SIAM: Software Infrastructure and Applications for MOOS

Tom O’Reilly

September 13, 2004
MOOS Mooring

- Interdisciplinary ocean observatory
- Moored network of surface, midwater, and benthic instrument nodes
- High power and data rate to sea floor
  - 30+ Watts
  - 10 BaseT, TCP/IP
- Instruments can be accessed in near-realtime over wireless link (satellite, RF)
MOOS Mooring

- System can respond *autonomously* to interesting events
  - With or without shore involvement
  - Modify sampling schedules and strategies
  - Launch AUV

10 km
SIAM deliverables

- Data acquisition and onboard logging of data and associated metadata
- Remote instrument control
- *Plug-and-work* self-describing instruments
- Autonomous mooring management
SIAM deliverables

- Utilities for system integration, deployment, maintenance
- User interfaces for instrument and mooring access
- SIAM does not include shore-side database
  - See MOOS Shore-Side Data System project
Big problems addressed by SIAM

- Support many kinds of instruments
- Remote control of instruments by human operators
- Autonomous interactions with instruments - *event detection and response*
- Instrument integration and configuration
- Association of acquired science data with contextual metadata
MOOS characteristics

• Multiple (>20) moorings and benthic nodes, each node hosts multiple (>10) instruments

• Many kinds of instruments (> 50)
  – “Legacy” instruments
  – Commercial instruments
  – Custom-built instruments
  – Mostly serial RS-232, TCP/IP coming soon

• System evolves; new instruments added, old instruments removed
MOOS characteristics

- Intermittent communications with shore
  - Wireless links
  - Power limitation

- High degree of onboard autonomy required
SIAM architecture

As implemented and deployed today...
Instrument services

• Enables clients to interact with instruments from anywhere on the network
  – Human clients - e.g. scientist controls instrument from user interface
  – Software clients - e.g for event detection and response

• System policies can be consistently enforced by services
  – E.g. power management

• Very flexible “building blocks”
Instrument services

• Controls and configures instrument
  – At startup
  – On client command
  – In response to event (e.g. adaptive sampling)

• Retrieve, log, and distribute instrument data across network

• Reflects current instrument state
  – Control settings, sampling rates, etc
  – Fault conditions
Instrument service

standard interface

• All instrument services implement SIAM standard instrument interface, including generic methods to
  – initialize instrument
  – get instrument status
  – acquire data sample
  – shutdown instrument

• Thus, all MOOS instruments can be accessed in a standard way
Standard interface and service implementations

Define standard operations accessible to clients

Instrument

extends

ADCP (additional methods)

Add operation to set bin size

implements

CTD service

GPS service

Fluoro service

ADCP service

Implementation details hidden from clients
Why **standard** software interfaces?

- We don’t need to develop standard software interfaces if
  - Instruments never interact autonomously with other instruments or subsystems
  - Instruments are always installed, configured, controlled, and sampled manually
  - Observatory operators have detailed knowledge of all instruments

- But the above characteristics are the *antithesis* of the system we want!
Why standard software interfaces?

• Heterogeneous components can interact
• Standard generic user utilities
  – Utilities can access instruments without “knowing” details of instrument serial interface
  – Greatly simplifies utility software
• System maintenance, trouble-shooting
  – Display state of instruments to system operators
  – Standard methods of starting and shutting down instruments, etc.
Why **standard** software interfaces?

- Simplifies autonomous interactions
  - Onboard resource management (power, data storage and telemetry)
  - Installing and removing instruments (e.g. standard way to start and stop service)
  - Event detection and response
    - Autonomous event detector component could retrieve data readings from multiple sensors
    - Standard notification interface
    - Standard way to set sampling schedule
Example: no standard interface

Data logger

Time synchronizer

"acquire"

"ts"

"time="

"mmddyy=…"
Example: with standard interface

Data logger

HydroRad instrument service

Time synchronizer

SBE-16 instrument service

HydroRad

SBE-16

"acquire"

"time=...

"ts"

"mmddyy=..."
Problems addressed by distributed standard interfaces

- Support many kinds of instruments
- Remote control of instruments by human operators
- Autonomous interactions with instruments - *event detection and response*
- Instrument integration and configuration
- Association of acquired science data with contextual metadata
Automatic configuration
Adding an instrument...

• Install instrument software on observatory node
  – Transfer software to node
  – Associate software with instrument port
  – Configure port, etc...

• Associate metadata with data stream - crucial for data interpretation

• Increasing challenge with number and variety of instruments
Solution: instrument pucks and self-describing data streams

- Puck contains instrument service software and descriptive information, physically attached to instrument
- Node retrieves puck contents when pucked instrument is plugged in, runs the instrument service
- Service generates data stream, adds descriptive information into data stream
• Simple, small storage device
• Puck is closely coupled to a specific instrument, always travels with its instrument
• Stores instrument service software and metadata (unique ID#, serial number, calibration coefficients, parsing instructions, etc)
Instrument puck concept

- Service software and metadata are retrieved by node when puck and instrument are plugged into node’s serial port
- Node runs the instrument service
Moored network

Node

Scheduler

Port manager

Instrument service

Plug in new puck’ed instrument

Plugging in an instrument at sea
Port manager applies power to port
Operator runs `scanPort`
Port manager extracts service code, XML metadata, etc from puck
Port manager starts instrument service

Service joins network

Instrument passes sampling schedule to scheduler
Instrument puck concept

• Simplifies system integration and maintenance
  – Scalable to large observation network

• Addresses critical problem of data traceability and context

• Any serial instrument - off-the-shelf, custom - can be outfitted with a puck
Puck design philosophy

• Encourage adoption by oceanographic community and instrument vendors

• Keep it simple, small, and cheap

• Straightforward mechanical, electrical, software interfaces

• Content-neutral
  – Can store simple text, IEEE 1451 data sheet, Java bytecode - puck doesn’t “care”
External puck implementations

Deep puck (4000 meters)

Molded puck (50 meters)
Embedded pucks

- Puck protocol implemented by electronics inside the instrument housing
- No need for external connectors
- Vendors are interested in this, if standard puck protocol can be established

MBARI MOOS serial ADC board
Embedded Persistor CF2
Puck consortium

- Forum for advancement and adoption of puck technology
- Facilitates the development of puck-enabled instruments and platforms, promotes benefits of puck technology to the oceanographic community at large
SIAM data streams

• While instrument service runs, it generates SIAM packets, and writes them to onboard log in time-sequential order

• Each packet contains
  – Instrument ID - unique integer key assigned to each instrument by SSDS before deployment
  – Time-tag - time at which packet was created
  – Sequence number - increments with each packet generated (to detect missing packets)
SIAM data streams - packet types

- Sensor data - raw and/or processed
- Metadata (XML and binary)
  - What kind of instrument produced the data
  - Instrument calibration coefficients
  - Where is instrument located
  - Data field descriptions, how to parse data
  - Current state (e.g. sampling parameters, gains)
- Text message - human-readable ascii text
  - User can send message to instrument service
Self-describing SIAM data stream: example

- Service started
- User changed instrument setting
- User sent text message
- Service detected state change

- = XML doc (from puck)
- = text message
- = state metadata
- = sensor data
Retrieval of data streams

- SIAM *portal* contacts node when wireless link is available, retrieves data packets, and distributes to Shore Side Data System (SSDS)
1. Instrument services acquire and log packets

2. Node periodically turns on radio, notifies portal

3. Portal retrieves packets created since last retrieval (through instrument service)

4. Portal publishes packets to SSDS

---

**Diagram Description:**

- **SSDS** sends a message to the **portal** indicating that the **radio link is up**.
- **Portal** retrieves packets created since the last retrieval (through instrument service) and publishes them to the **SSDS**.
- **Moored network** communicates with the **portal** through the **radio link**.
- **Shore network** also communicates with the **portal** through the **radio link**.
5. Node turns off radio when retrieval finished
6. Instrument services continue to acquire and log data
Problems addressed by pucks, SIAM data streams, and portal

• Support many kinds of instruments
• Remote control of instruments by human operators
• Autonomous interactions with instruments - event detection and response
• Instrument integration and configuration
• Association of acquired science data with contextual metadata
SIAM extensions, enhancements

- Simplify instrument service software development
  - By default a generic instrument service interacts with instrument based on user-supplied data sheet (ala IEEE 1451.x)
- Define and implement event detection and response experiment
- *Embedded puck* implementation
- Refine metadata and state models
- Incorporate calibration operations into service interface
SIAM goes to MARS?
Many MARS requirements TBD

- What kinds of instruments will be accommodated?
- How will users interact with their instruments?
- Scalability - how many nodes and instruments?
- Autonomous interactions between instruments?
- Metadata tracking and binding to data?
SIAM on MARS?

- Beneficial to have some commonality between moored and cable-to-shore software components
- Some SIAM components could be portable to MARS
- Instrument service classes are designed to be independent of host details
  - Abstractions
  - Implemented in Java
Instrument Service is insulated from details of host environment by interfaces
On MOOS, instrument services execute on MOOS mooring controller.
On MARS instrument services execute on MARS shore station

Interacts with instrument via MARS communication and power protocols
SIAM on MARS

- All SIAM components are on shore

MARS network

Serial instrument commands

Instrument data

Instrument
MARS node, SIIM
Shore computer
Instrument service
MARS node service
Client

Node 1

Node 2

Shore side data system

MARS shore workstation

I-1 I-4 I-5 I-6

I-2 I-5 I-6

I-1 I-2 I-3

I-4 I-5 I-6

N-2

N-2

client

client

client

NSF Sensors Workshop 2004
Pucks on MARS

• Could readily implement plug-and-work on MARS:
  – *port manager* would run on MARS shore workstation, retrieve instrument service code and metadata from deployed instruments via MARS remote port, then execute service…
Coexisting network applications

- SIAM could be an option for a MARS instrument, but not required
- SIAM-enabled instruments could coexist on MARS network with other schemes
  - E.g. some instruments may just be controlled over terminal server by manufacturer’s Windows GUI
Questions?
Extra slides
MOOS Mooring Project

- Portable multi-disciplinary science platform
  - Air-sea interface to sea-floor
- Near real-time telemetry to shore
- Event response
- Long-duration, high-availability
- High power and data rate from surface to seafloor
MOOS Mooring

• System can respond autonomously to interesting events
  – Modify sampling strategies
  – Launch AUV

• Nodes are “self-configuring” - plug-and-work
MOOS Mooring

- Moored network of surface, midwater, and benthic instrument nodes

- Instruments can be accessed in near-realtime over wireless link (satellite, RF)

- System can respond autonomously to interesting events
  - Modify sampling strategies
  - Launch AUV
Instrument services

- Implemented by Java RMI distributed objects
- Presents *standard instrument interface* to clients (locally and across network)
- Clients only interact with an instrument through its instrument service
  - Service hides instrument’s serial port and implementation details; clients
Why **standard software interfaces**?

- Data acquisition, archiving and retrieval
  - Generic way of retrieving data from instruments to shore
  - Standard interfaces enable generic query and retrieval from data archives
SIAM MMC node components

MMC node

Node Service

Instrument service

Instruments and pucks

Moored network
Pre-deployment puck configuration

- Attach a puck to the instrument, attach pucked instrument to operations workstation
- Assign unique ISI ID, fill out XML metadata form
- Select appropriate instrument service code
- Write bytes to the puck
/** Simple distributed object interface */
interface Benchmark {

    /** No parameters, no return value. */
    void emptyTest();

    /** Primitive parameters and return value. */
    long primitiveTest(short a,
                       double b,
                       long c);

    /** Returns an object. */
    TObject objTest(byte fillValue);
}

/** Simple object returned by Benchmark.objTest() */
class TObject {
    short x;
    long y;
    double z;

    byte array[] = new byte[1000];
}
### Benchmark RMI overhead

<table>
<thead>
<tr>
<th>Step Description</th>
<th>Bytes Transferred</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>application</strong></td>
<td><strong>ppp</strong></td>
</tr>
<tr>
<td>Step 1: get distributed object service = (BenchMark )Naming.lookup(url);</td>
<td>N/A</td>
</tr>
<tr>
<td>Step 2: service.emptyTest();</td>
<td>0</td>
</tr>
<tr>
<td>Step 3: long ret = service.primitiveTest(short sval, double dval, long lval);</td>
<td>26</td>
</tr>
<tr>
<td>Step 4: TObject obj = service.objTest(byte bval);</td>
<td>1019</td>
</tr>
<tr>
<td>Step 5: disconnect from service System.exit()</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>204</td>
</tr>
<tr>
<td></td>
<td>1432</td>
</tr>
<tr>
<td></td>
<td>1143</td>
</tr>
<tr>
<td></td>
<td>341</td>
</tr>
<tr>
<td></td>
<td>196</td>
</tr>
</tbody>
</table>
### MOOS Test Mooring telemetry

<table>
<thead>
<tr>
<th>Data stream</th>
<th>Sample interval (minutes)</th>
<th>Total samples retrieved</th>
<th>Total bytes retrieved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triaxys wave sensor</td>
<td>30</td>
<td>4922</td>
<td>1941049</td>
</tr>
<tr>
<td>ASIMET short wavelength radiometer</td>
<td>10</td>
<td>15437</td>
<td>5159930</td>
</tr>
<tr>
<td>ASIMET long wavelength radiometer</td>
<td>10</td>
<td>15436</td>
<td>5994108</td>
</tr>
<tr>
<td>ASIMET wind sensor</td>
<td>10</td>
<td>15398</td>
<td>5793036</td>
</tr>
<tr>
<td>ASIMET relative humidity and temperature</td>
<td>10</td>
<td>15437</td>
<td>5422523</td>
</tr>
<tr>
<td>Garmin GPS</td>
<td>10</td>
<td>15431</td>
<td>5975157</td>
</tr>
<tr>
<td>RDI Workhorse ADCP</td>
<td>60</td>
<td>2574</td>
<td>4303122</td>
</tr>
<tr>
<td>Mooring power can</td>
<td>10</td>
<td>15448</td>
<td>52017200</td>
</tr>
<tr>
<td>Main environmental</td>
<td>10</td>
<td>15431</td>
<td>2114895</td>
</tr>
<tr>
<td>EOM cable diagnostics</td>
<td>10</td>
<td>15461</td>
<td>7379389</td>
</tr>
<tr>
<td>Node diagnostics</td>
<td>60</td>
<td>2648</td>
<td>21146895</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>N/A</strong></td>
<td><strong>133623</strong></td>
<td><strong>138394199</strong></td>
</tr>
</tbody>
</table>

Measurement interval: 108 days (April 23 through August 9, 2004)
Total Globalstar connections: 3487
Total broken connections: 187 (5%)
Average download duration: 127 ± 66 seconds
Total connection time: 182 hours