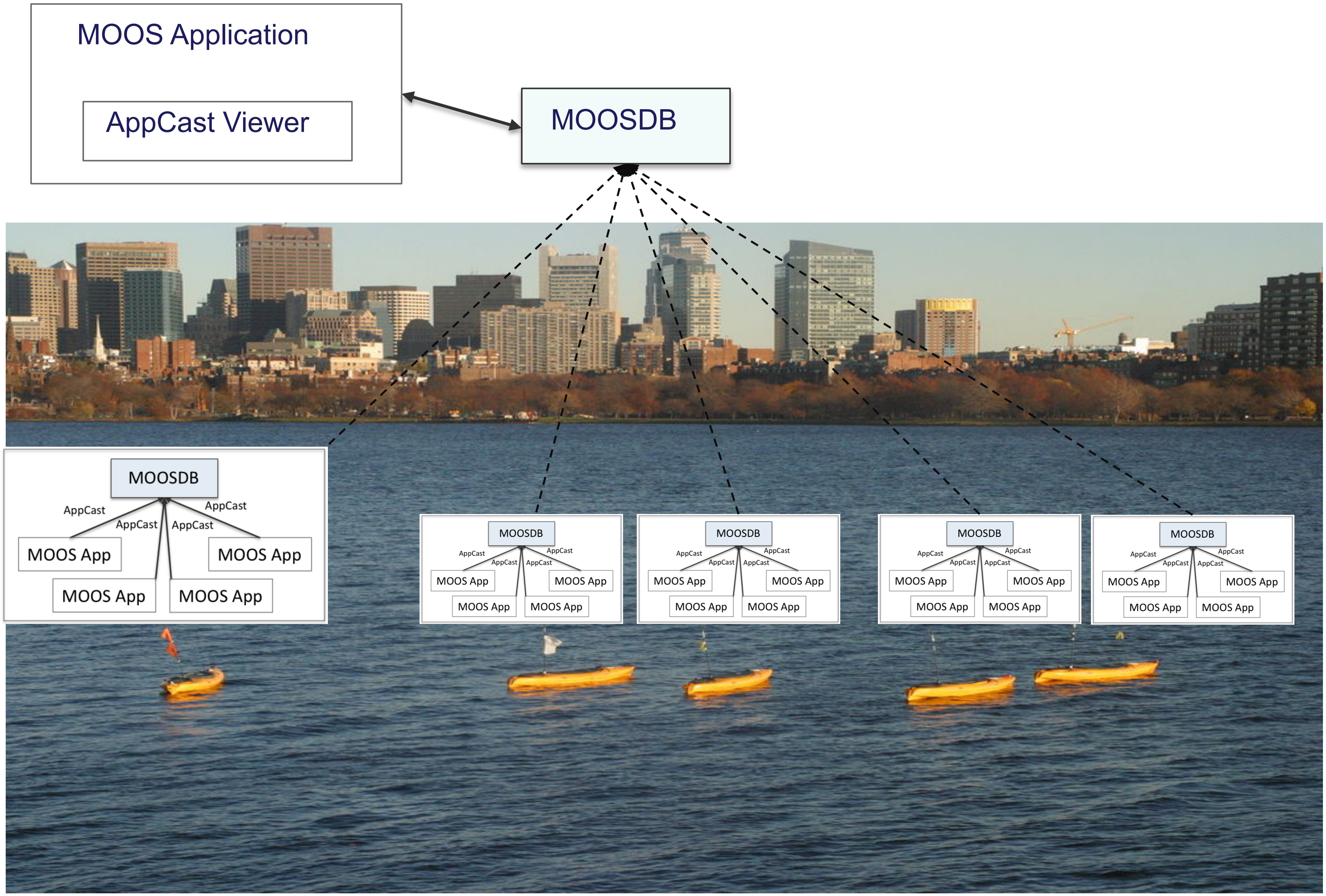
- The AppCast viewer may "connect" to multiple applications.
- The AppCast viewer can switch between "channels".
- The AppCast viewer brings Config and RunTime alerts to the user's attention even when not monitoring that channel.

AppCast Viewing



- The AppCast viewer may connect to multiple vehicles, diving down to the vehicle and application it selects.

AppCast Viewing

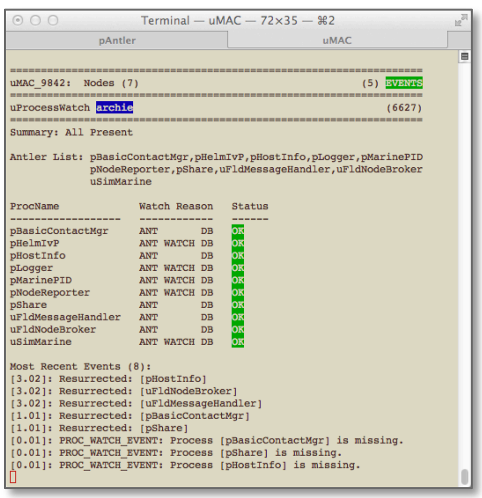


AppCast Viewers

What does an AppCast Viewer do?
- Sends AppCast requests to clients.
- Renders received AppCasts.
- Allows the user to select/switch between different MOOSApps and vehicles

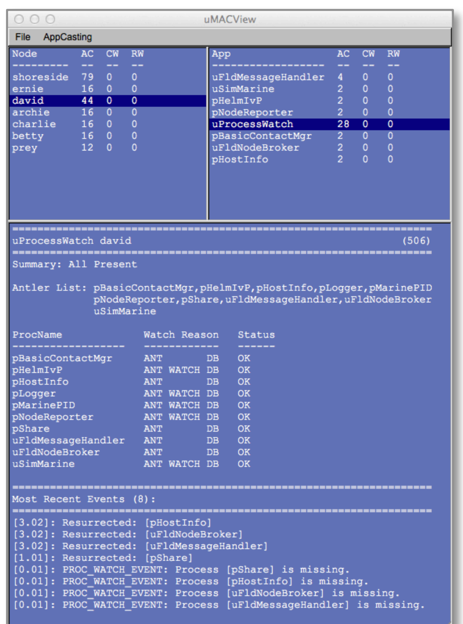
Currently there are three AppCast Viewer applications:

(1) uMAC



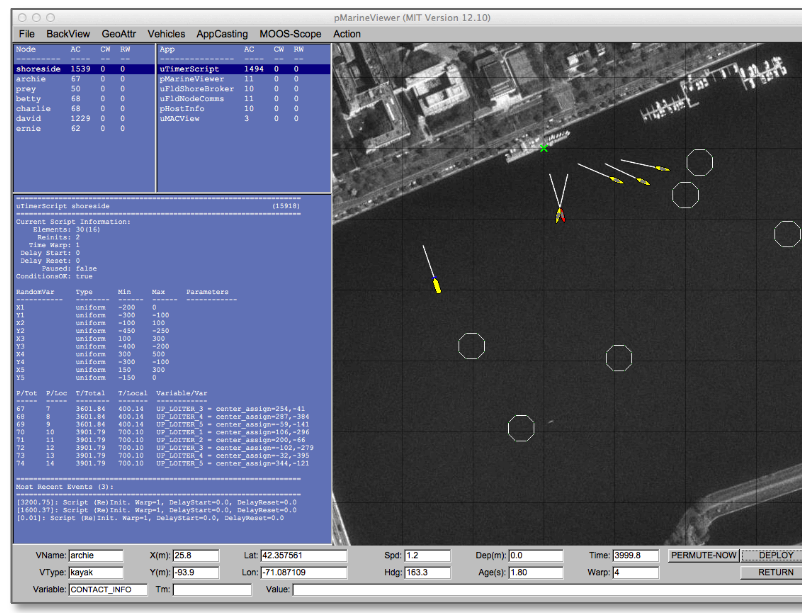
Terminal
(good for ssh'ing into a remote vehicle)

(2) uMACView



GUI (fltk)

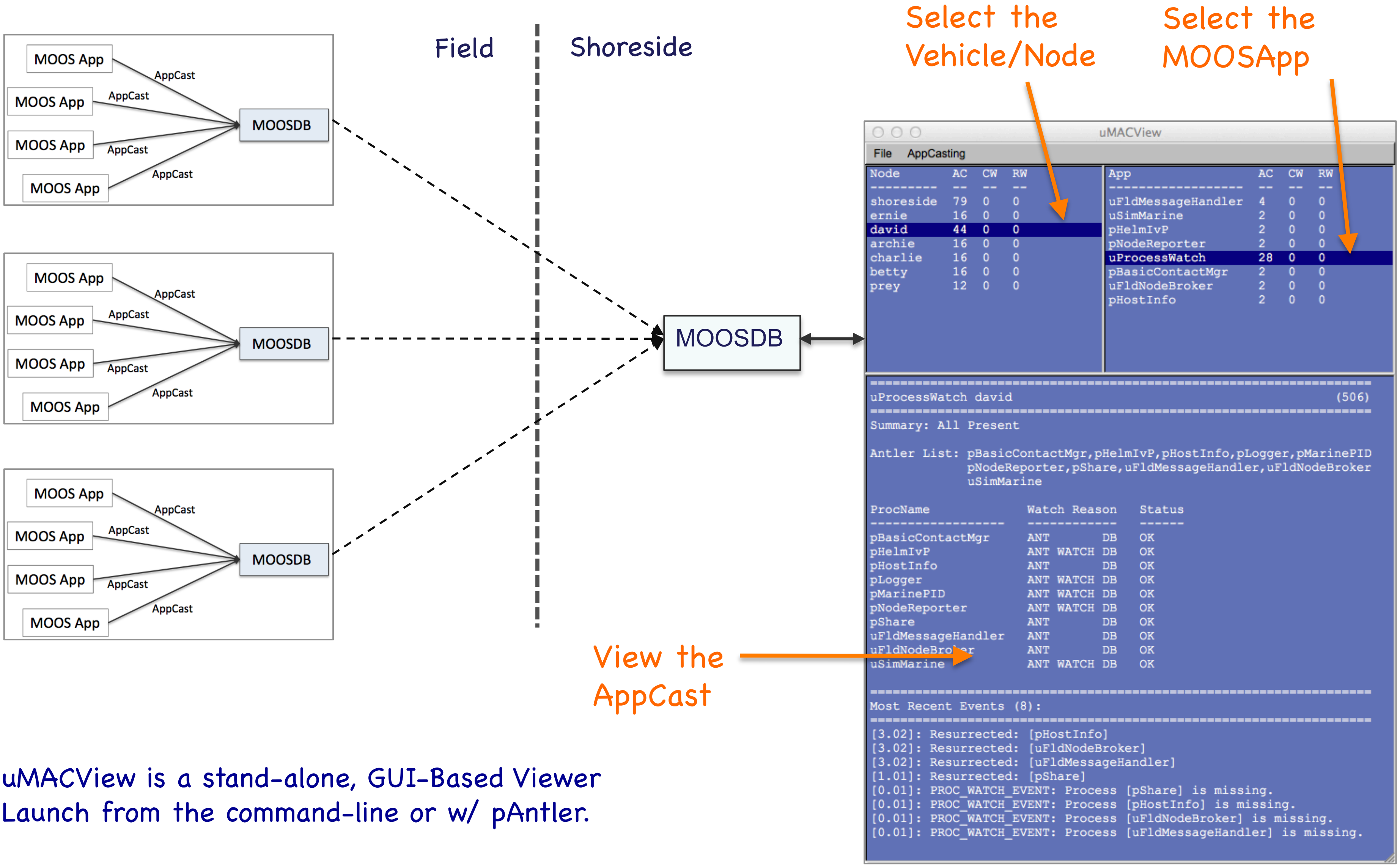
(3) pMarineViewer



GUI (fltk)

AppCast Viewing

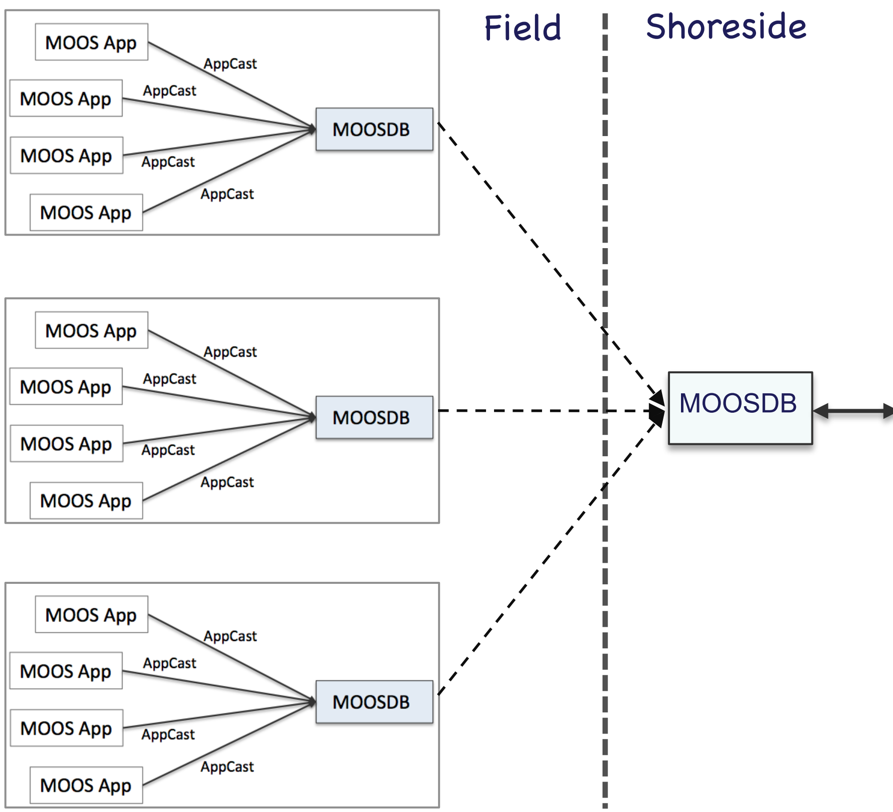
with the uMACView Tool

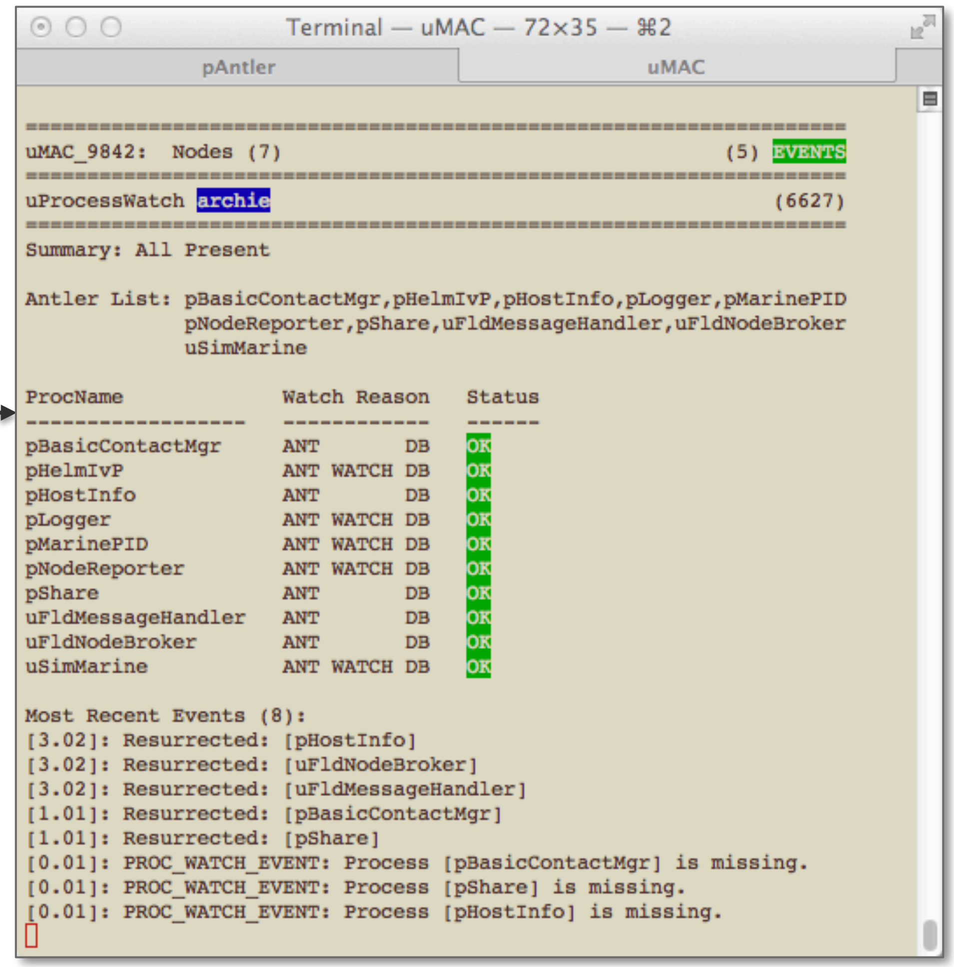


- uMACView is a stand-alone, GUI-Based Viewer
- Launch from the command-line or w/ pAntler.

AppCast Viewing

with uMAC





- Terminal interface provides most of what the GUI tools provide.
- Primary advantage: When a remote vehicle is not sending AppCasts to a shoreside, user can ssh into the vehicle and launch uMAC to debug.

The AppCast Structure

AppCast Config Warnings

- An AppCast is an instance of the AppCast C++ class.
- It contains:

Config warnings:

- Usually created at App startup time.
- Unlimited in quantity.

Config Warnings

Run Warnings

General messages

Events

```
============================================================================
uProcessWatch gilda                                            1/1 (150)
============================================================================
Configuration Warnings: 1
 [1 of 1]: Unhandled config line: foobar=abracadabra

Runtime Warnings: 1
 [1]: Process [pNodeReporter] is missing.

Summary: AWOL: pNodeReporter

Antler List: pBasicContactMgr,pHelmIvP,pHostInfo,pLogger,pMarinePID
             pNodeReporter,pShare,uFldMessageHandler,uFldNodeBroker
             uSimMarine,uXMS

ProcName            Watch Reason   Status
------------------  ------------   -------
pBasicContactMgr    ANT        DB   OK
pHelmIvP            ANT  WATCH DB   OK
pHostInfo           ANT        DB   OK
pLogger             ANT        DB   OK
pMarinePID          ANT  WATCH DB   OK
pNodeReporter       ANT  WATCH DB   MISSING
pShare              ANT        DB   OK
uFldMessageHandler  ANT        DB   OK
uFldNodeBroker      ANT        DB   OK
uSimMarine          ANT  WATCH DB   OK

============================================================================
Most Recent Events (8):
============================================================================
[120.15]: PROC_WATCH_EVENT: Process [pNodeReporter] is missing.
[0.00]: Noted to be present: [pShare]
[0.00]: Noted to be present: [pLogger]
[0.00]: Noted to be present: [pBasicContactMgr]
[0.00]: Noted to be present: [pHostInfo]
[0.00]: Noted to be present: [uFldNodeBroker]
[0.00]: Noted to be present: [uFldMessageHandler]
[0.00]: Noted to be present: [uSimMarine]
```

The AppCast Structure

AppCast Run Warnings

- An AppCast is an instance of the AppCast C++ class.
- It contains:

Run warnings:

- Created typically well after launch time when something goes wrong.
- Limited in quantity, (don't want the size of an appcast to grow unbounded)
- Provisions are made in AppCast Viewers to ensure RunWarnings come to the user's attention.

Config Warnings

Run Warnings

General messages

Events

```
==========================================================================
uProcessWatch gilda                                           1/1 (150)
==========================================================================
Configuration Warnings: 1
 [1 of 1]: Unhandled config line: foobar=abracadabra

Runtime Warnings: 1
 [1]: Process [pNodeReporter] is missing.

Summary: AWOL: pNodeReporter

Antler List: pBasicContactMgr,pHelmIvP,pHostInfo,pLogger,pMarinePID
             pNodeReporter,pShare,uFldMessageHandler,uFldNodeBroker
             uSimMarine,uXMS

ProcName             Watch Reason   Status
-----------------    ------------   -------
pBasicContactMgr     ANT        DB   OK
pHelmIvP             ANT WATCH  DB   OK
pHostInfo            ANT        DB   OK
pLogger              ANT        DB   OK
pMarinePID           ANT WATCH  DB   OK
pNodeReporter        ANT WATCH  DB   MISSING
pShare               ANT        DB   OK
uFldMessageHandler   ANT        DB   OK
uFldNodeBroker       ANT        DB   OK
uSimMarine           ANT WATCH  DB   OK

==========================================================================
Most Recent Events (8):
==========================================================================
[120.15]: PROC_WATCH_EVENT: Process [pNodeReporter] is missing.
[0.00]: Noted to be present: [pShare]
[0.00]: Noted to be present: [pLogger]
[0.00]: Noted to be present: [pBasicContactMgr]
[0.00]: Noted to be present: [pHostInfo]
[0.00]: Noted to be present: [uFldNodeBroker]
[0.00]: Noted to be present: [uFldMessageHandler]
[0.00]: Noted to be present: [uSimMarine]
```

The AppCast Structure

AppCast Messages

- An AppCast is an instance of the AppCast C++ class.
- It contains:

Messages:

- Free in format. Up to the user to pick the information and layout.
- Typically "cleared" on each generation of an appcast.
- Just a list of strings.
- Tables, columns etc. done by the user.

Config Warnings

Run Warnings

General messages

Events

```
=====================================================================
uProcessWatch gilda                                        1/1 (150)
=====================================================================
Configuration Warnings: 1
 [1 of 1]: Unhandled config line: foobar=abracadabra

Runtime Warnings: 1
 [1]: Process [pNodeReporter] is missing.

Summary: AWOL: pNodeReporter

Antler List: pBasicContactMgr,pHelmIvP,pHostInfo,pLogger,pMarinePID
             pNodeReporter,pShare,uFldMessageHandler,uFldNodeBroker
             uSimMarine,uXMS

ProcName              Watch Reason   Status
------------------    ------------   -------
pBasicContactMgr      ANT       DB   OK
pHelmIvP              ANT WATCH DB   OK
pHostInfo             ANT       DB   OK
pLogger               ANT       DB   OK
pMarinePID            ANT WATCH DB   OK
pNodeReporter         ANT WATCH DB   MISSING
pShare                ANT       DB   OK
uFldMessageHandler    ANT       DB   OK
uFldNodeBroker        ANT       DB   OK
uSimMarine            ANT WATCH DB   OK

=====================================================================
Most Recent Events (8):
=====================================================================
[120.15]: PROC_WATCH_EVENT: Process [pNodeReporter] is missing.
[0.00]: Noted to be present: [pShare]
[0.00]: Noted to be present: [pLogger]
[0.00]: Noted to be present: [pBasicContactMgr]
[0.00]: Noted to be present: [pHostInfo]
[0.00]: Noted to be present: [uFldNodeBroker]
[0.00]: Noted to be present: [uFldMessageHandler]
[0.00]: Noted to be present: [uSimMarine]
```

The AppCast Structure

AppCast Events

- An AppCast is an instance of the AppCast C++ class.
- It contains:

Config Warnings

Run Warnings

General messages

Events

```
==========================================================================
uProcessWatch gilda                                            1/1  (150)
==========================================================================
Configuration Warnings: 1
 [1 of 1]: Unhandled config line: foobar=abracadabra

Runtime Warnings: 1
 [1]: Process [pNodeReporter] is missing.

Summary: AWOL: pNodeReporter

Antler List: pBasicContactMgr,pHelmIvP,pHostInfo,pLogger,pMarinePID
             pNodeReporter,pShare,uFldMessageHandler,uFldNodeBroker
             uSimMarine,uXMS

ProcName             Watch Reason    Status
------------------   ------------    -------
pBasicContactMgr     ANT        DB   OK
pHelmIvP             ANT  WATCH DB   OK
pHostInfo            ANT        DB   OK
pLogger              ANT        DB   OK
pMarinePID           ANT  WATCH DB   OK
pNodeReporter        ANT  WATCH DB   MISSING
pShare               ANT        DB   OK
uFldMessageHandler   ANT        DB   OK
uFldNodeBroker       ANT        DB   OK
uSimMarine           ANT  WATCH DB   OK

==========================================================================
Most Recent Events (8):
==========================================================================
[120.15]: PROC_WATCH_EVENT: Process [pNodeReporter] is missing.
[0.00]: Noted to be present: [pShare]
[0.00]: Noted to be present: [pLogger]
[0.00]: Noted to be present: [pBasicContactMgr]
[0.00]: Noted to be present: [pHostInfo]
[0.00]: Noted to be present: [uFldNodeBroker]
[0.00]: Noted to be present: [uFldMessageHandler]
[0.00]: Noted to be present: [uSimMarine]
```

Events:

- Created typically after launch time when something "notable" happens.
- Limited in quantity. (don't want the size of an appcast to grow unbounded)
- Each event is just a string.

How do you make an "AppCast-Enabled" MOOS application?

On-Demand AppCasting

To implement on-demand appcasting, a few things need to be done in each application.

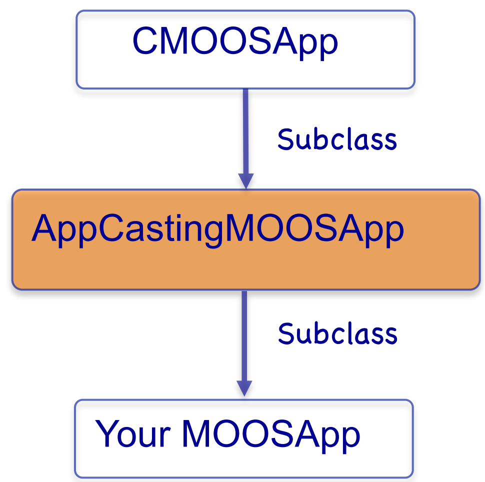
- Apps must register for `APPCAST_REQ` mail.
  An AppCast request will renew a token for some number of seconds
  Until the token expires, the app generates an appcast repeatedly.

- Even while appcasting, the app only generates an AppCast every N secs.
  The app keeps track of the last real-time appcast generation.

- Each app handles a config setting indicating whether an xterm is open.
  This setting is a global variable in the .moos config file.

So a new generic "AppCastingMOOSApp" class is used:

Minimizes boilerplate in individual apps.



Using the AppCastingMOOSApp Superclass

Six Steps

Step 1: Subclass the AppCastingMOOSApp Superclass

Step 2: Invoke two superclass methods in your Iterate()

Step 3: Invoke a superclass method when you register variables.

Step 4: Invoke a superclass method during OnNewMail().

Step 5: Invoke a superclass method during OnStartUp()

Trivial, 1-2 line changes in each case

Step 6: Implement your buildReport() function.

This is where you get to be creative about what your app reports.

Using the AppCastingMOOSApp Superclass

Your Class Definition

Step 1: Subclass the AppCastingMOOSApp Superclass

```
#include “MOOS/libMOOS/Thirdparty/AppCasting/AppCastingMOOSApp.h”
class YourMOOSApp : public AppCastingMOOSApp
{
  // All your normal class declaration stuff

  bool  buildReport();
};
```

The buildReport() function is a virtual function in the superclass. It is where you can do the work of constructing an AppCast.

Using the AppCastingMOOSApp Superclass

Modifying Your Iterate() and Registrations

Step 2: Invoke two superclass methods in your Iterate()

```
bool YourMOOSApp::Iterate()
{
  AppCastingMOOSApp::Iterate();

  // Do all your normal Iterate stuff

  AppCastingMOOSApp::PostReport();
  return(true);
};
```

Updates the current MOOSTime, and # of iterations.

Determines if an AppCast is warranted, and invokes buildReport() if so.

Step 3: Invoke a superclass method when you register variables.

```
void YourMOOSApp::registerVariables()
{
  AppCastingMOOSApp::RegisterVariables();

  // Do all your other registrations
}
```

The superclass will register for APPCAST_REQ, indicating another app, like uMAC, is interested in appcasts from this app.

Using the AppCastingMOOSApp Superclass

Modifying Your OnNewMail() and OnStartUp()

Step 4: Invoke a superclass method when you handle mail.

```
bool YourMOOSApp::OnNewMail(MOOSMSG_LIST &NewMail)
{
  AppCastingMOOSApp::OnNewMail(NewMail);

  // Do all your other normal mail handling.
}
```

The superclass will handle the APPCAST_REQ mail.

Step 5: Invoke a superclass method during OnStartUp()

```
void YourMOOSApp::OnStartUp()
{
  AppCastingMOOSApp::OnStartUp();

  // Do all your other startup stuff
}
```

The superclass will register for APPCAST_REQ, indicating another app, like uMAC, is interested in appcasts from this app.

END