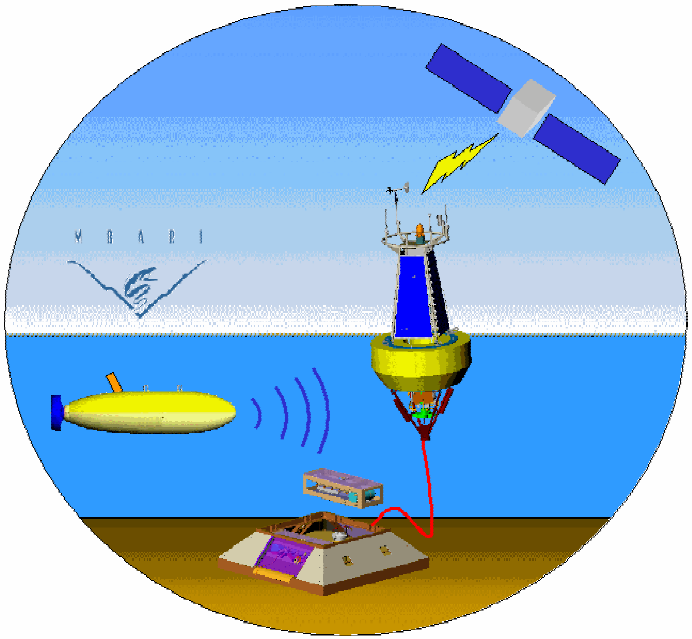
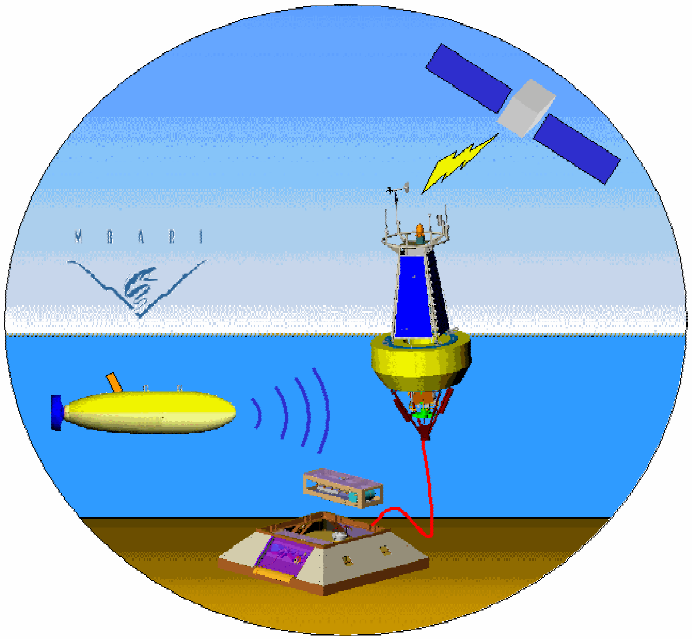
SIAM Design Review



November 17, 2003

SIAM Project Development



Tom O'Reilly

M B A R I

Deployed component operators

Deployed

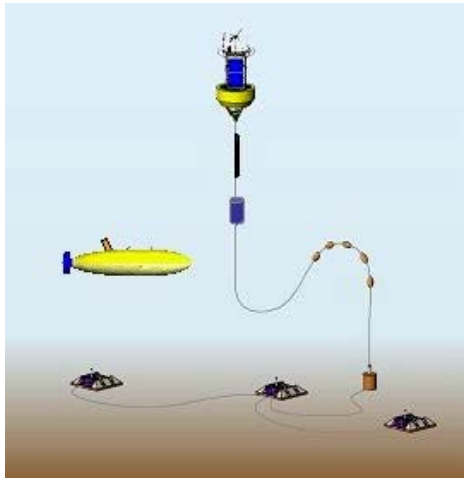
commands

telemetry

Operations

commands

telemetry



telemetry

device descriptions

SSDS

queries

processed data

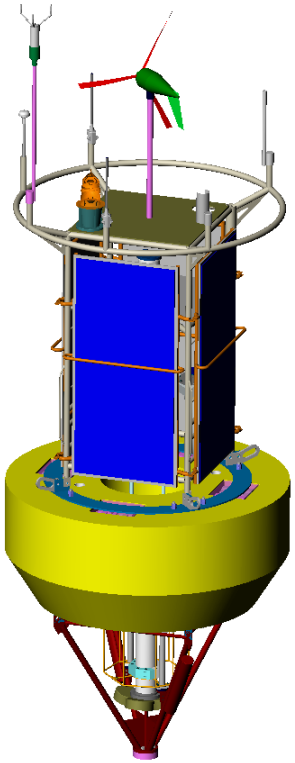
Science users

MOOS Subsystems

MBARI

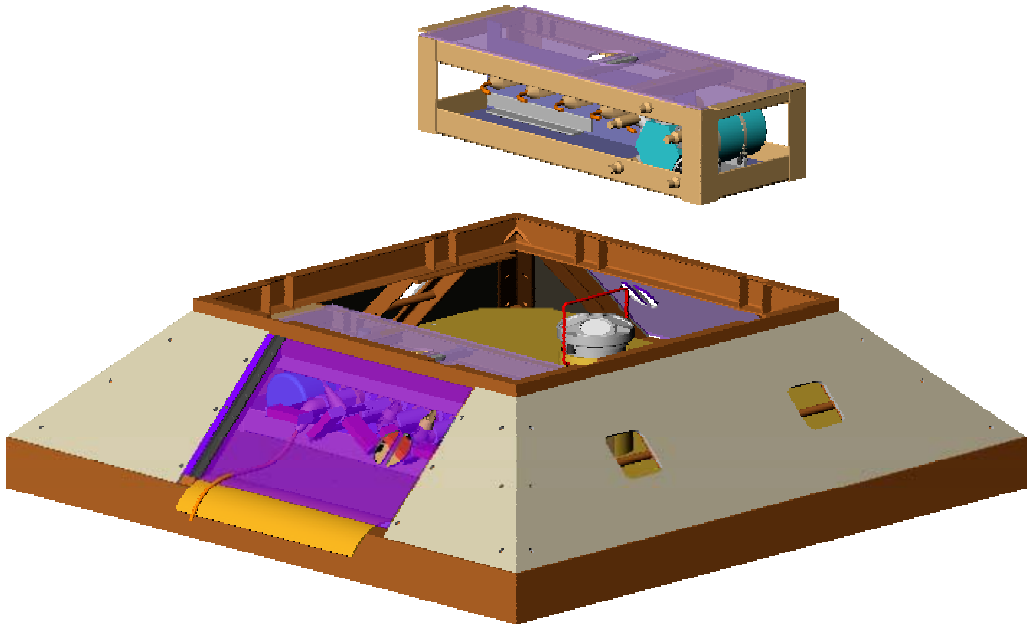
SIAM deliverables

- Instrument control
- Data acquisition
- Autonomous event response
- Self-describing instruments: *plug-and-work*



SIAM deliverables

- Utilities for system integration, deployment, maintenance
- Shoreside portal gateway
- Instrument user interfaces



MOOS Mooring Controller

- MMC - not SIAM - supplies the following:
  - Node hardware
  - Optical communications interface
  - Linux kernel support, Linux device drivers
  - Time-keeping and synchronization

Object-oriented design and implementation

- Many benefits to object oriented approach...
  - Manages system complexity
  - Robust
  - Rapid development
  - Reusable code

- SIAM application is implemented in Java
  - Can run on any platform with Java CDC profile

Instrument Software Infrastructure (ISI)

- Precursor to SIAM
- Developed concepts, prototypes, and deployed infrastructure for MOOS mooring applications
- August, 2001: ISI Concept Review
  - Presented distributed object architectures, instrument puck proof-of-concept
  - Instrument vendor and NEPTUNE attendees

Instrument Software Infrastructure (ISI)

- Elements of ISI utilized by *New Mooring Controller* software
  - Two at-sea engineering deployments in 2002
- ISI software package is used by SIAM

Software development methodologies

- Software development is very different in some ways from other engineering development
  - Predetermining requirements can be extremely difficult
  - Deterministic testing of software can be extremely difficult
  - Software perceived as “easily” modified

Software development methodologies

- Single-pass waterfall approach - specify requirements, design, construct - not appropriate for most software projects
- *Agile methodologies* developed in late 1990s by Kent Beck, Martin Fowler, and others
  - AKA “iterative” or “adaptive” methodologies
  - Formalizes pre-existing concepts

Agile methodologies

- Accept that requirements and scope will evolve during development
- Work closely with users to iteratively formulate requirements
- Frequently produce working versions of system for evaluation by users

Agile methodologies

- De-emphasize formal requirements and design documentation
  - Coding is designing
- Less bureaucratic, more realistic, more productive
- Adaptive software development approach first proposed for MOOS at 1999 *MOOS Workshop* (Paul Rogers)

eXtreme Programming (XP)

- XP is a popular agile methodology
- Short development “sprints”, with stable requirements for each sprint
- Brief daily meetings (“scrums”) of development team
- Validate each revision with automated test procedures

SIAM development

- Based on NMC and SSDS experience, SIAM has adopted several XP practices:
  - Team of committed developers and end-users collaboratively steers development
  - Short software release cycles (~2 weeks)
    - Releases evolve toward deployable system
  - User evaluation/feedback to developers on each cycle

Close collaboration with operations and science users

- Better understanding of user requirements
  - Requirements can be iteratively clarified or even changed
- User feedback provides “course-correction” during development
- Helps insure that system is truly useful and usable
  - Aids transition to operational status

User stories

- Users and developers collaborate to write brief stories of how system works, e.g.
  - *Shutting down a port guarantees that power is disconnected from the port.*
- Users and developers communicate to get better mutual understanding of stories
- Developers implement the stories during development cycle

SIAM users in 2003

- Operations - Mike Kelley, Zorba
- Science - Francisco Chavez, Eric Reinecker
- Support Engineering - Craig Okuda

Frequent releases which actually work

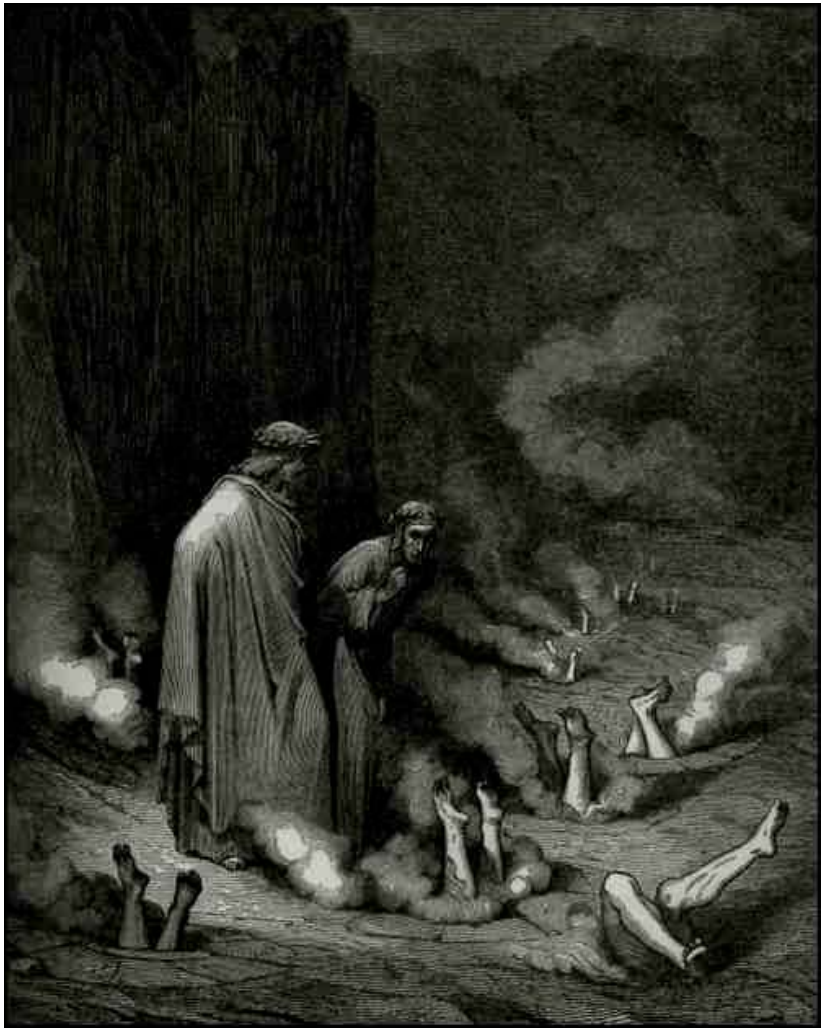
- Quick user feedback on latest software
- Easier to track progress
- Usually easier to improve a working system
- Easier to integrate for actual deployments

Eschew future-proofing

- Focus on near-term user stories
  - Users often don't know what they will *really* want in the future
- Assume that the simplest solution is the best solution
- Use object-oriented techniques to make future re-factoring easier

Team work area

- Highly-focused team, co-located in same work area
- Easy to communicate and address problems as they arise
- Daily morning *scrum*; monitor progress, plan day's work



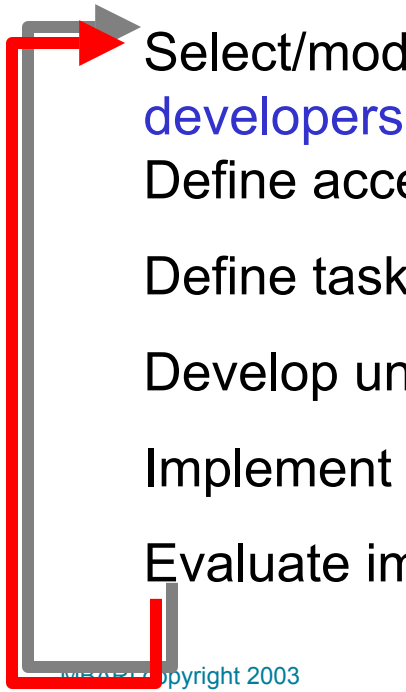
MBARI

XP development process

Compose user stories - developers, users

Rough out implementation sequence which meets project milestones - developers, users

On each 2-week development cycle:



Select/modify/invent stories for implementation on this cycle - developers, users

Define acceptance test for each story - developers, users

Define tasks, estimate work for each story implementation - developers

Develop unit tests - developers

Implement selected stories - developers

Evaluate implementations; correct and complete? - developers and users

Deployments: as often as possible and practical

- Invaluable first-hand experience for development team
  - Understanding of operating environment
  - Drives team to develop a robust and usable system

Why “parking lot” deployments?

- Verify system functionality under realistic conditions
- Discover integration problems well before actual at-sea deployment
- Excellent mobilization practice

Some issues

- XP requires involved and committed end-users
  - But most of our users have many other tasks to do
- Need wider science representation
- Balance between too much and too little user input

Some issues

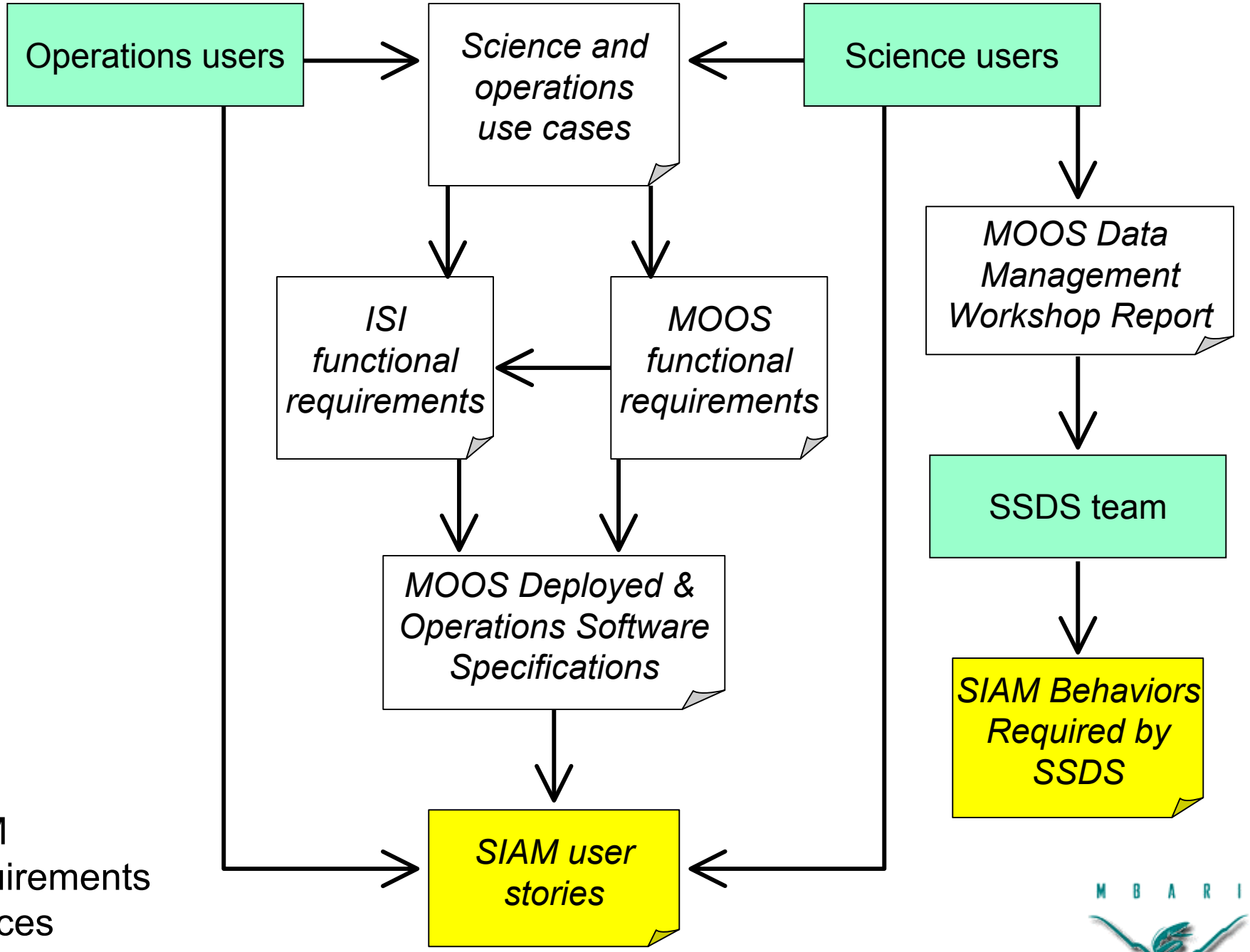
- Automated testing - key XP concept, vital to verify correct functionality of system
  - Takes some effort to get this started
  - SIAM project needs to work harder in this area

Some issues

- XP requires developers to focus intently on the project
  - Context switching between projects can be difficult

SIAM primary requirements documents

- *SIAM User Stories*
- *SIAM Behaviors Required by SSDS*
- *Power Management Software for MOOS: Functional Requirements*



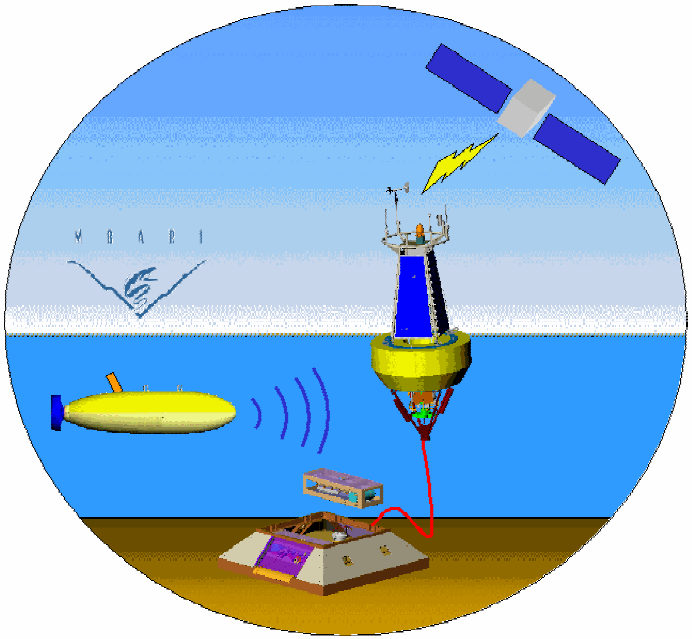
SIAM Requirements Sources

MBARI

Design review

- In accordance with the XP approach, this review isn't strictly a PDR or a CDR
  - Many concepts have been designed, implemented, tested, and deployed
  - Other concepts are very preliminary
- When presenting designs, reviewers will refer to relevant requirements

SIAM Architectural Overview

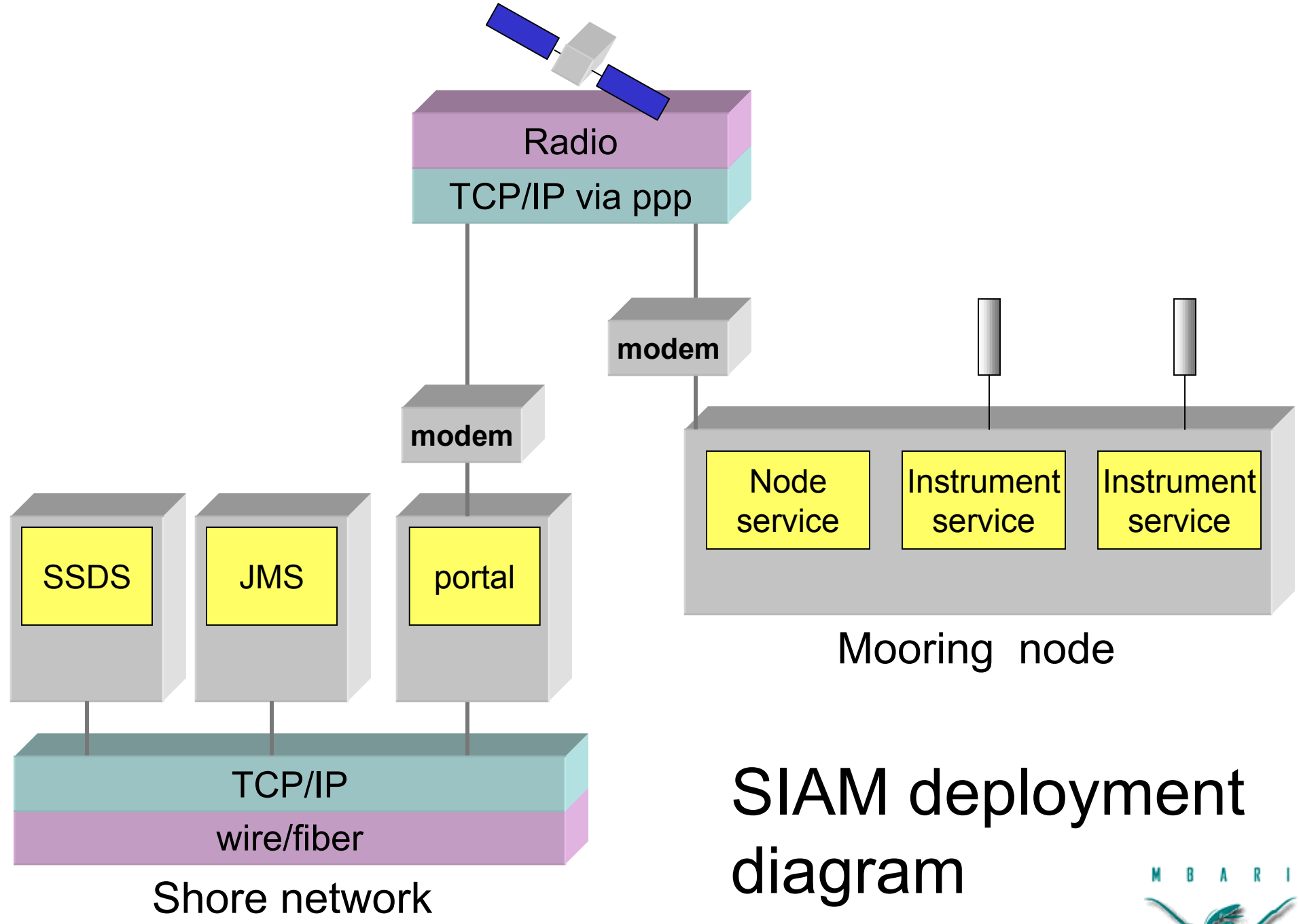


Tom O'Reilly

M B A R I

Primary requirement sources

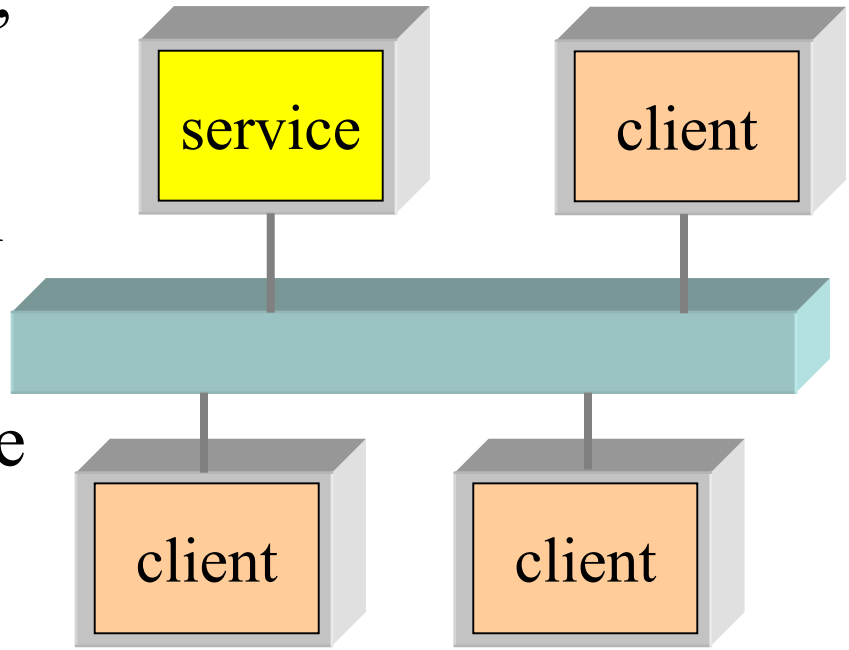
- *SIAM Behaviors Required by SSDS:*
  - numerous requirements
- *MOOS Deployed Subsystem Software Requirements Specification*
  - numerous requirements
- *MOOS Operations Subsystem Software Requirements Specification*
  - numerous requirements



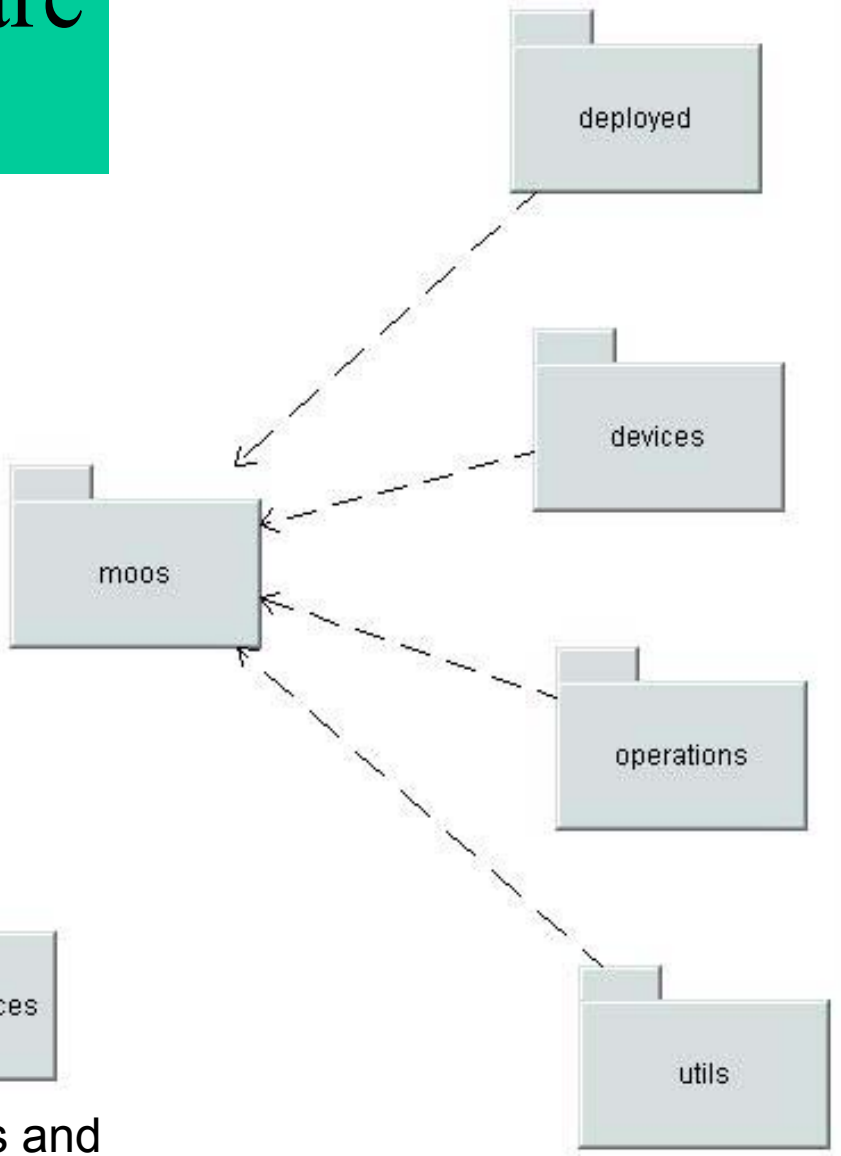
SIAM deployment diagram

Service architecture

- *Service:*
  - Instrument device control, data retrieval, etc
- Service awaits requests from *client*
- Servers and clients can reside anywhere on network



SIAM software packages



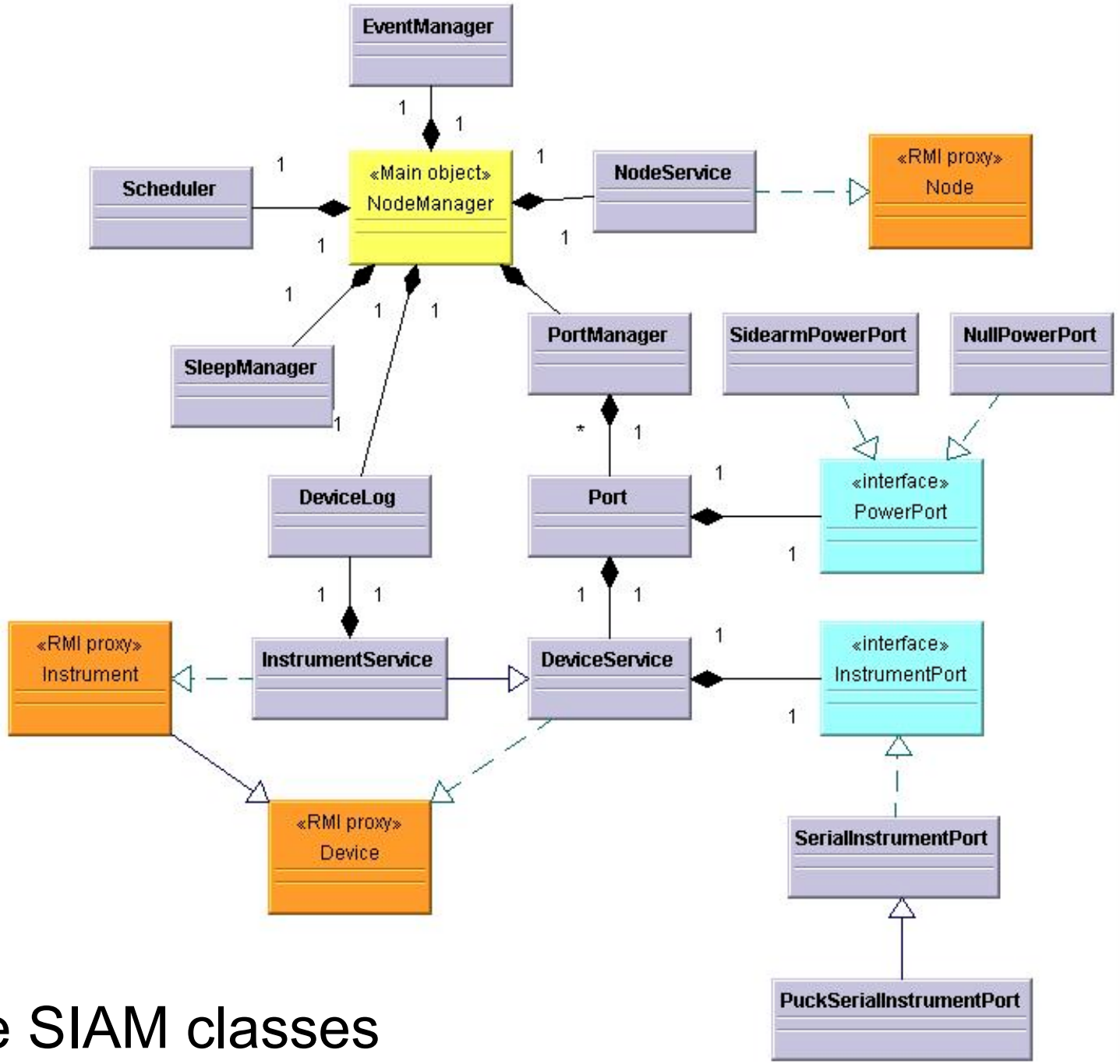
Node infrastructure, instrument service framework

Instrument service implementations

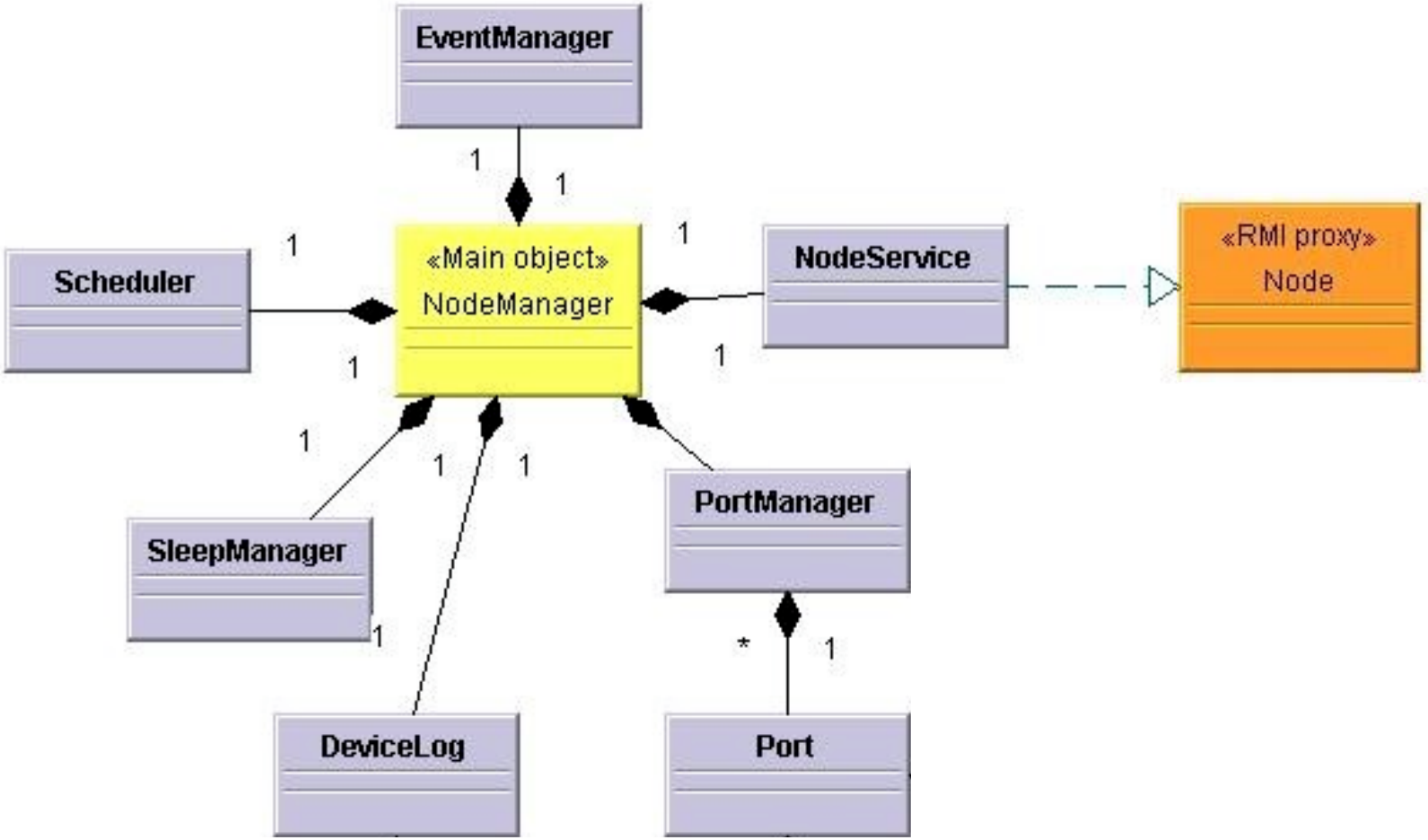
Portal, user interfaces and utilities

General utility classes

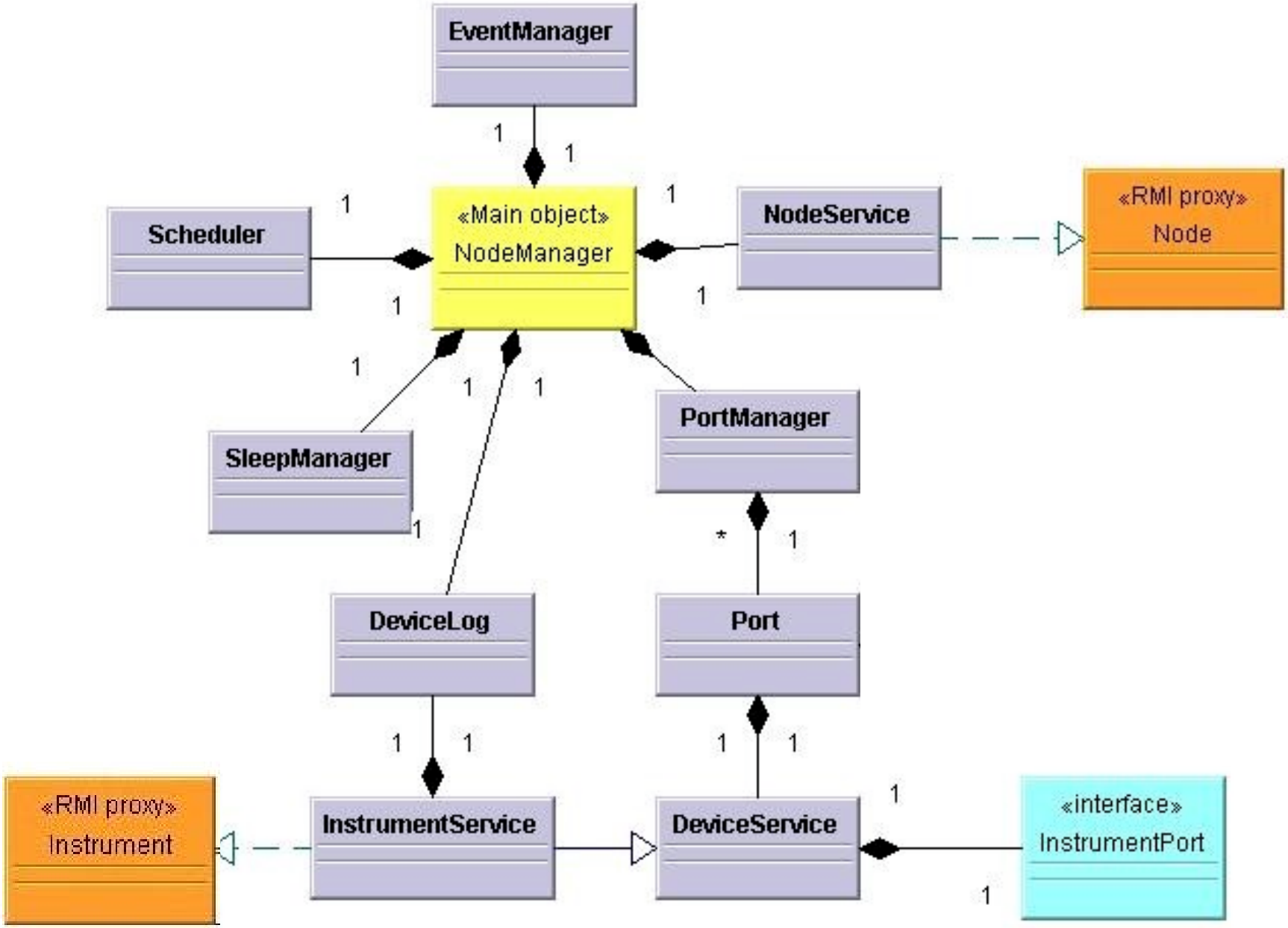
ISI interfaces and container classes



Some SIAM classes



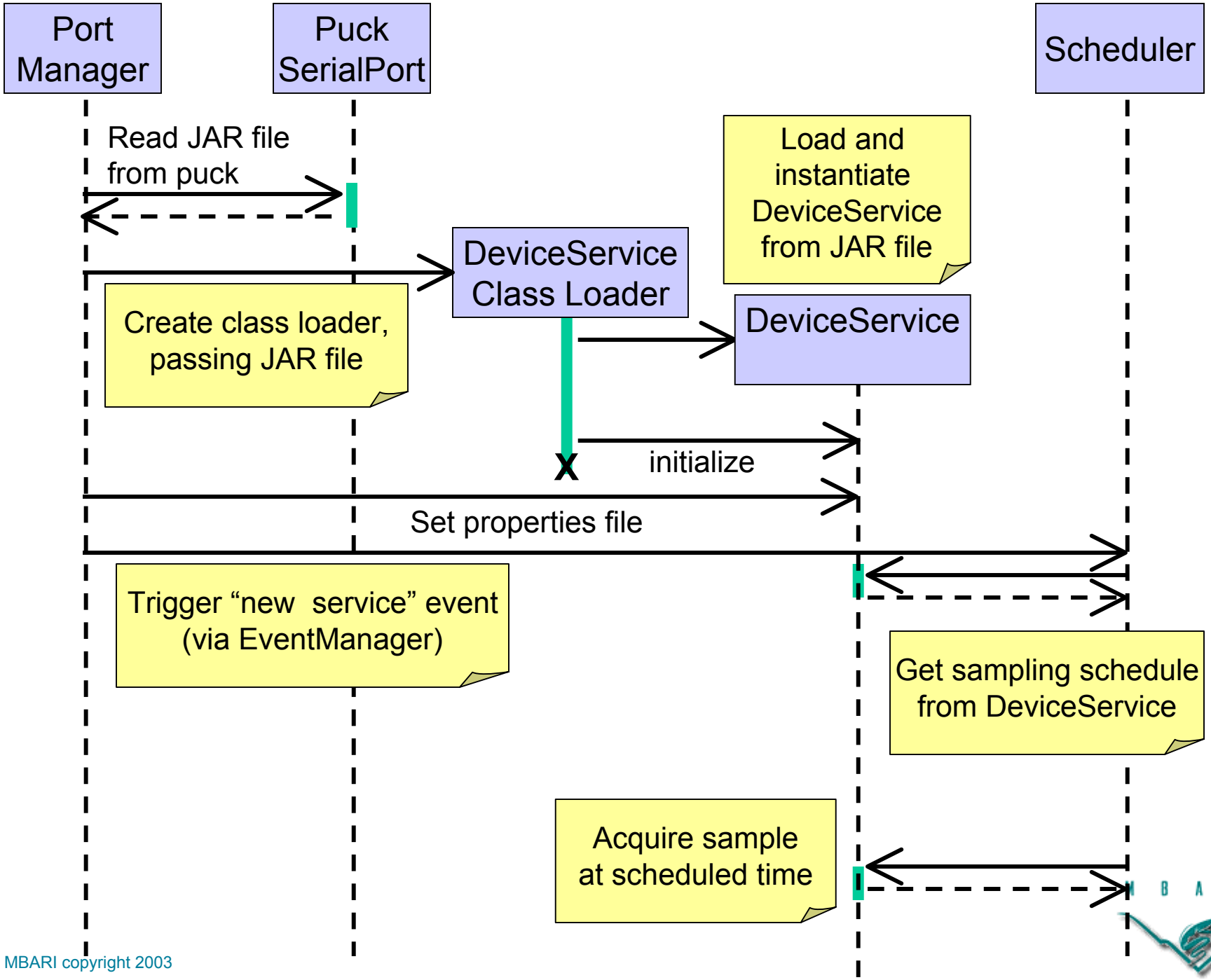
Classes instantiated at start-up



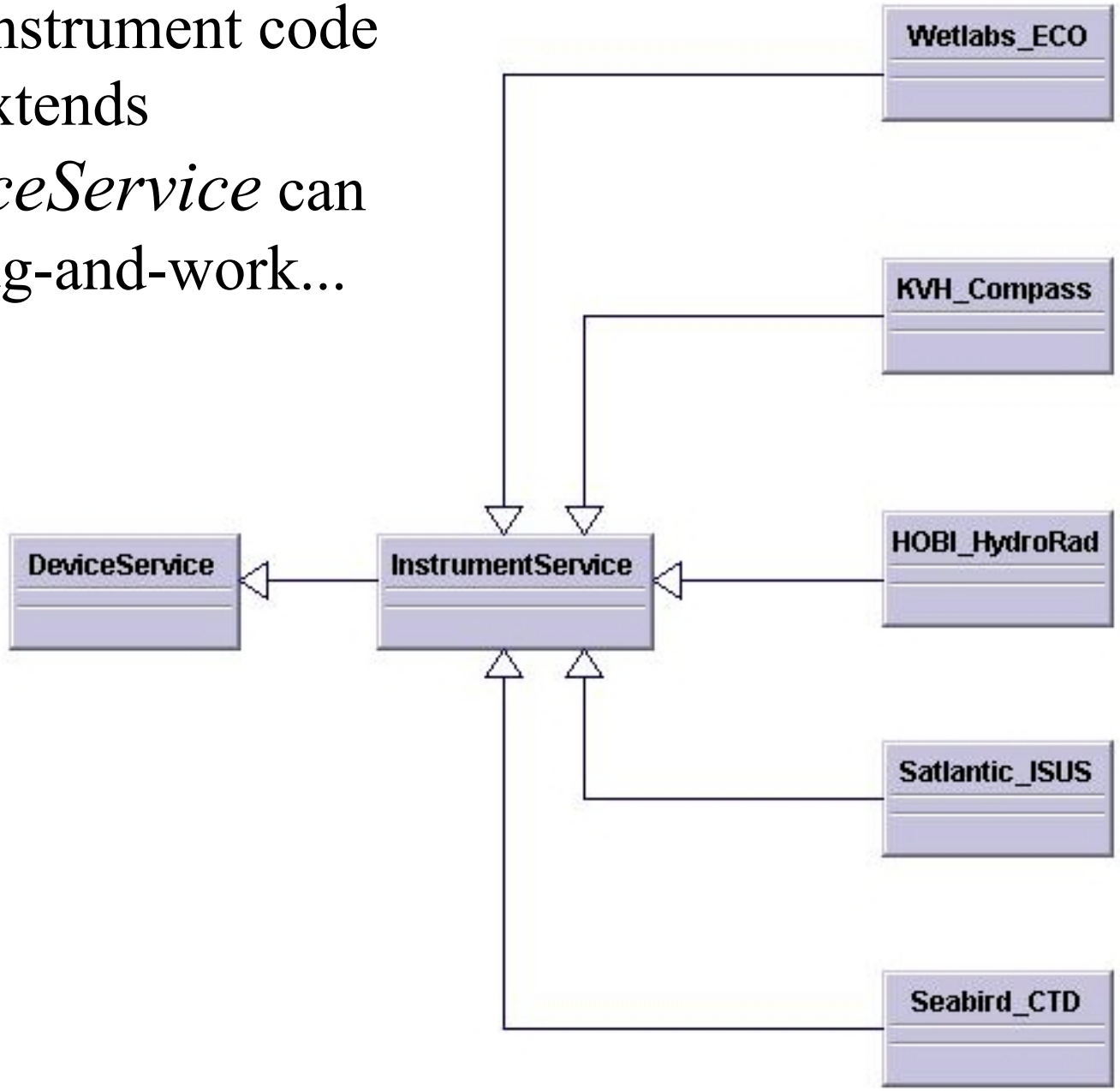
Classes instantiated during plug-and-work

Plug-and-work instruments

- PortManager is responsible for plug-and-work:
  - Contacts puck and retrieves JAR file
  - Loads instrument service class from JAR file
  - Invokes service initialization method



- Any instrument code that extends *DeviceService* can be plug-and-work...

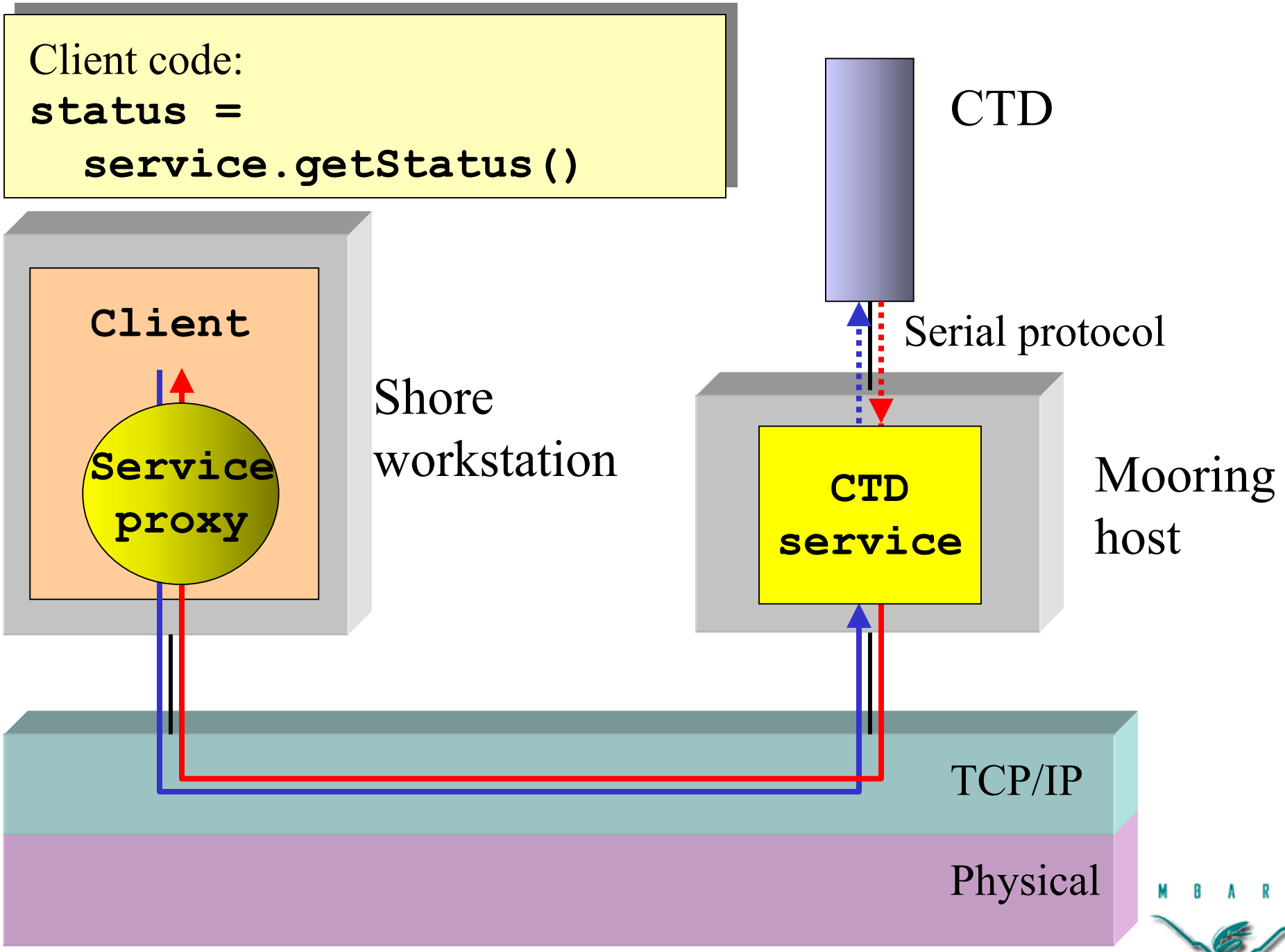


Distributed objects

- Object-oriented services
- Uniform high-level protocols (e.g. TCP/IP)
- Service interface is distributed across network through *proxy* objects
  - Client obtains service proxy object
  - Client invokes service methods on proxy
  - Proxy marshalls input, transmits requests to service
  - Service executes method, marshalls return values and transmits back to client
  - Proxy looks almost like any local object

Client code:

```
status =
   service.getStatus()
```

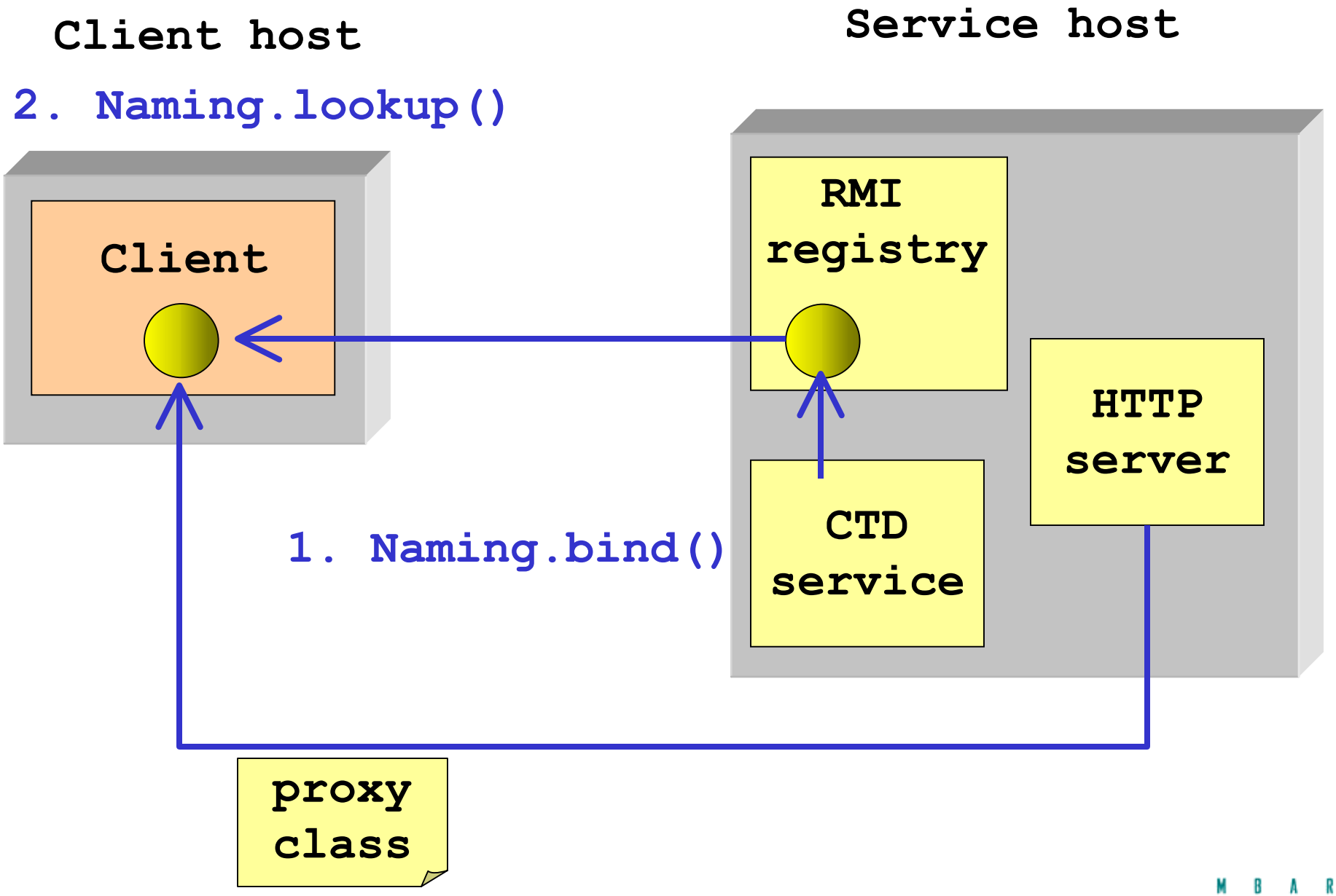


Distributed objects

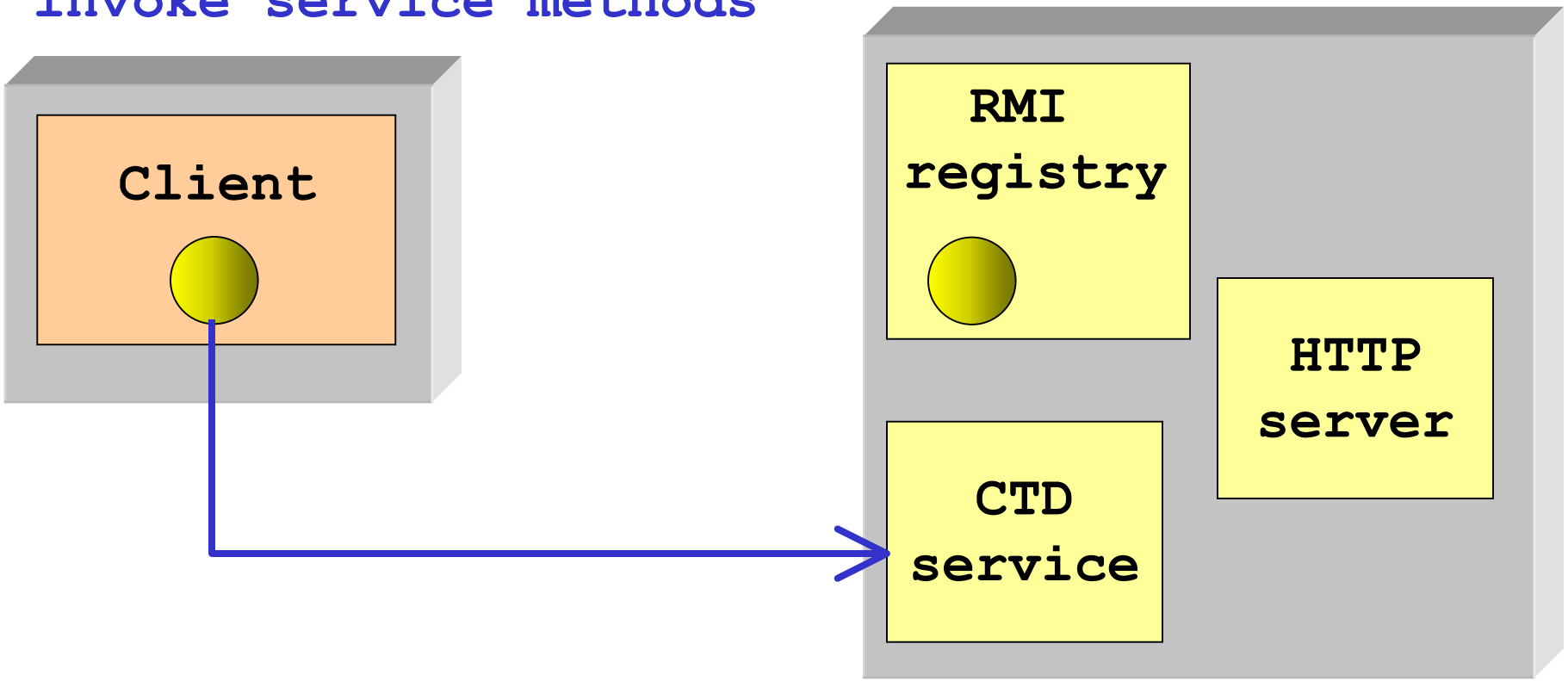
- Enable remote control and data retrieval from deployed devices
- Benefits of object orientation
- Distributed object model is well-suited to networked nature of MOOS

Java RMI distributed objects

- RMI - *Remote Method Invocation*
- TCP/IP-based protocols
- Services and clients must be implemented in Java
- Actual Java objects can be transferred over the network



invoke service methods



Example RMI code

```
// Get node proxy from URL specified on
// command line
String nodeURL = args[0];
try {
  node = (Node )Naming.lookup(nodeURL);
  // Invoke node methods…
  […]
}
catch (RemoteException e) {
  // Handle exception; network failed, etc…
  System.err.println("Got RemoteException:" + e);
}
```

RMI bandwidth costs

- Initial connection overhead
  - TCP socket initialization, etc.
- Primitive arguments and return values marshalled without padding
- Objects can be passed as parameters or return values
  - Class descriptive data as well as object data may need to be transmitted

Benchmark RMI interface

```
public interface Benchmark1 extends Remote {

  // No arguments, no return value
  void emptyTest() throws RemoteException;

  // Primitive arguments and return value
  long primitiveTest(short a, double b, long retvalue)
                        throws RemoteException;

  // Return a serialized object
  Struct1 structTest(byte fillValue)
    throws RemoteException;

}
```

Benchmark test object

```
// Struct1 object returned by Benchmark.struct1Test()
public class Struct1 implements Serializable {
  public short x;
  public long y;
  public double z;
  public byte array[] = new byte[1000];
}
```

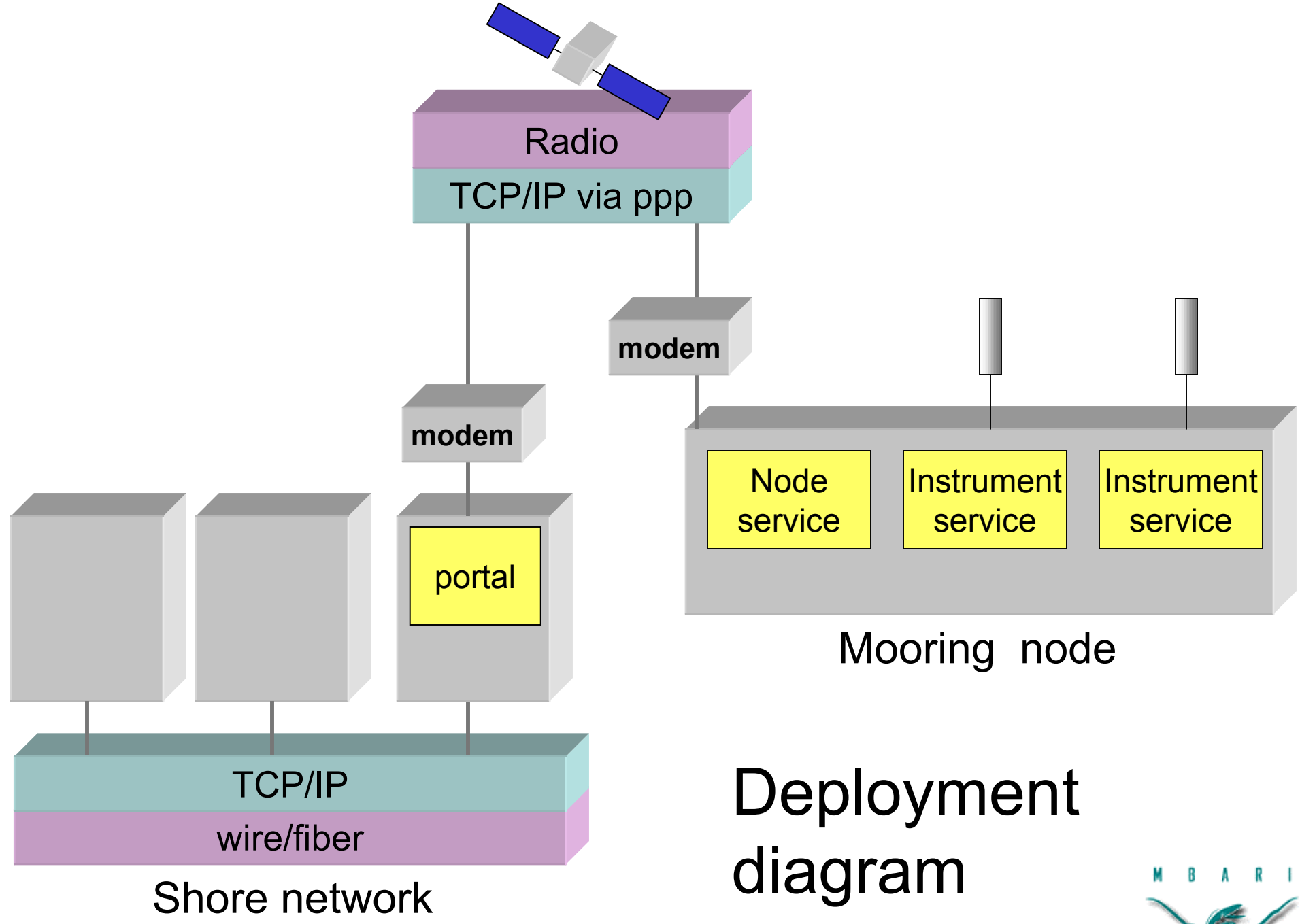
RMI benchmark tests

- Connect client and service hosts with null serial cable
  - Inline device to measure bytes transferred
- Run ppp protocol over serial connection
  - Simulated RF connection
  - Ran ppp with compression enabled

RMI data volumes

	Nominal payload* (bytes)	JRMP TCP/IP layer (bytes)	JRMP PPP layer (bytes)	Nominal transmit time @7800 bps (msec)
Connection and proxy download	346	2782	937	961
emptyTest()	0	377	178	182
primitiveTest()	26	403	176	180
structTest()	1018	1571	225	231
Shutdown	-	416	141	145

* Payload for proxy is the class size. Payload for methods is nominal size of arguments plus return data



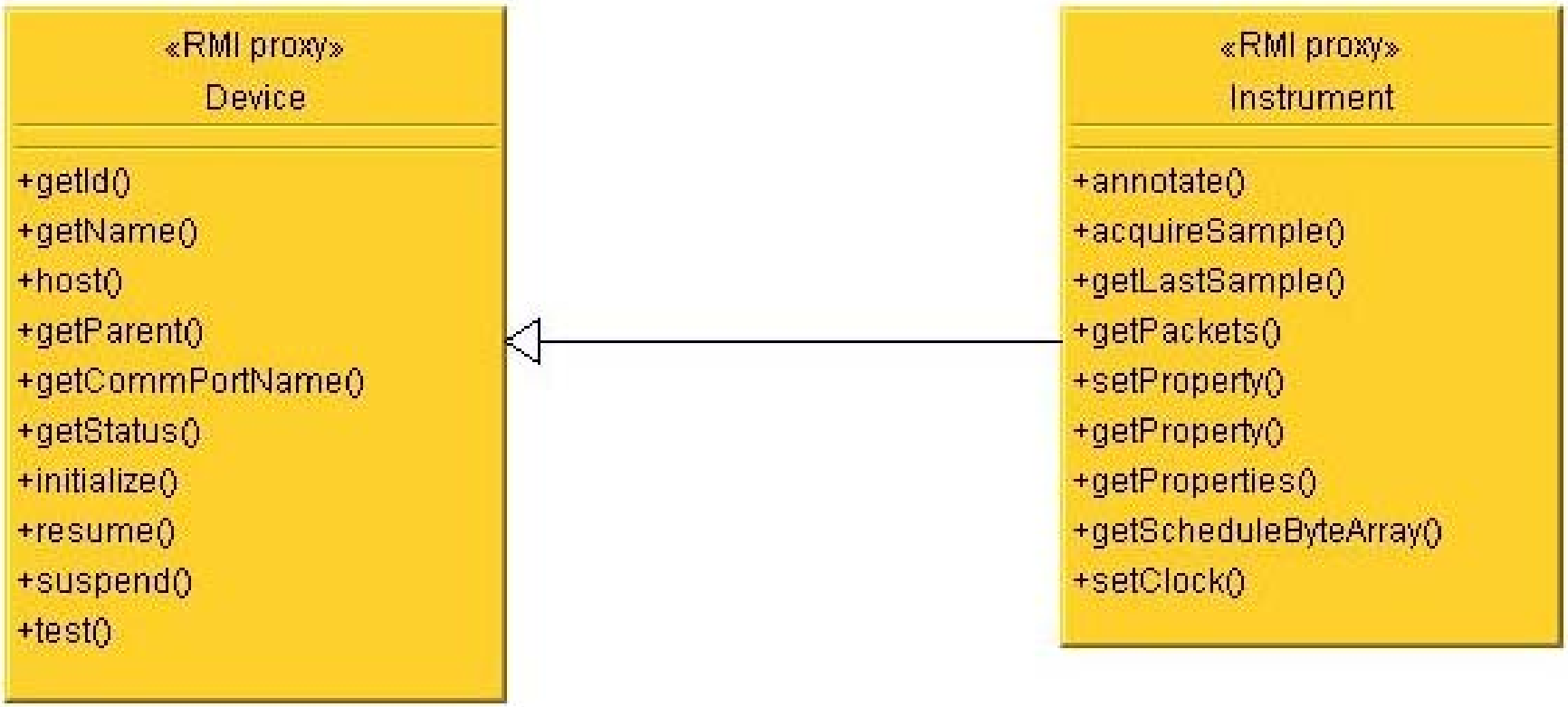
Deployment diagram

RMI interfaces: *Node*

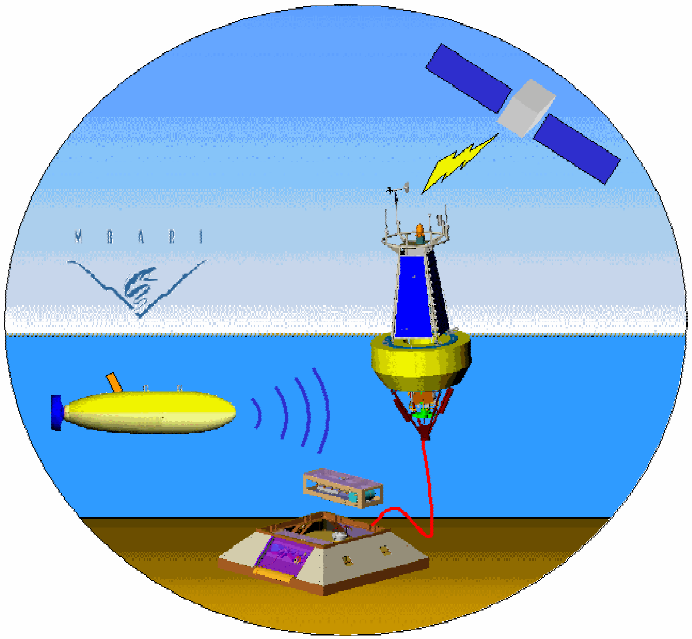
MBARI

RMI interfaces: *Device*

RMI interfaces: *Instrument*



Service discovery



Tom O'Reilly

M B A R I

Primary requirement sources

- *SIAM Behaviors Required by SSDS:*
  - Requirements 19, 22
- *MOOS Deployed Subsystem Software Requirements Specification*
  - Requirements 3.2.11, 3.2.14

MOOS network is dynamic…

- Nodes come and go from network due to power management
- Wireless links are intermittent
  - Weather, sea-state, satellite availability…
- Node and network failures

Need for “service discovery”

- Some applications need to “know” what services are available on network at any given time:
  - User interface which displays all nodes
  - Application to retrieve data from all nodes
  - Event response applications; look for instrument of particular type

Need for “service discovery”

- System should be robust in light of link dropouts, node shutdowns
- Need easy ways to add/configure new nodes
  - Avoid extensive manual configuration

Service discovery implementations

- Several emerging technologies:
  - Jini (Sun Microsystems)
  - UPnP (*Universal Plug-and-Play;* MicroSoft*)*
  - Zeroconf (includes Apple *Rendezvous*)
- Running Jini with SideARM's J9 environment is problematic
- Both Jini and UPnP development seem to have stagnated in industry

Zeroconf

- Working group of Internet Engineering Task Force (http://www.zeroconf.org)
- Based on existing *DNS* (*Domain Name Service*) protocol
- Provides service discovery, among other features
- Much simpler than Jini
- Several *Zeroconf* implementations, including Apple *Rendezvous*, Strangeberry's *JRendezvous*

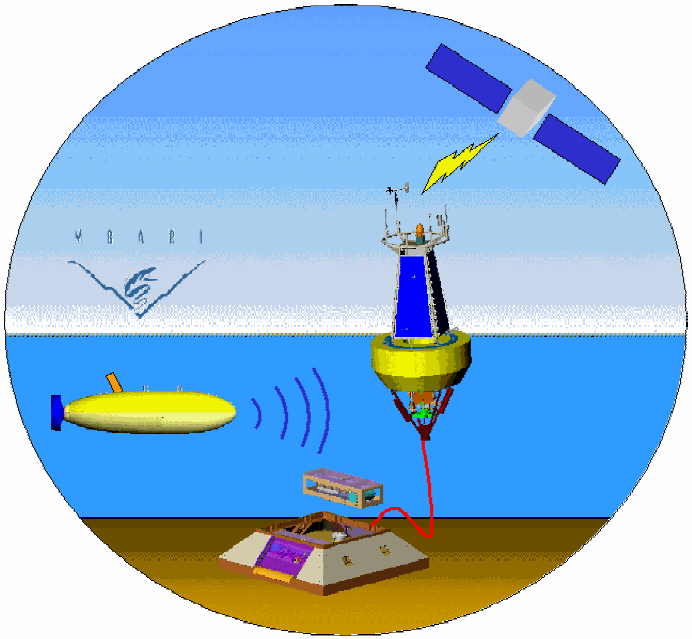
Zeroconf

- Client broadcasts request for service of particular type and domain
- Eligible services respond, providing list of additional attributes (e.g. service *time-to-live*)
  - Client can then choose service based on attributes
- Client can also be asynchronously notified of new services and service removal

Zeroconf

- Currently evaluating *Zeroconf* for SIAM
  - *NodeService* now implements Zeroconf service interface
  - Node GUI displays all nodes currently on network

Sample Acquisition and Logging

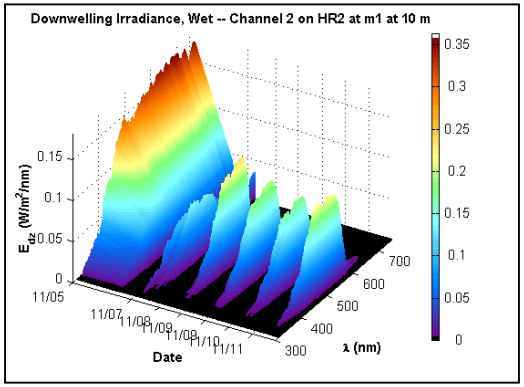


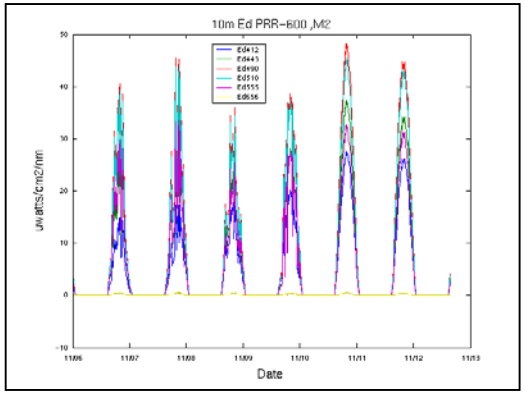
Kent Headley

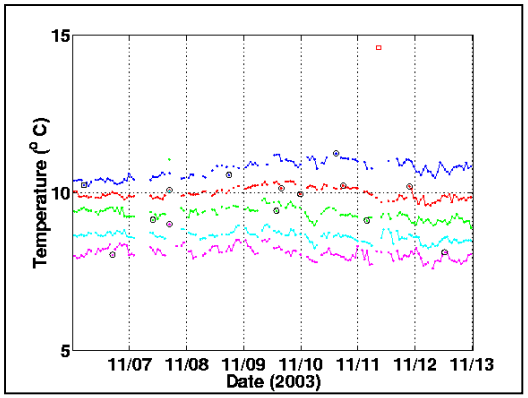
M B A R I

Sample acquisition and logging

- Sample acquisition
  - Service properties
  - Scheduling
    - Scheduler features
    - Representation of schedules
    - Managing schedules
- Data logging







Primary requirement source(s)

- *SIAM User Stories:*
  - 5,6,8,14,26-28

Service properties

- Java properties used to configure instrument services
- Some properties are common to all services:

Sampling schedule	Prompt string
Power policy	Sample terminator
Timeouts	Retries

- Instrument-specific properties may also be defined by software developer

Scheduler features

- Schedule types
  - Relative schedules
    - Periodic execution
  - Absolute
    - Execute at fixed set of times
    - Can be used to implement “day mask”
- Bells and whistles
  - Time zone
    - Awareness of site-local time
  - Execute fixed number of cycles
  - Scheduler API for programmatic schedule management
  - Objects may have multiple schedules

Scheduler internals

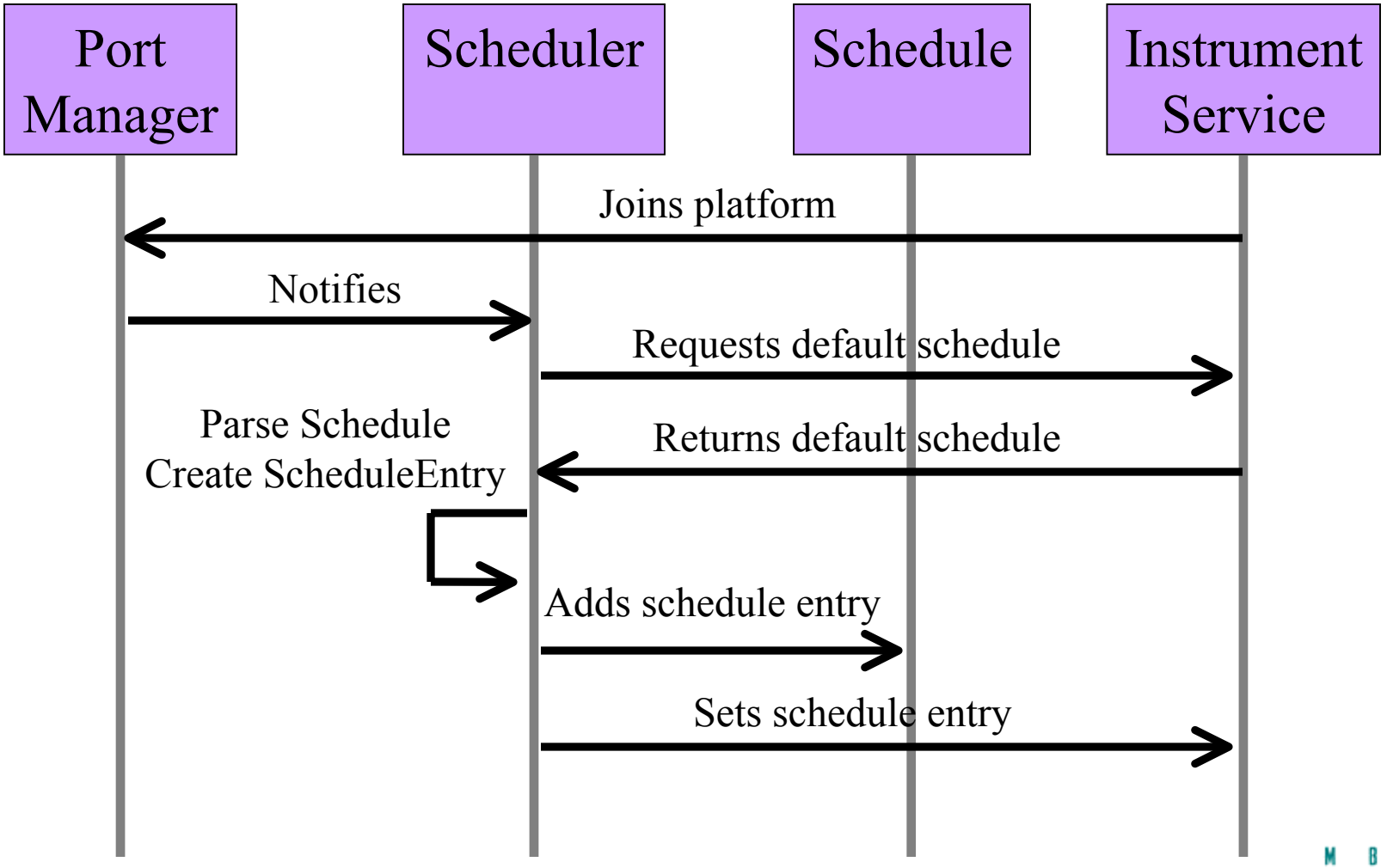
Representation of schedules

- Powerful, compact syntax derived from Unix cron
- Schedules represented as ASCII strings:

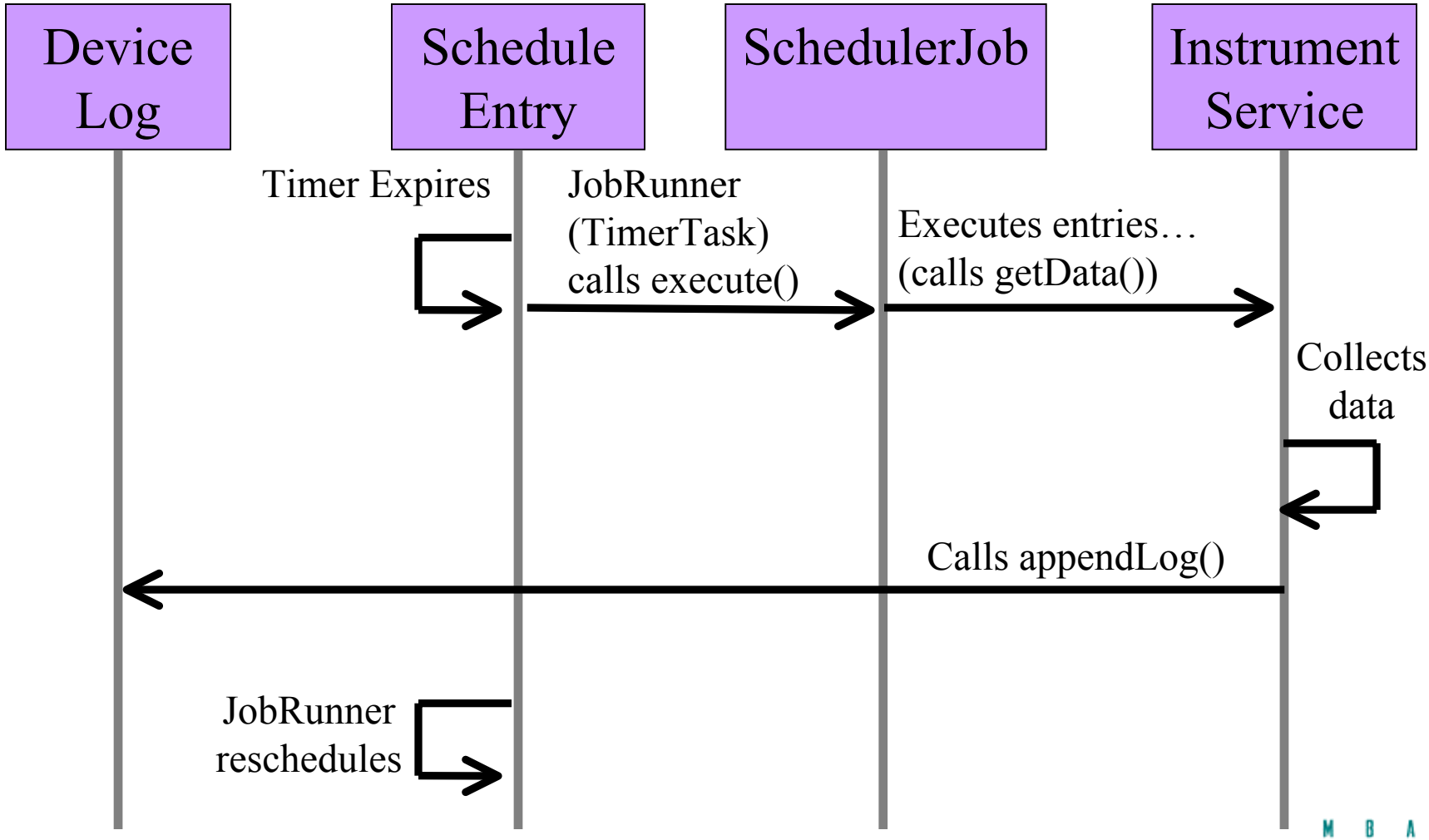
```
# Run every 10 seconds between 50 past and 10 past the minute
r 50-10/10 * * * * * * * GMT *

# Run at 0,15,30,45 past the hour, every other hour between midnight
# and 6 a.m. and between noon and 6 p.m. (GMT-8 time zone), mon-fri
# Stop after 300 cycles
a 0 0,15,30,45 0-6,12-18/2 * mon-fri * * * GMT-8 300
```

- Internally represented as long integers (relative) or as binary arrays (absolute)
- Each schedule runs in its own TimerTask thread (modified for compatibility with power manager)

Scheduler behavior



Scheduler behavior



Managing schedules

Setting and changing schedules

- Instrument services automatically scheduled when joining node
  - Scheduler is notified when new service joins node
  - Scheduler requests default sampling schedule string
  - Service creates and starts schedule
- Schedules may be modified/viewed via command line utilities

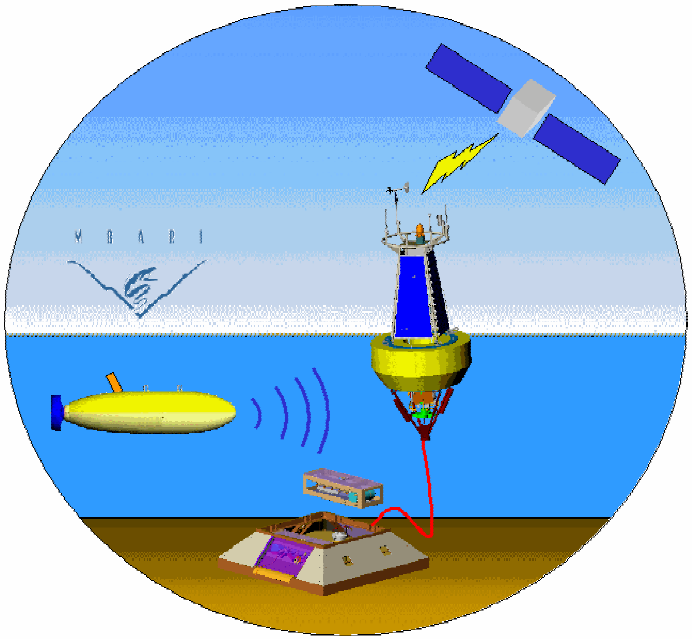
removeSchedule	suspendSchedule
addSchedule	resumeSchedule
showSchedule	synch

- Objects may create and manage schedules via scheduler API
- Open Issues
  - Alternative schedule utilities (better ways of indexing multiple schedules)
  - GUI scheduling utility

Data logging

- NMC logging code extended for SIAM
- Each instrument service has its own log
- NodeService also has log containing message packets
  - Latest instrument list
- Logs structured for rapid random access
  - Data file contains serialized SensorDataPacket objects
  - Index file maps multiple keys (time, sequence number) to offset in data file
- Ext2 file system used on compact flash (non-journaling; ext3?)

SIAM Portal



Tom O'Reilly

M B A R I

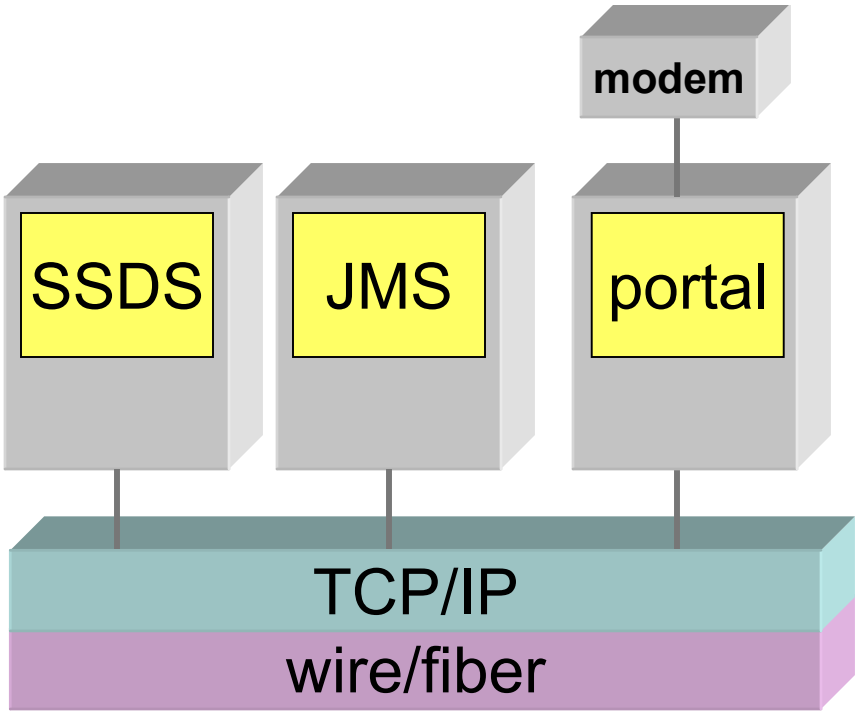
Primary requirement sources

- *SIAM Behaviors Required by SSDS:*
  - Numerous requirements

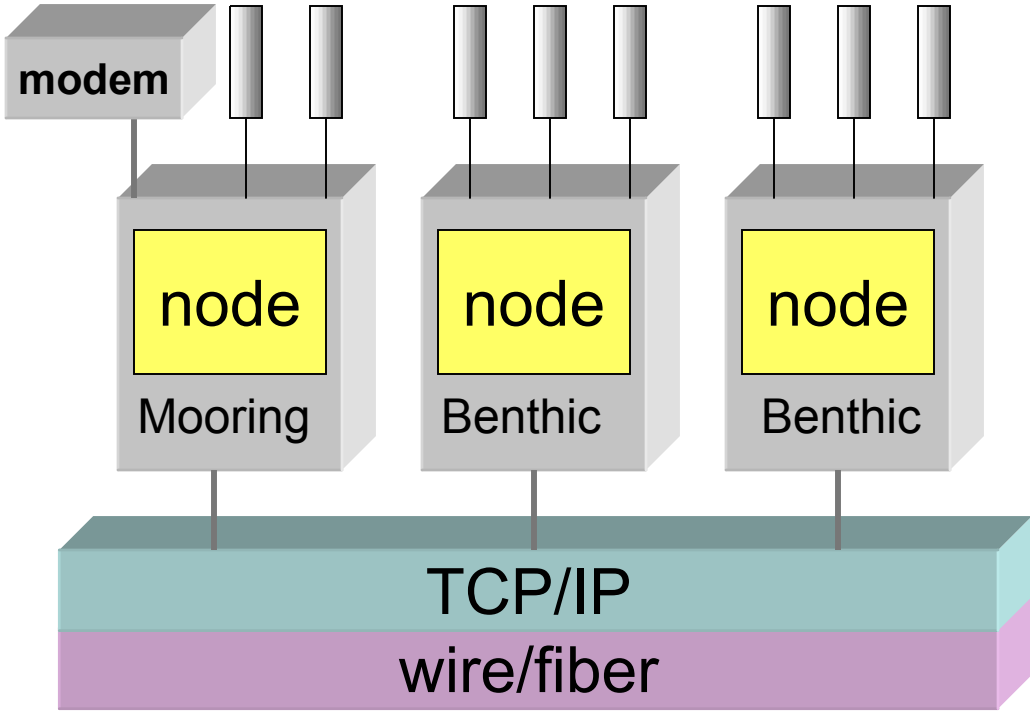
- *MOOS Operations Subsystem Software Requirements Specification*
  - Requirements 2.2.2, 2.2.3, 2.2.4, 2.2.5

Portal functions

- Retrieve data and metadata packets from node and its instruments when wireless link is available
  - Store retrieved packets in log files
  - Distribute packets to Shore-Side Data System
- Retrieved packets consist of instrument data, metadata, and “messages” (e.g. when instrument added to node)



Shore network



Moored network

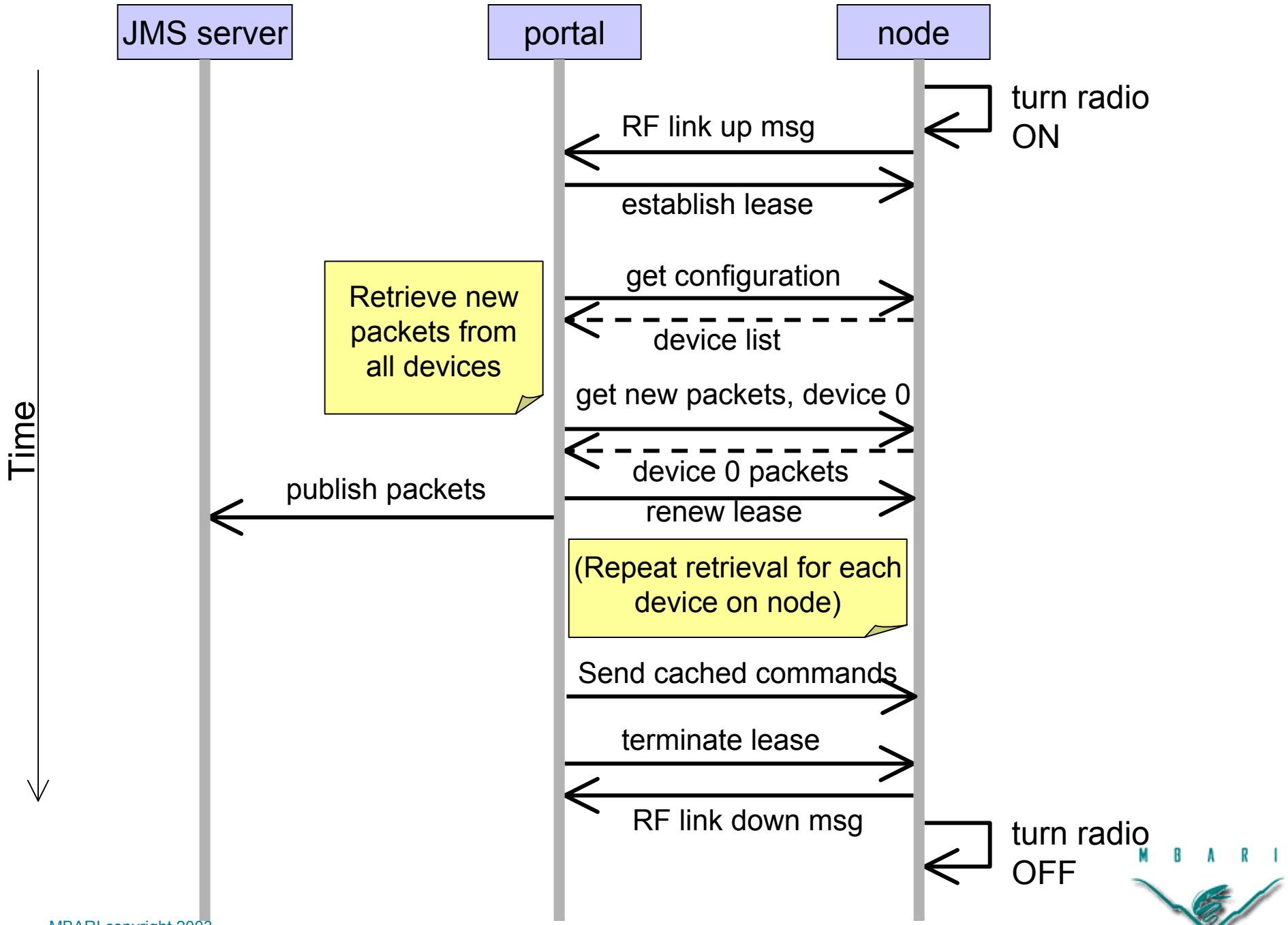
Node methods called by portal

```
public interface Node extends Remote {

    Port[] getPorts();

    DevicePacketSet getDevicePackets(long deviceID,
                                     long startTime,
                                     long endTime);
```

Portal retrieves list of ports and devices each time link is reconnected

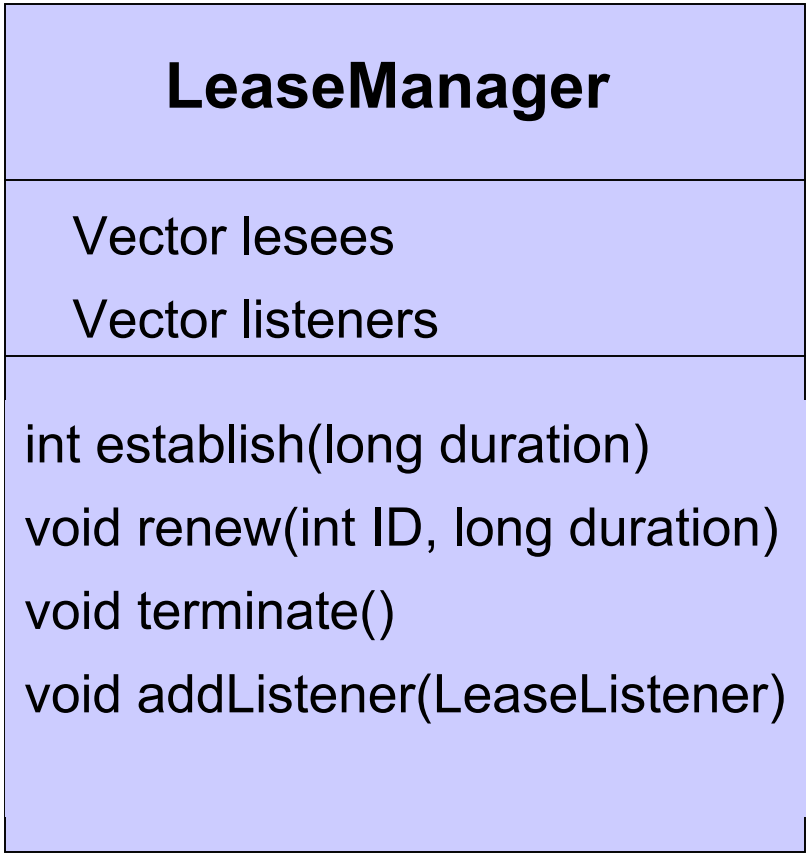
For each device, portal tries to retrieve all packets created since previous download

Note that node itself is a device; so portal retrieves node messages as well as instrument packets



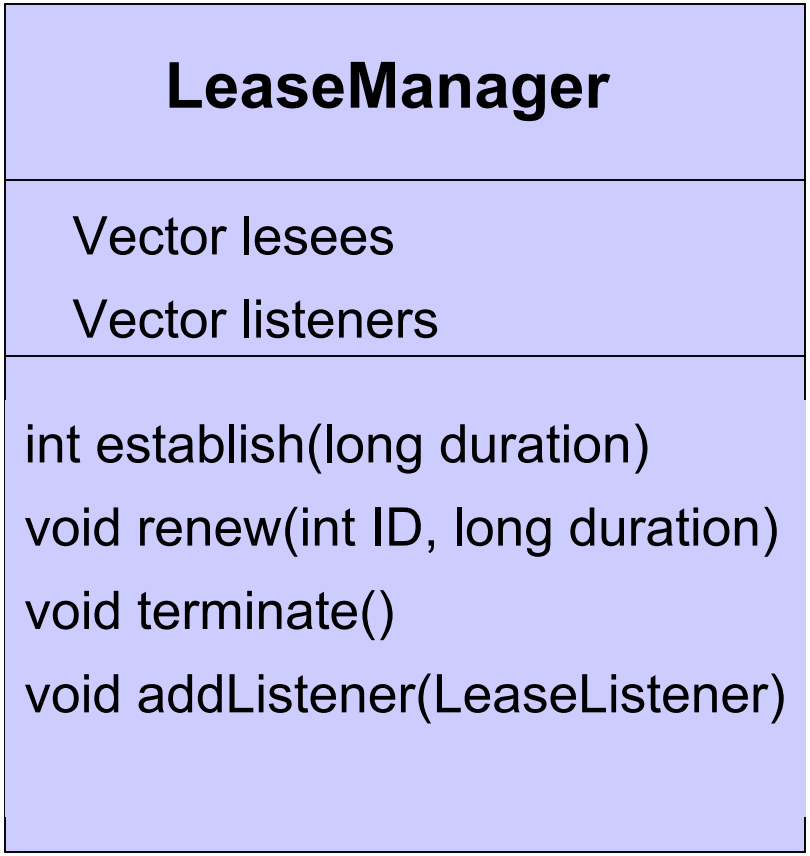
Lease manager

- Arbitrates use of limited resource by multiple entities
- Each entity (*lessee*) can use the resource for a specified time period (*lease duration*)
- Lessees are responsible for renewing their leases with lease manager
- As leases are established, renewed, terminated, or expire, manager notifies *lease listeners*



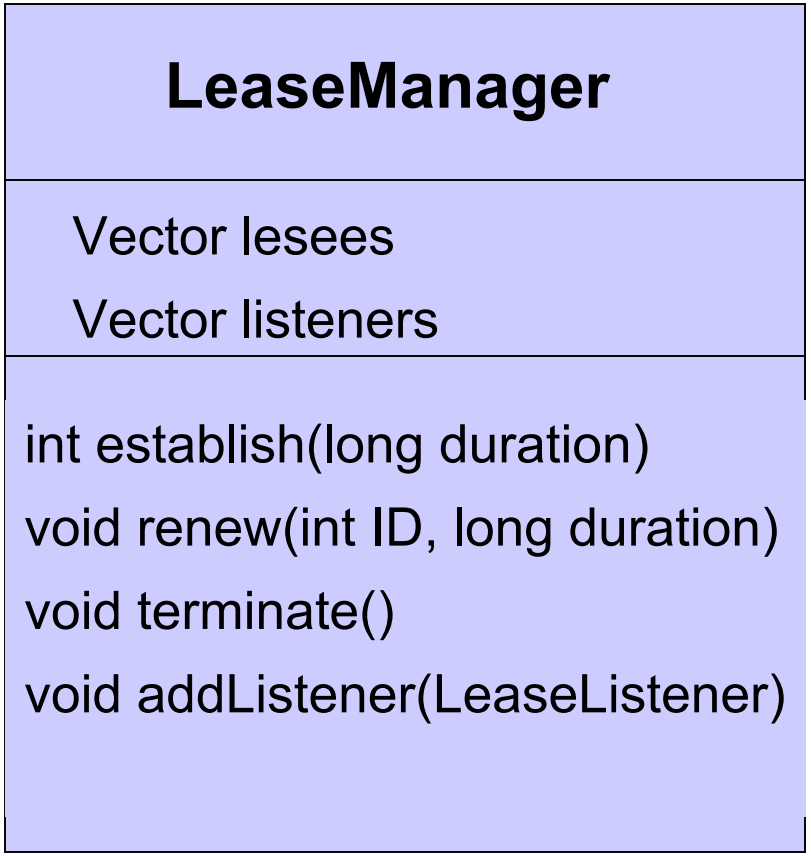
Lease manager

- Lessee is added to list each time *establish()* is called, timer is associated with each lessee
- Lessee's timer is reset when *renew()* is invoked
- Lessee is removed from list when lessee calls *terminate(),* or when the lessee's timer expires



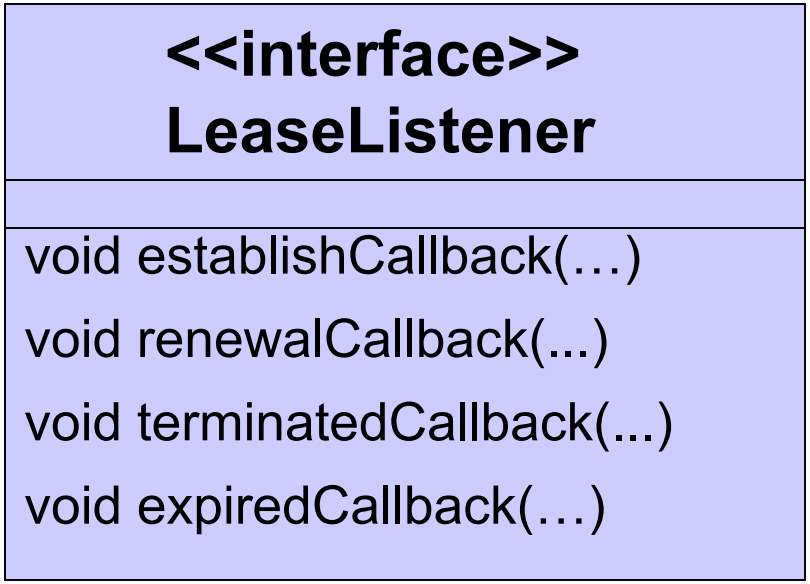
Lease manager

- Lease manager aggregates one or multiple *lease listeners*, which are added with the addListener() method



Lease listener interface

- *Lease listeners* are responsible for the management of the resource being leased.
- Listeners are notified by lease manager via callback when a lease is established, renewed, terminated, or expired; the listener callback takes appropriate management action

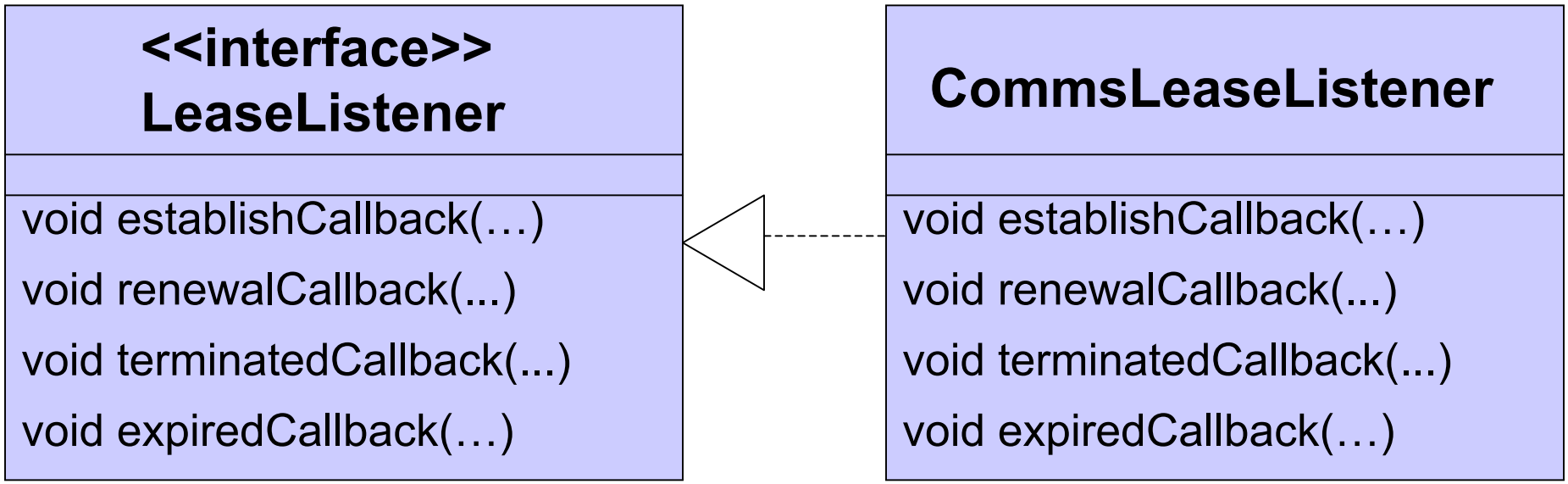


Lessees of RF power

- Onboard *CommsScheduler*
  - Requests that RF be switched on at regular intervals (to give offboard clients chance to access node)
- Shore-based portal
  - Requests that RF link remain powered during data transfers

CommsLeaseListener

- Manages RF power



CommsLeaseListener

```
public void establishCallback(int lesseeID, long period,
                              LeaseManager mgr) {
  // If this is first lessee, turn on power, wait
  // a few seconds, then initiate "link is up" message
  // to portal.
}

public void expiredCallback(int lesseeID,
                            LeaseManager mgr) {
  // If there are no more lessees, initiate "link
  // going down" message to portal, wait a bit, then
  // shut off radio power.
}
```

CommsLeaseListener

```
public void terminatedCallback(int lesseeID,
                               LeaseManager mgr) {
  // If there are no more lessees, initiate "link
  // going down" message to portal, wait a bit, then
  // shut off radio power.
  // (SAME AS expiredCallback…)
}

public void renewalCallback(int lesseeID, long period,
                            LeaseManager mgr) {
  // A lease has been renewed; don't need to do
  // anything.
}
```

Portal command-line options

-since *timestamp*

Retrieve all packets created since *timestamp*.

Valid timestamp formats:

mm/dd/yyyy	**Get all packets created since mm/dd/yyyy**
mm/dd/yyyyThh:mm	**Get all packets created since mm/dd/yyyy, hh:mm**
now	**Get all packets created since portal was started**

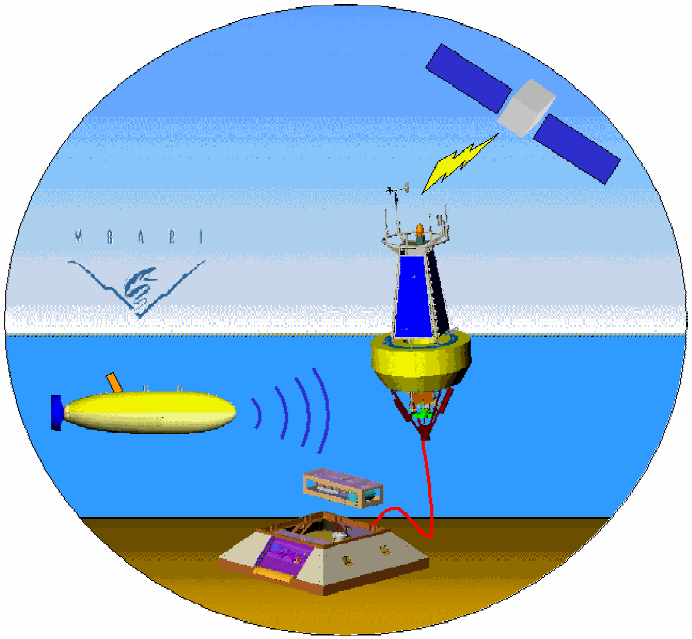
Possible future portal work

- Support for multiple wireless channels to node
  - E.g. low-bandwidth satellite plus high-bandwidth short-haul radio

Possible future portal work

- Utility to store device commands in portal
  - Portal will forward commands to node when link is available
- Regulate user interface bandwidth
  - Allow multiple users to utilize low-bandwidth link

Interfaces to Shore-Side Data System



Kent Headley and Dan Davis

M B A R I

SIAM metadata interface

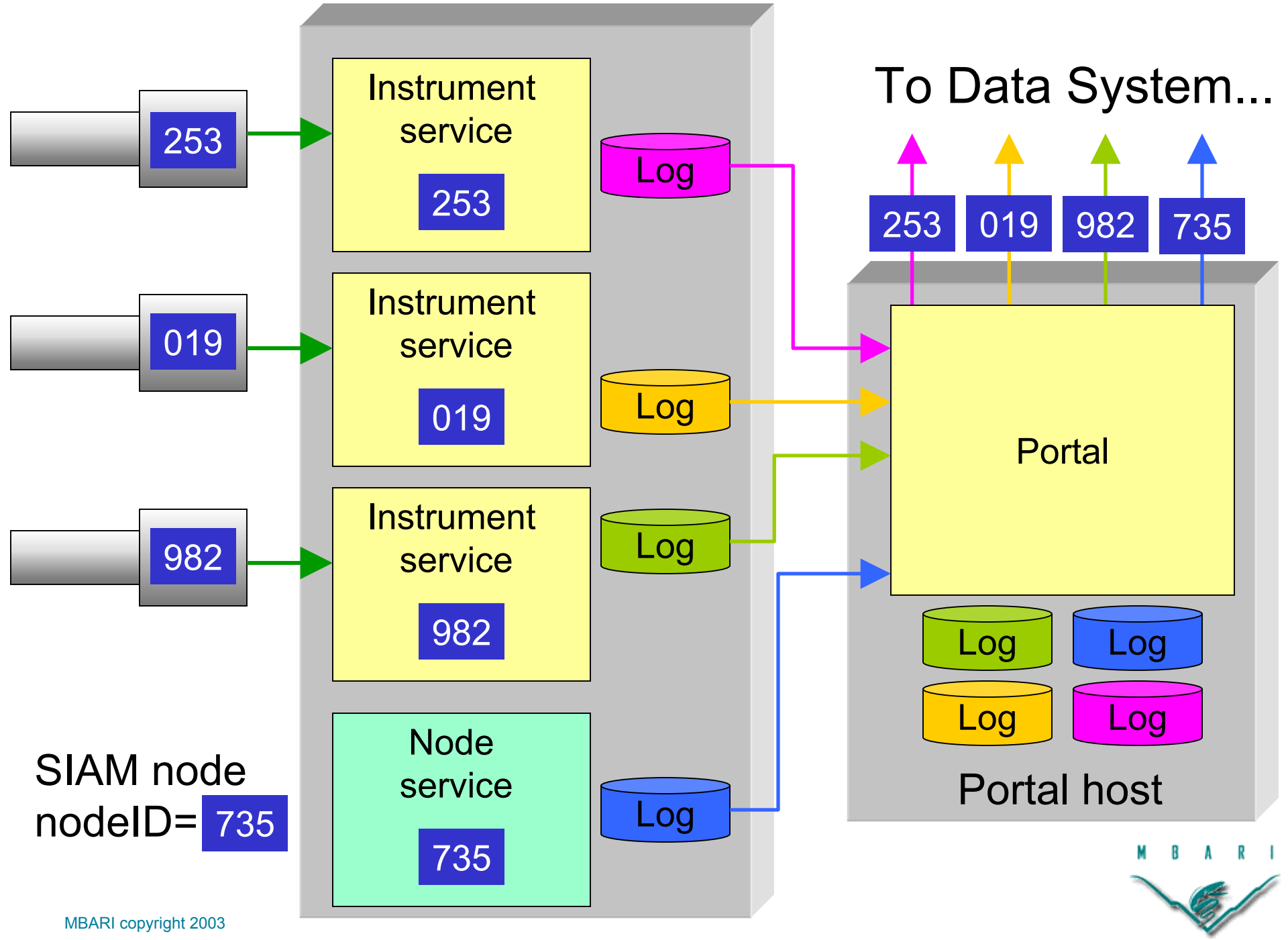
- SIAM's internal representation of metadata
- Metadata generation and delivery
- Open issues

Primary requirement source(s)

- *SIAM User Stories*
- *SIAM Behaviors Required by SSDS*

Postulate

- All SIAM communication of deployed device state occurs through device data streams
- Device data streams consist of a sequence of time stamped packets of different types



DevicePacket types

DevicePacket

- Base class for other packets
- Defines generic packet attributes; time stamp, sequence number, metadata reference sequence number

DevicePacket types

SensorDataPacket

- Sensor data

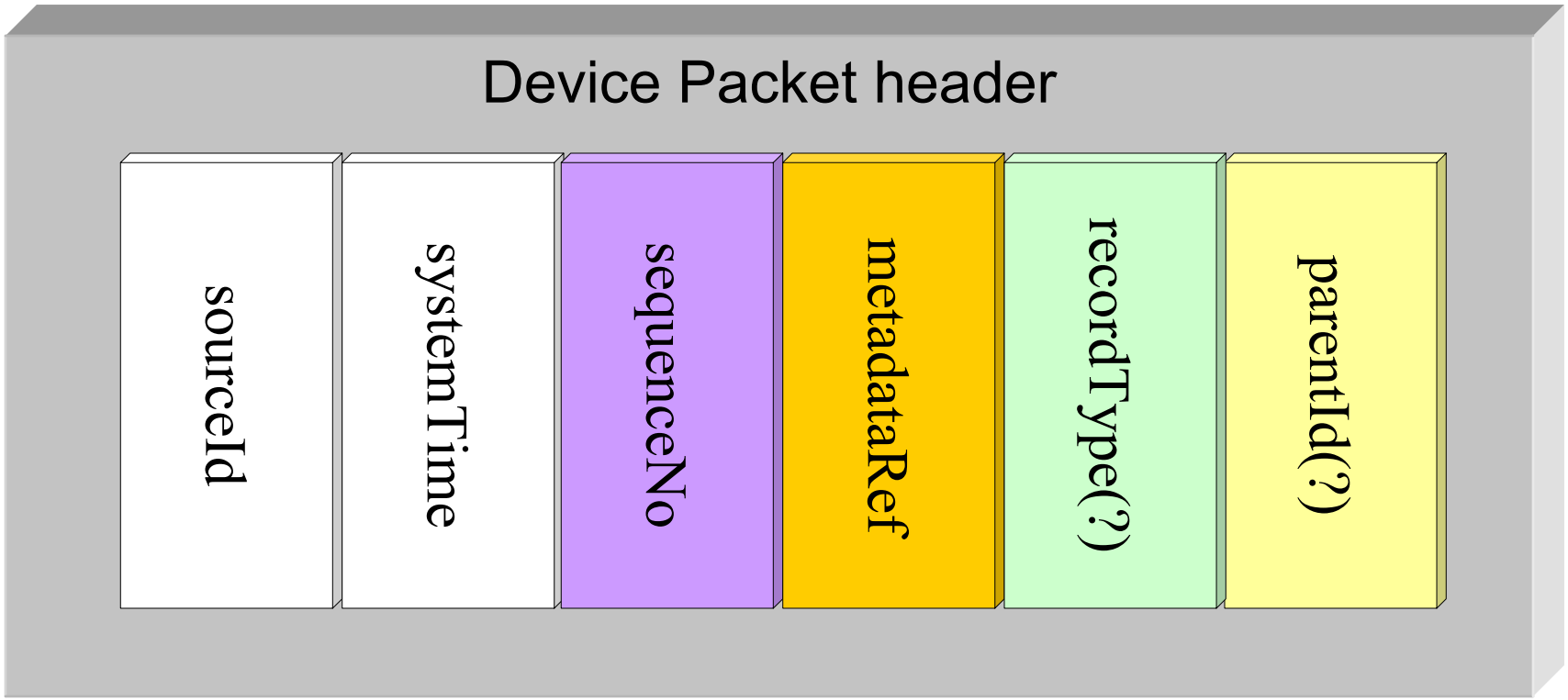
MetadataPacket

- Data about data; includes XML, service properties, instrument status

MessagePacket

- ASCII messages, recording “events”

Device packet header structure



Metadata packet generation

- Metadata packets are generated and logged under the following conditions:
  - Instrument service starts
  - User requests metadata packet
  - User sets service properties
  - User runs *conPort* utility
  - Service detects relevant state change

Metadata packet contents

- Header
- Payload, containing one or more of the following:
  - Puck XML document
  - Java properties file
  - Current values of service Java properties
  - Current “status message” string from device itself

Message packet generation

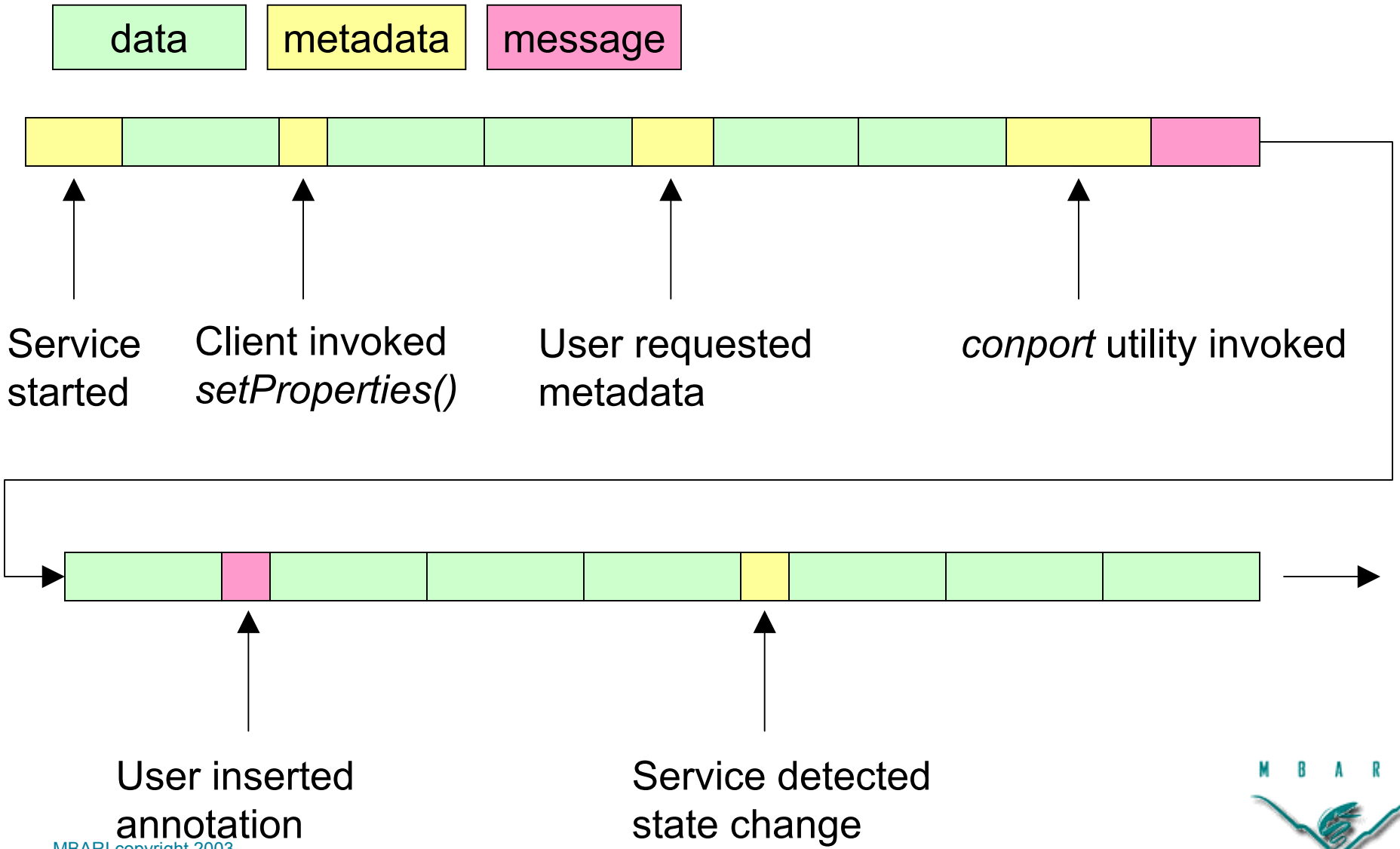
- Message packet written to instrument log when:
  - User runs *annotateService* utility
  - User runs *conPort* utility
  - When “serious” exceptions occur
- (Message)Packet written to node log when:
  - Node reboots
  - Instrument service added
  - Instrument service removed

Message packet contents

- Arbitrary human-entered text, e.g.
  - **`Removed barnacles from sensor head`**
- Error message text, e.g.
  - **`Metsys timed out reading sample`**
- Informational messages, e.g.

```
Added service for device 1215 to port /dev/ttySX0
ch=/dev/ttySX0 isiID=1215
ch=/dev/ttySX1 isiID=4297
ch=/dev/ttySX6 isiID=3002
ch=/dev/ttySX15 isiID=1955
```

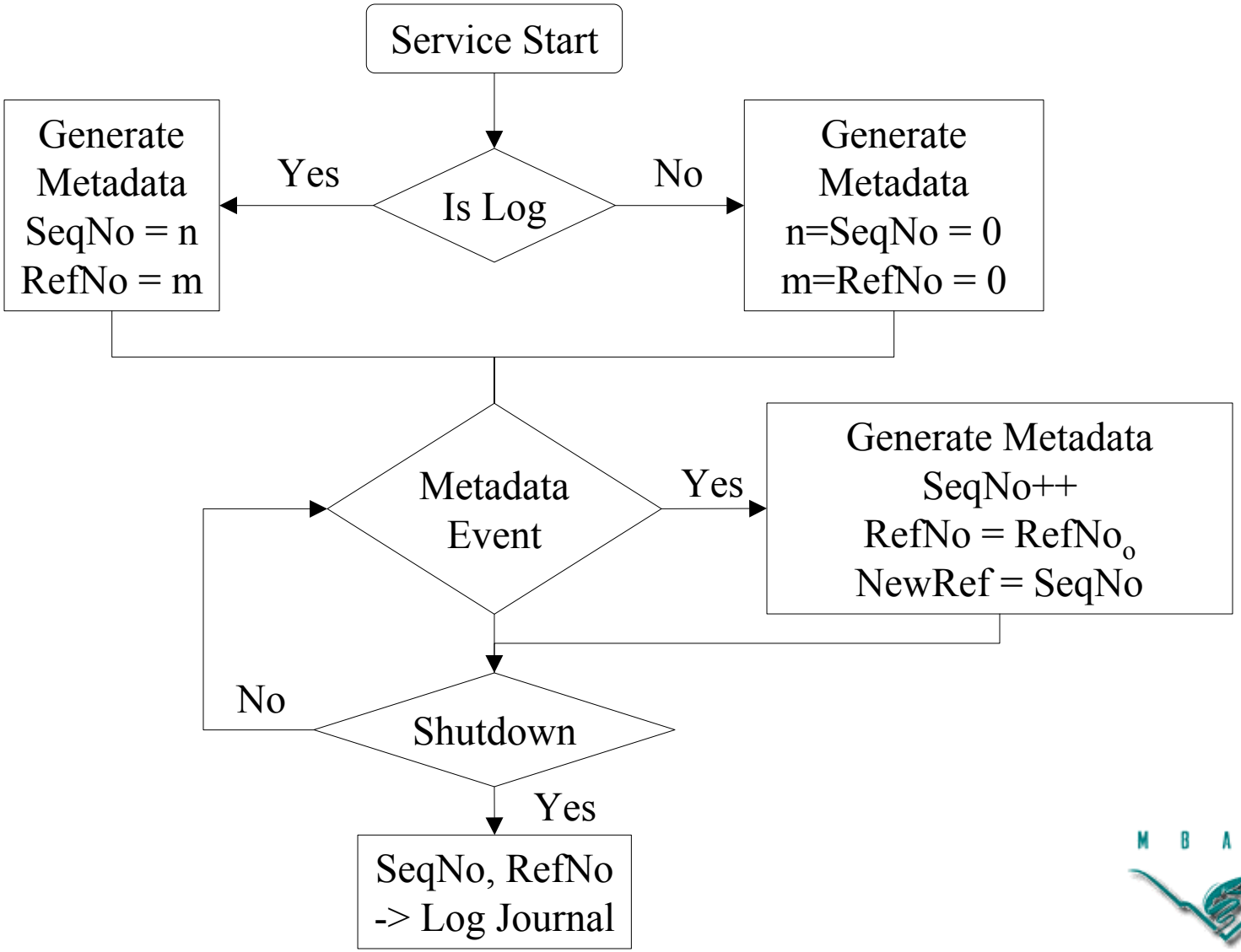
Sample data stream



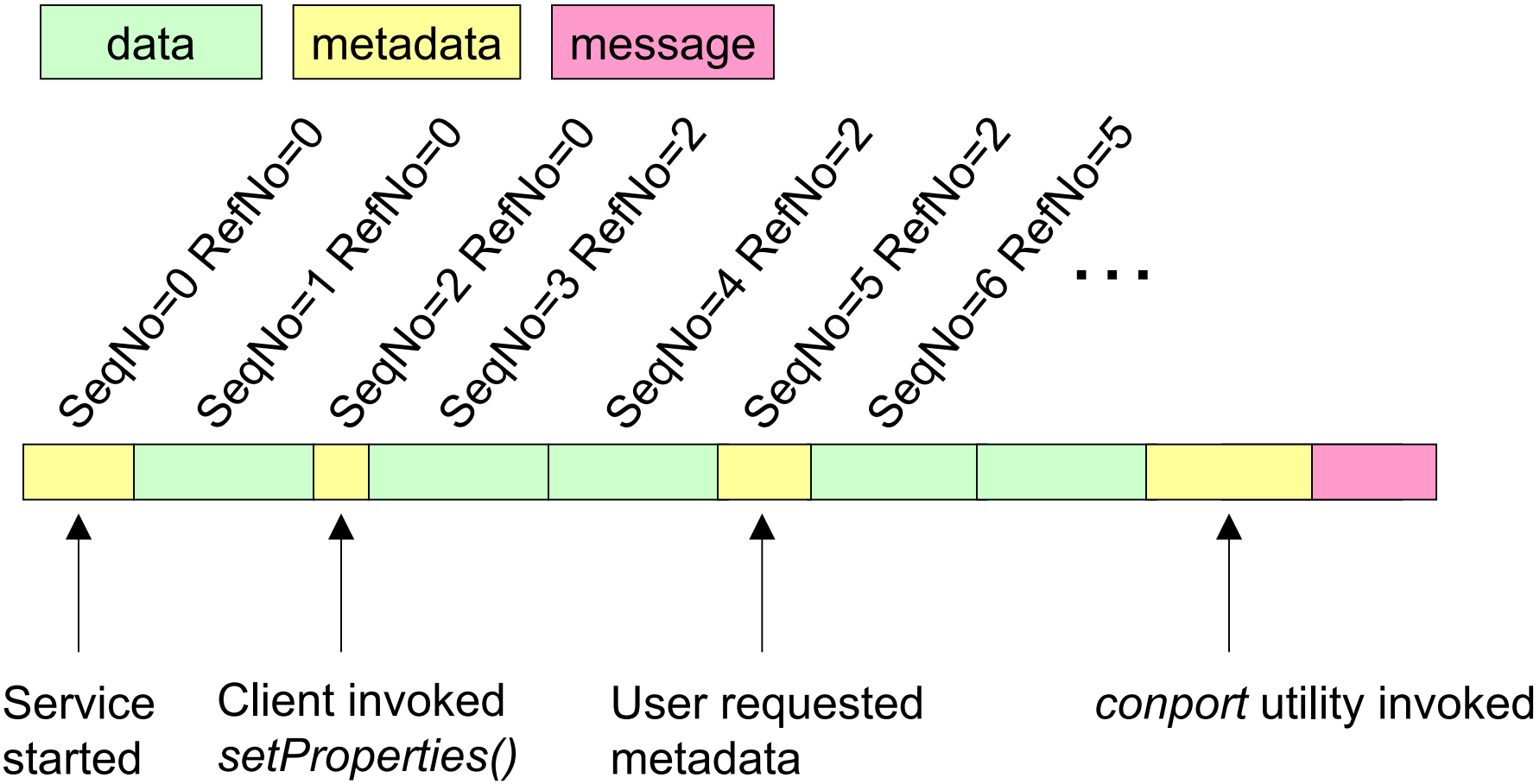
Metadata packet reference numbers

- Changes to metadata state may be tracked using reference number fields in packet header
- Missing data or metadata may be requested from portal

Metadata packet reference numbers



Sample data stream



SIAM metadata interface

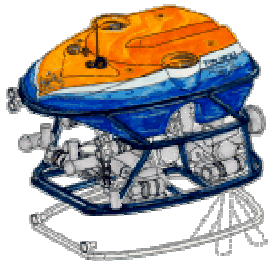
Open issues

- When to include parentID and recordType
  - Every packet/only when changes occur
  - Impacts bandwidth
  - Impacts data system processing
- Using XML metadata in deployed subsystem
- XML schema design: what to deploy
- Co-developing automated tests with SSDS to verify interface to SSDS

MOOS goals

- Configurable, re-deployable, instruments and platforms (using ships and ROVs)
- Suitable for deep ocean, or coastal studies, with low power, and low bandwidth, intermittent communication links to shore
- Smart nodes, with some on-board data processing to facilitate autonomous event detection, and response

Metadata system requirements



Metadata needed to:

- interpret scientific data from sensors
- describe sensors, instruments, platforms, and communication links in a deployment
- system infrastructure data support
- support the real time operation of the system, including monitoring and diagnostics

Metadata general requirements

- Must readily define, capture, access and represent system data of varying types
- Ability to convert between a wide variety of data formats
- Flexible and extensible
- Easy to display, use, and understand by a wide variety of users

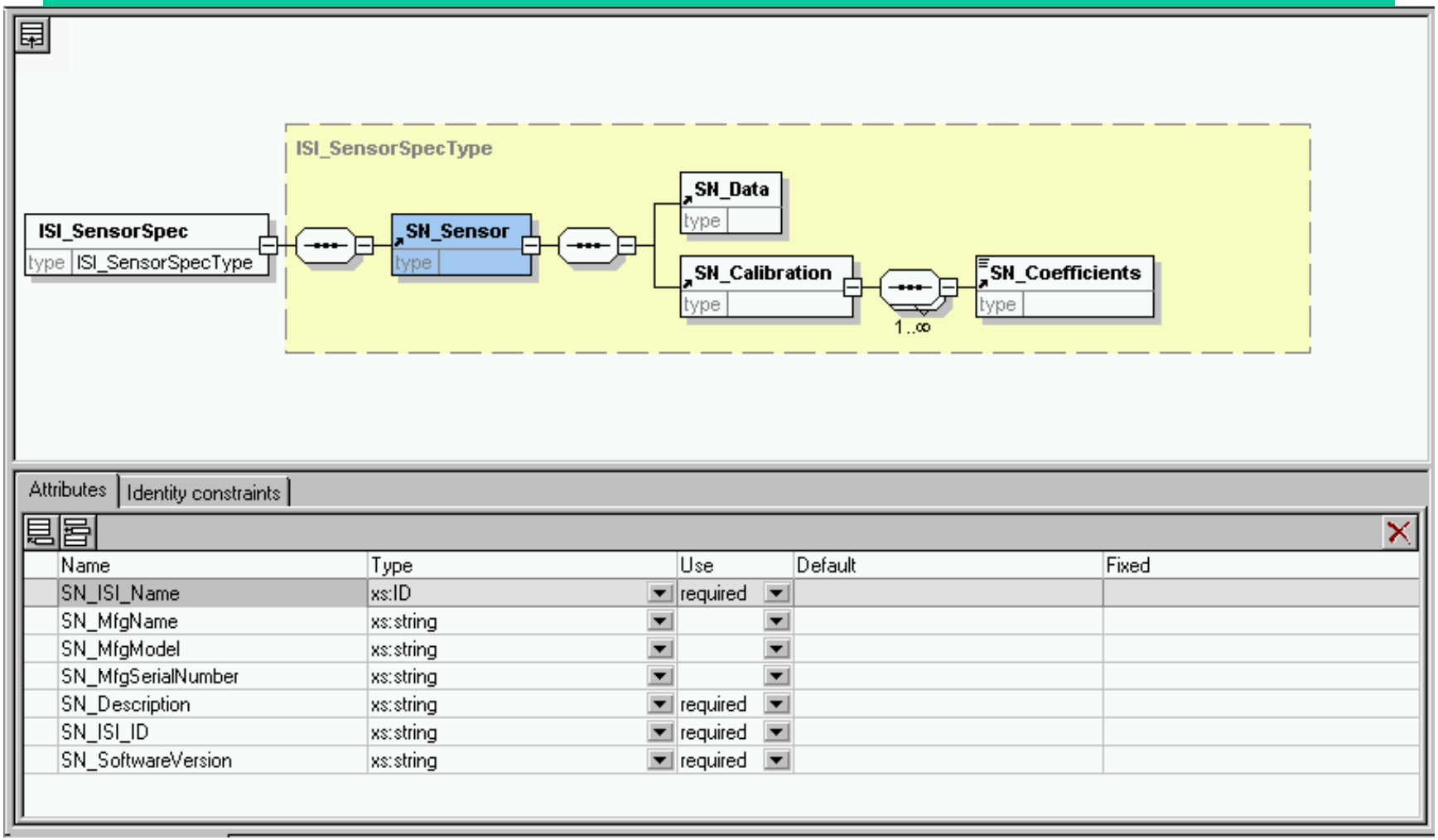
Technical approach to metadata

- XML best match for every requirement except ease of use
- Use XML driven GUI technology to create forms to create, display, access, and use metadata for configuation and in the SSDS
- Users should never have to directly read XML
- Bind XML metadata early to each device through its puck

Steps to using XML for metadata

- Designer creates an XML schema for a class of devices: platforms, instruments or sensors

Metadata Schema Design

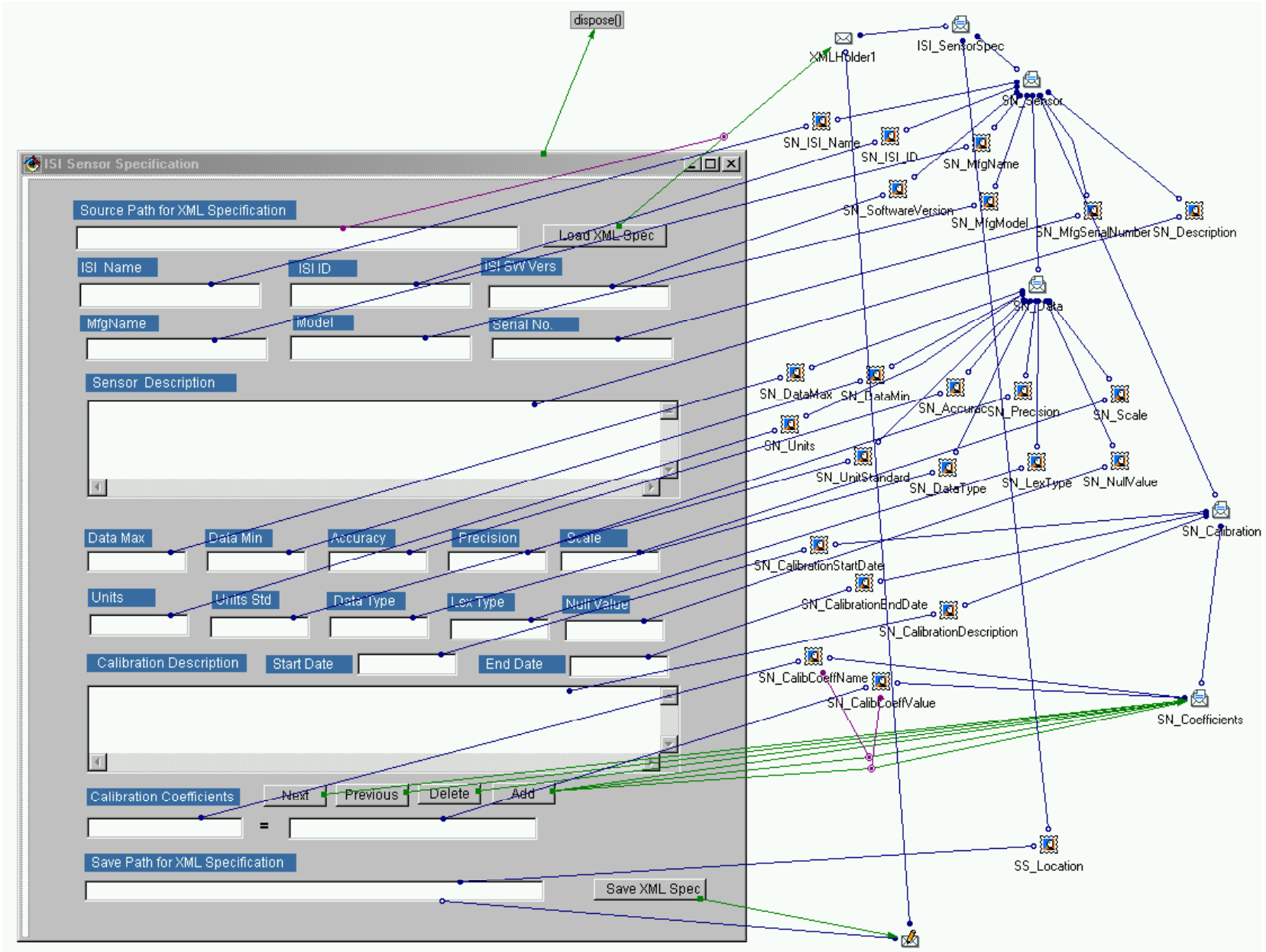


Attributes | Identity constraints

Name	Type	Use	Default	Fixed
SN_ISI_Name	xs:ID	required		
SN_MfgName	xs:string			
SN_MfgModel	xs:string			
SN_MfgSerialNumber	xs:string			
SN_Description	xs:string	required		
SN_ISI_ID	xs:string	required		
SN_SoftwareVersion	xs:string	required		

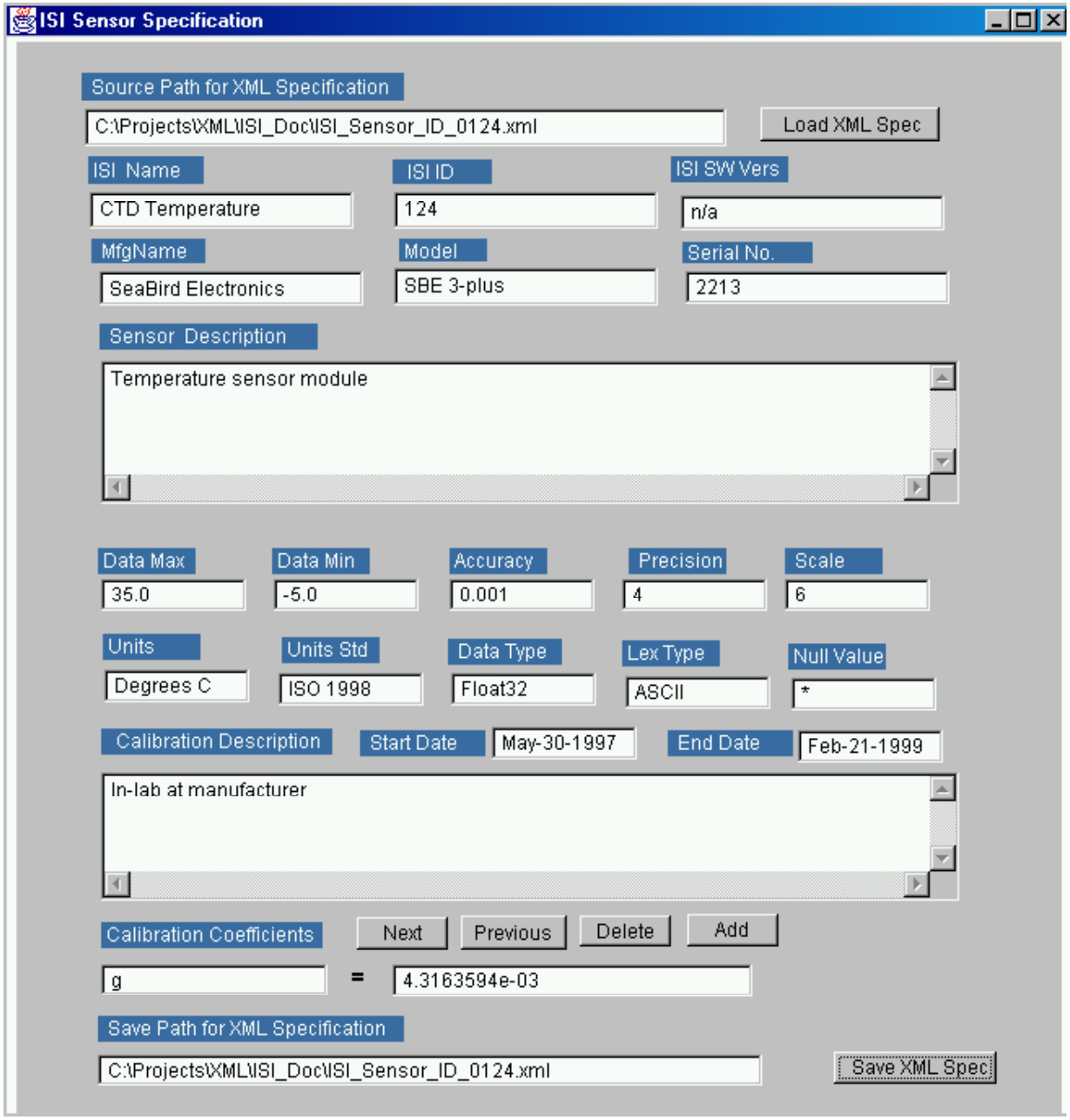
Metadata User Form Design

- User interface designer uses schema to build a form for creation, display, access, of metadata instances
- There may be different forms for different users (e.g. scientific, system, and operational) to create, and display metadata of interest



Instrument configuration

- Metadata forms are used during device configuration to create metadata that is entered into device puck
- Similarly metadata forms are used during configuration of other system elements, such as platforms, and even communication links. This metadata is maintained in system nodes.

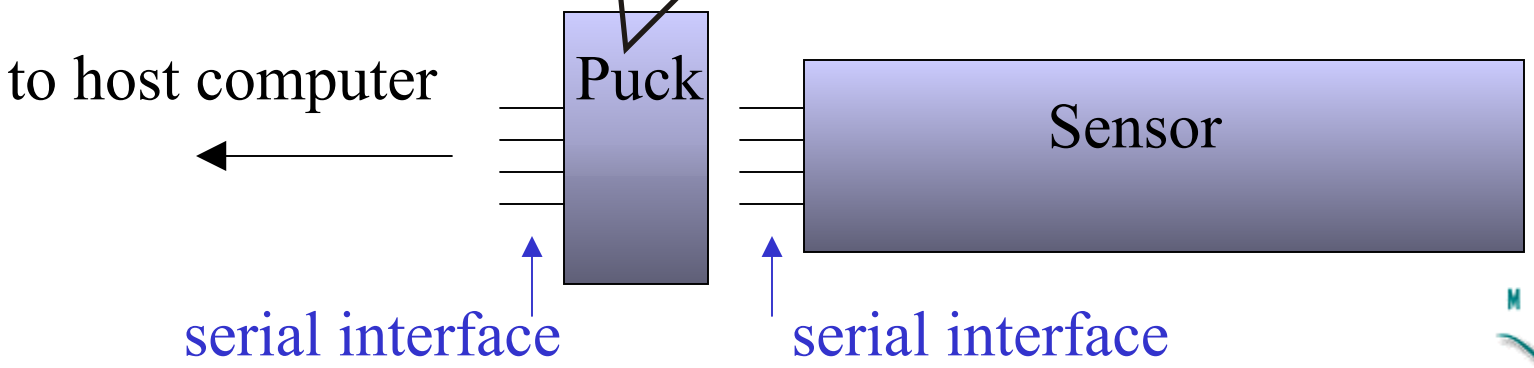


Actual metadata instance object

```
<?xml version="1.0" encoding="UTF-8" ?>
- <ISI_SensorSpec SS_Location="C:\Projects\XML\ISI_Doc\ISI_Sensor_ID_0124.xml">
  - <SN_Sensor SN_Description="Temperature sensor module" SN_ISI_ID="124"
      SN_ISI_Name="CTD Temperature" SN_MfgModel="SBE 3-plus" SN_MfgName="SeaBird
      Electronics" SN_MfgSerialNumber="2213" SN_SoftwareVersion="n/a">
      <SN_Data SN_Accuracy="0.001" SN_DataMax="35.0" SN_DataMin="-5.0"
        SN_DataType="Float32" SN_LexType="ASCII" SN_NullValue="*" SN_Precision="4"
        SN_Scale="6" SN_UnitStandard="ISO 1998" SN_Units="Degrees C" />
    - <SN_Calibration SN_CalibrationDescription="In-lab at manufacturer"
        SN_CalibrationEndDate="Feb-21-1999" SN_CalibrationStartDate="May-30-1997">
        <SN_Coefficients SN_CalibCoeffName="g" SN_CalibCoeffValue="4.3163594e-03" />
        <SN_Coefficients SN_CalibCoeffName="h" SN_CalibCoeffValue="6.41530157e+04" />
        <SN_Coefficients SN_CalibCoeffName="i" SN_CalibCoeffValue="2.27235685e-05" />
        <SN_Coefficients SN_CalibCoeffName="j" SN_CalibCoeffValue="2.1715345e-06" />
      </SN_Calibration>
    </SN_Sensor>
  </ISI_SensorSpec>
```

Metadata stored in puck interface

```
<?xml version="1.0" encoding="UTF-8" ?>
- <ISI_SensorSpec SS_Location="C:\Projects\XML\ISI_Doc\ISI_Sensor_ID_0124.xml">
  - <SN_Sensor SN_Description="Temperature sensor module" SN_ISI_ID="124"
      SN_ISI_Name="CTD Temperature" SN_MfgModel="SBE 3-plus" SN_MfgName="SeaBird
      Electronics" SN_MfgSerialNumber="2213" SN_SoftwareVersion="n/a">
      <SN_Data SN_Accuracy="0.001" SN_DataMax="35.0" SN_DataMin="-5.0"
        SN_DataType="Float32" SN_LexType="ASCII" SN_NullValue="*" SN_Precision="4"
        SN_Scale="6" SN_UnitStandard="ISO 1998" SN_Units="Degrees C" />
    - <SN_Calibration SN_CalibrationDescription="In-lab at manufacturer"
        SN_CalibrationEndDate="Feb-21-1999" SN_CalibrationStartDate="May-30-1997">
        <SN_Coefficients SN_CalibCoeffName="g" SN_CalibCoeffValue="4.3163594e-03" />
        <SN_Coefficients SN_CalibCoeffName="h" SN_CalibCoeffValue="6.41530157e+04" />
        <SN_Coefficients SN_CalibCoeffName="i" SN_CalibCoeffValue="2.27235685e-05" />
        <SN_Coefficients SN_CalibCoeffName="j" SN_CalibCoeffValue="2.1715345e-06" />
      </SN_Calibration>
    </SN_Sensor>
  </ISI_SensorSpec>
```



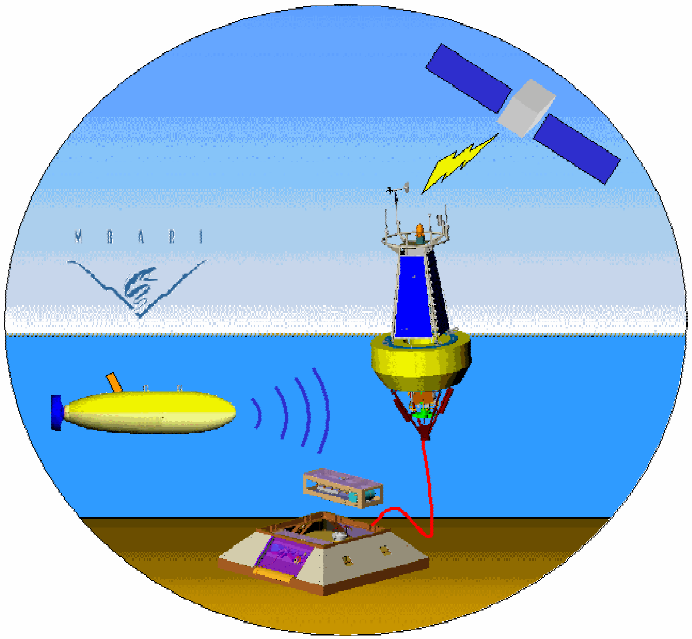
Metadata access and display

- Metadata loaded to a host node is accessed across network through ISI by a portal on shore and supplied to the SSDS with the data from the device
- The same form used for creation and configuration of the device is used to display metadata on shore

Summary

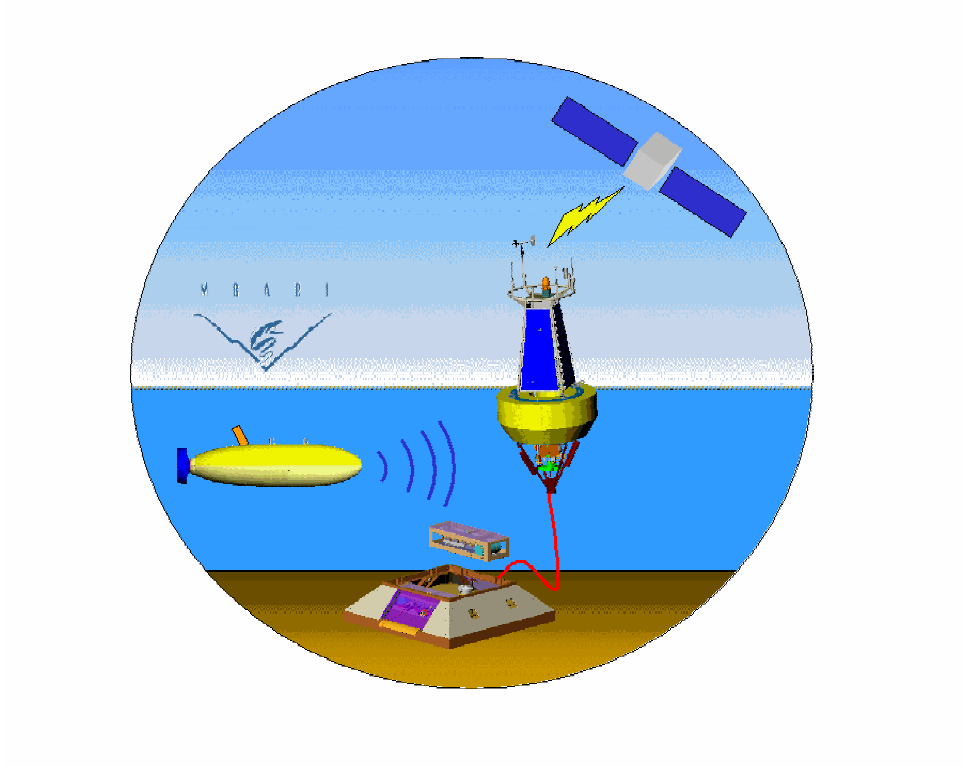
- The metadata problem for observing systems includes the problem of managing system as well as scientific metadata
- A successful approach should make it easy for users to create, access, and display metadata
- The metadata solution should be integrated into the overall design of the observing system
- Early binding of the metadata to the data source associated to it, is desirable for maintaining integrity of the data system

Break



M B A R I

Instrument Service Framework



Tom O'Reilly

Primary requirement sources

- Good design practice

Instrument services

- Supported interfaces
- Application framework
- Configuration and control
- Aids to development

Supported instrument interfaces

- Currently supported
  - RS-232
- Not currently supported:
  - RS-422/RS-485, Ethernet

Instrument service development

- Each instrument on node must have an accompanying instrument service that conforms to SIAM service interface standards
- Need to make service development process straightforward and robust as possible

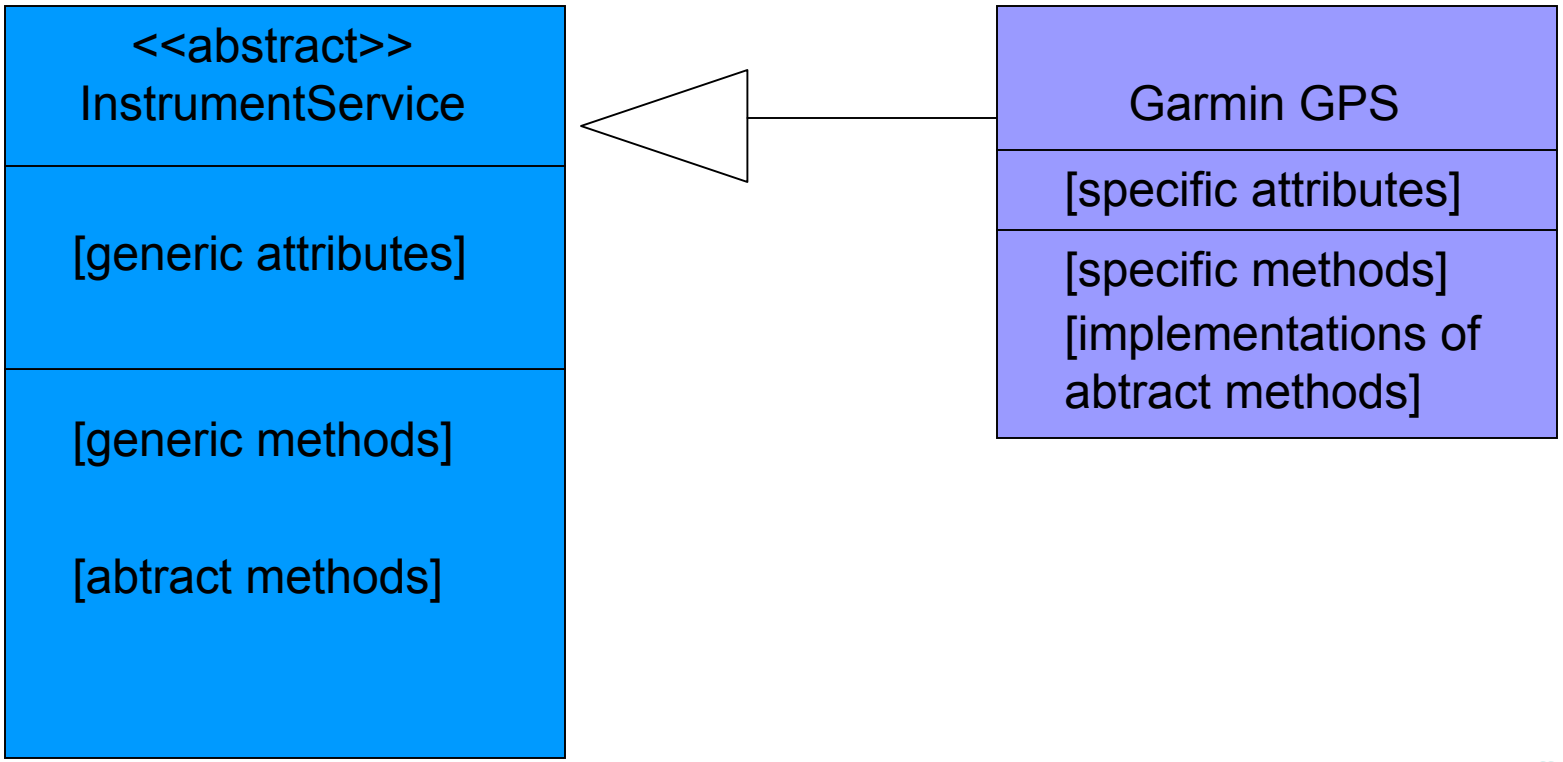
Application framework

- Application framework is a “semi-complete” application
- Consists of base classes that provide generic functionality, and “place-holders” for specific implementation

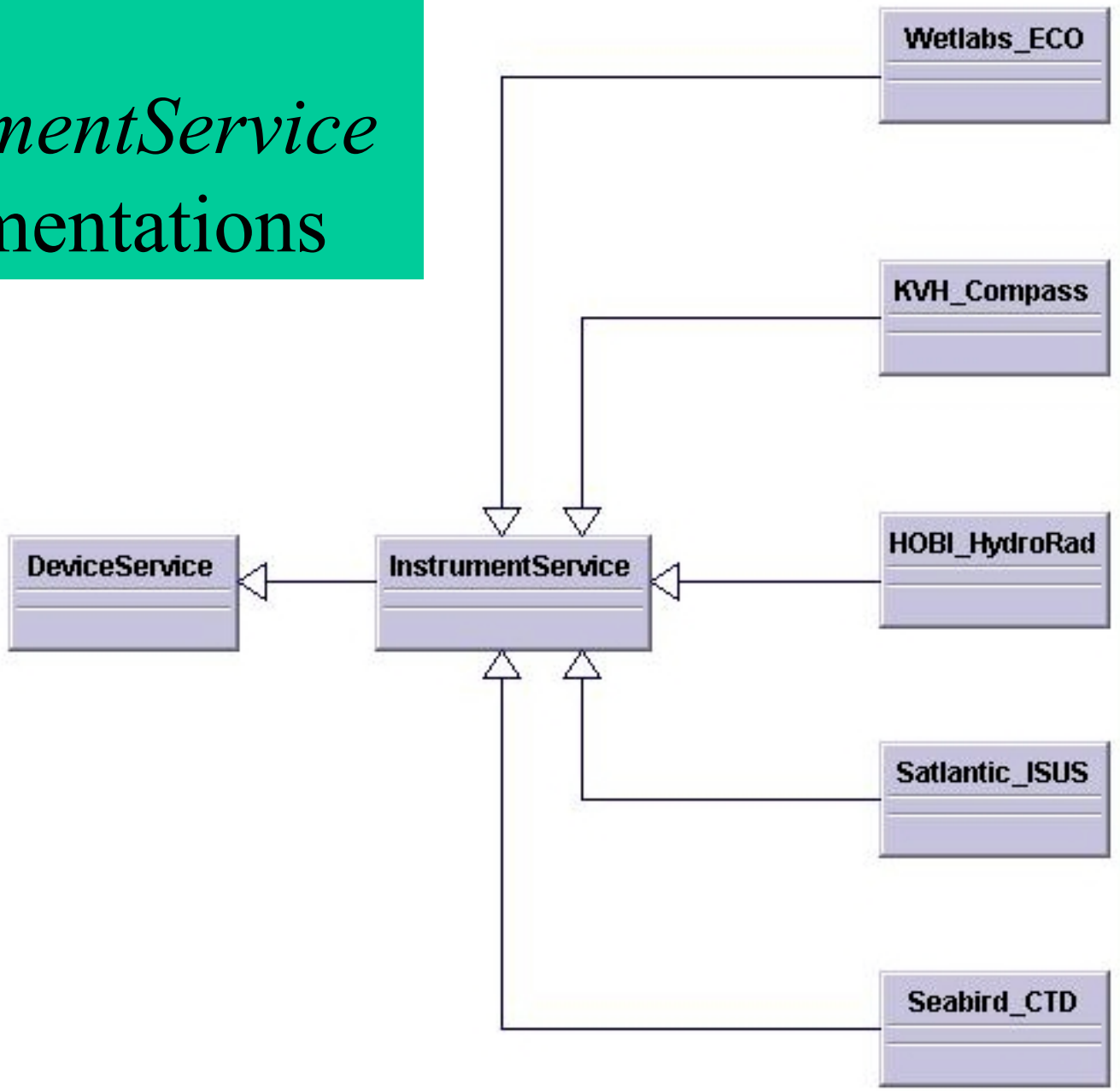
InstrumentService framework

- Defines generic methods for instrument initialization, configuration, data acquisition and logging
  - Subclass can override most generic methods if necessary
- Some methods are *abstract*, i.e. define interface only
  - Subclass is required to implement abstract methods

Subclassing *InstrumentService*



Some *InstrumentService* implementations



InstrumentService framework

- Rapid development
  - Less service-specific code
  - Most of the complicated stuff is in framework
  - Service details tend to be straightforward

- Robust: derived services inherit framework logic

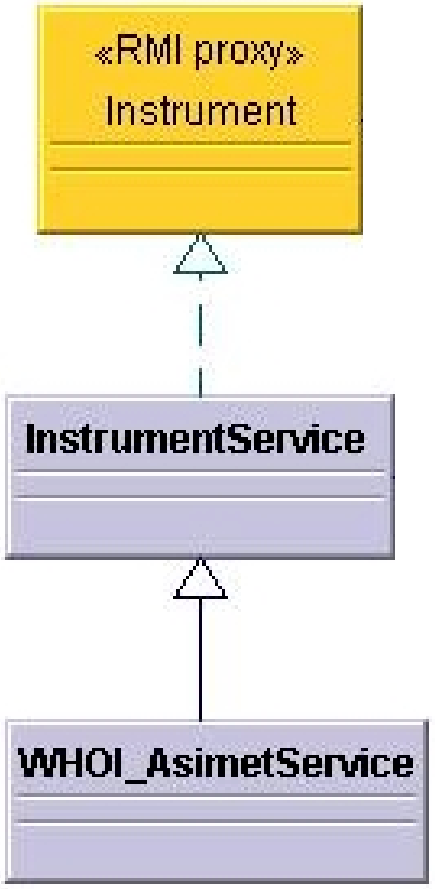
InstrumentService framework

- Easier maintenance: Repairs and enhancements to framework are inherited by all services

- *InstrumentService* code documentation

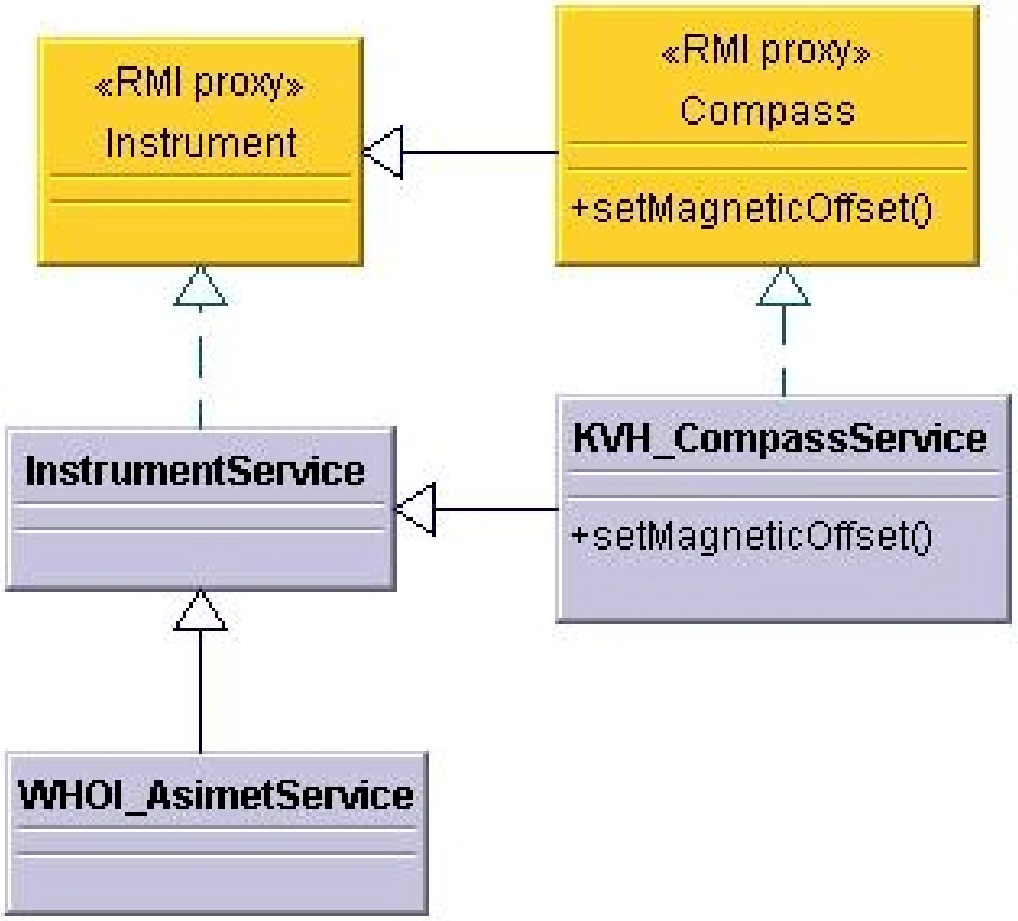
RMI interfaces

- Note that *InstrumentService* implements the *Instrument* RMI interface
- Most SIAM instrument service subclasses also just implement *Instrument*...



RMI interfaces

- Can also extend *Instrument* interface to enable instrument-specific remote methods

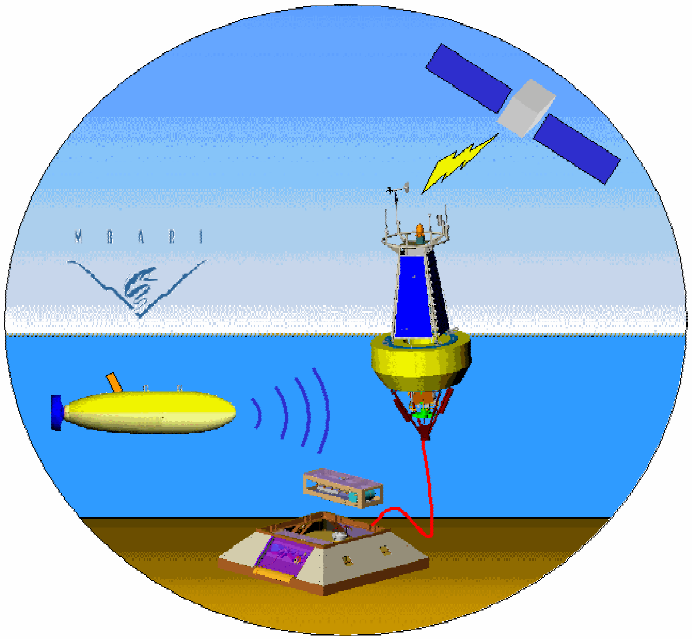


Issues

- Should instrument configuration be generic (e.g. *setProperty()* method), or strongly typed for specific instrument?

- New instrument service must be implemented by Java programmer
  - Do we need to make service implementation simpler?

Device Coordination



Kent Headley

M B A R I

Device coordination

- Services need to coordinate with devices and other services:
  - Shutters
  - Pumps
  - Future applications
- Issues:
  - Must not violate plug-n-work (the platform should have no *a priori* knowledge of coordinating devices/services
  - The mechanism must be generic, because sensors may change
  - Pumps and shutters are fundamentally different from applications

Primary requirement source(s)

- *SIAM DeployedSoftwareRequirements*
  - *3.2.21 et al*

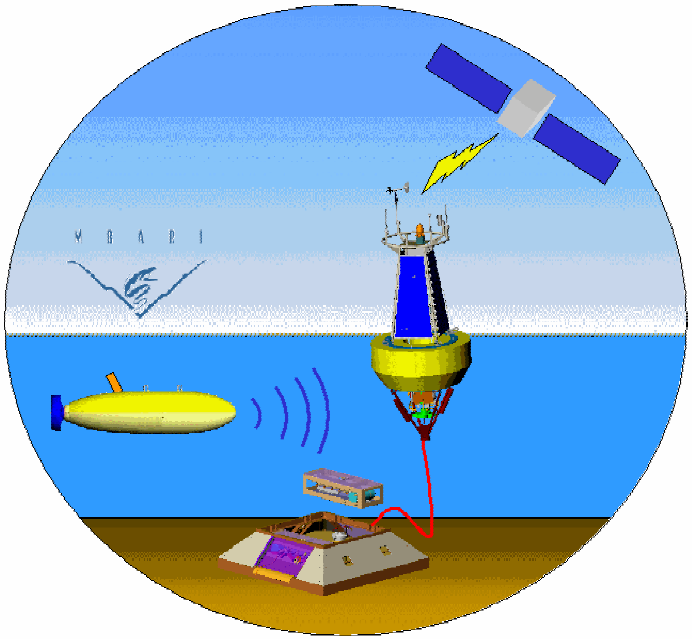
Device coordination

- Possible architectures
  - Best not to re-invent the wheel
  - Registry (rmi)
    - local lookup scope
    - lightweight
    - simple
    - mature standards
  - Discovery protocols
    - rendezvous, jini, salutation…
    - network lookup scope
    - no clear leader in existing protocols
    - development, support road map not clear

Device coordination

- Degree of automation
  - Associations between devices could be formed with human intervention
    - e.g., During scan port
    - Does this bend plug-and-work too far?
    - How much complexity is needed and justifiable for truly automatic device coordination

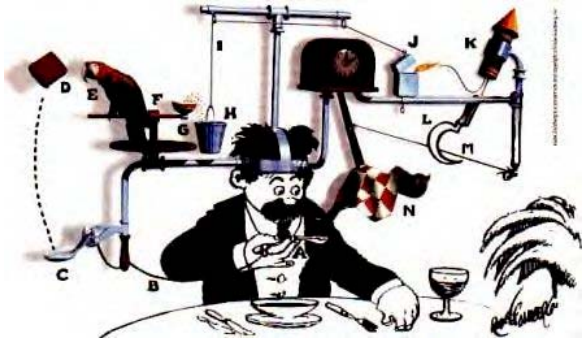
User Interfaces



Kent Headley

User interfaces

- Command Line Interfaces
- Prototype GUI
- Objective of review is to determine:
  - Functional completeness
  - Effectiveness and efficiency of implementations



MBARI

Primary requirement source(s)

- *SIAM User Stories*
- *SIAM DeployedSoftwareRequirements*
  - *2.1.2.n*
- *SIAM OperationsSoftwareRequirements*
  - *2.1.2 et al*

User interfaces

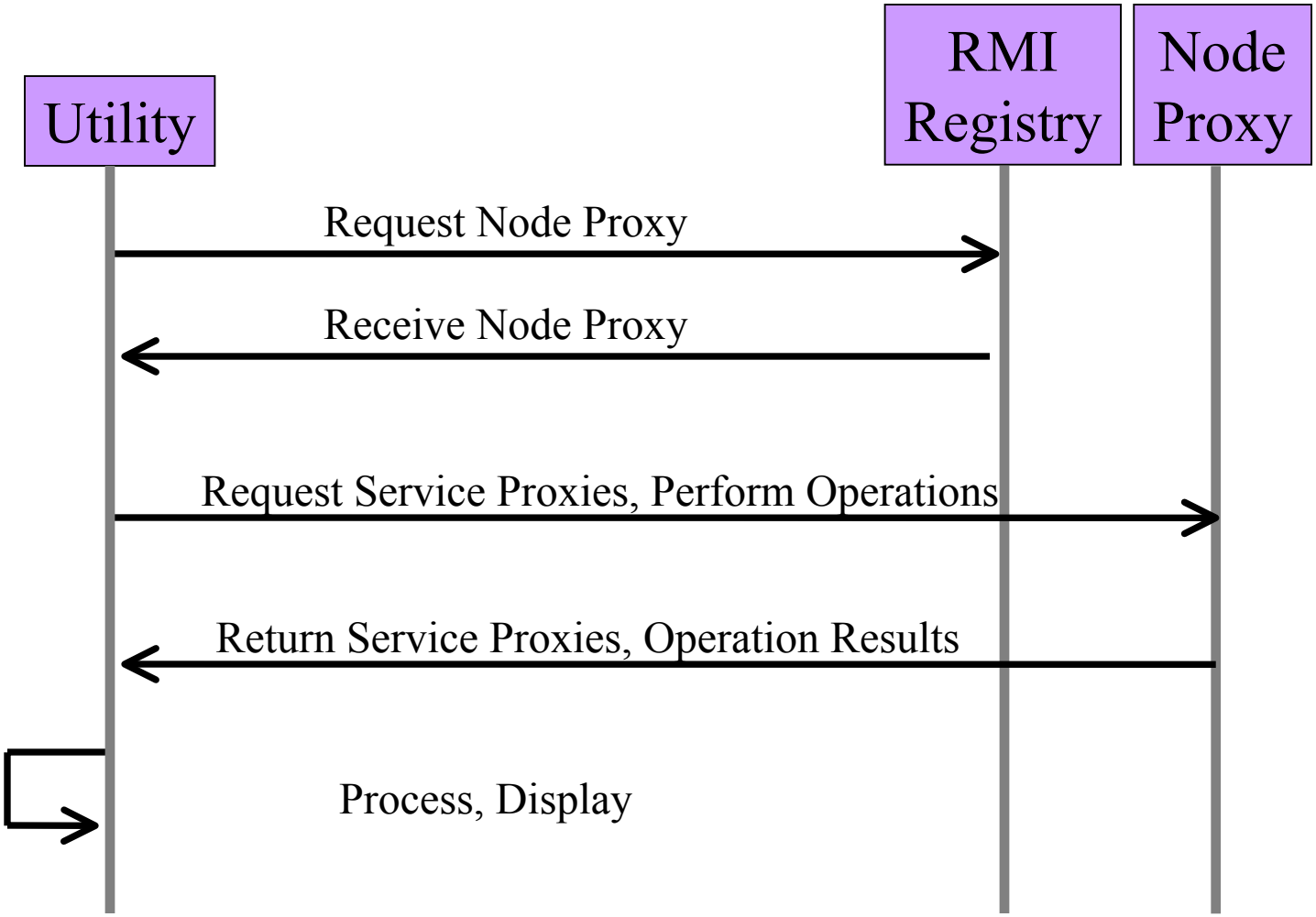
Core Functions

- Configuration
  - DPA
  - SideARM
  - Puck
  - Instrument and node services
  - Portal
- Diagonostics/Utilities
  - DPA
  - SideARM
  - Miscellaneous
- Status/Information
  - Instrument services
  - Logs (data, system)
  - Node hardware
  - Environmental monitoring
- Maintenance
  - Node
  - Instrument control
  - Puck
  - Portal

User interfaces

- Command Line Interface (CLI) utilities
  - Ease of use in the field
  - Preferred for diagnostics and configuration
  - Mixture of standard Linux and custom utilities (shell scripts and Java applications)
- Graphical User Interface (GUI) tools
  - Portable across platforms
  - Suited to information/status queries

Remote utility mechanism



SIAM utilities

Configuration

Service
addSchedule
removeSchedule
addScheduleEntry
removeScheduleEntry
suspendScheduleEntry
resumeScheduleEntry
(get/setProperty)
synch

Puck
mkpuck
Puckit
Puckjar

Node
Discover
locator
mytop

Puck
PuckTest
readPuck
writePuck
switchPuck

Diagnostic/Test/Utils

RMI,Networking
ClientTest
CallbackTest
rxtxTest
comIoTest.sh
socktest

Node
getUUID
portTo485

Logs
convertTime.awk
et2ut.cc

DPA
testDpa
dpaView

Node
rs485/rs485.ops
rfpower/rfpower.ops

Information/Status

Puck
catJar
catPuck

Node
listPorts
portProperties
listSwitches
samplePort
getLastSample
showSchedule
getMetadata

Logs
logView
ut2et.cc

Maintenance

Node
startServices
scanPort/scanPorts
shutdownPort
suspendPort
resumePort
ppp
runppp runpppPersist
runnode
jinistart
node/nodeTest
IpViaUplinkPort

Instruments
nodeGUI/nodeGUI.cyg
conPort/hyperConPort
annotatePort

Portal
portal

dpaView

Name: *dpaView*

Synopsis:
dpaView

Description:
Utility to test and control DPA hardware. Offers complete visibility and control of all DPA registers and settings. By default operates as a Bash sub-shell, taking input from stdin. May also take input from files, allowing common operations to be scripted for convenience.

mkpuck

Name: *mkpuck*

Synopsis:

mkpuck <serviceClassname> <mnemonic> <isiID> <outputJarfile> [<optional param>...]

Description:

Create a puck jar-file, using specified class, service name mnemonc, isiID. Optional parameters are appended to the service.properties file.

logView

Name: *logView*

Synopsis:
logView <isiID> <logDirectory>

Description:
Display log records for instrument with specified ISI ID from the specified log directory.

listPorts

Name: *listPorts*

Synopsis:
listPorts <nodeURL>

Description:
Display the status of all instruments on the specified node.

showSchedule

Name: *showSchedule*

Synopsis:
showSchedule <nodeURL> [<lookAheadSeconds>]

Description:
Display the default sampling schedule for the specified node. This also shows the time remaining until the next scheduled events; by default, it will look ahead 30 seconds for items with absolute schedules. This may be overridden using the optional lookAheadSeconds parameter.

samplePort

Name: *samplePort*

Synopsis:
samplePort <nodeURL> <port>

Description:
Request a sample from the specified port and display the results. This will not alter the sample schedule, though it may potentially cause a scheduled sample to occur later than expected if there is contention.

conPort

Name: *conPort*

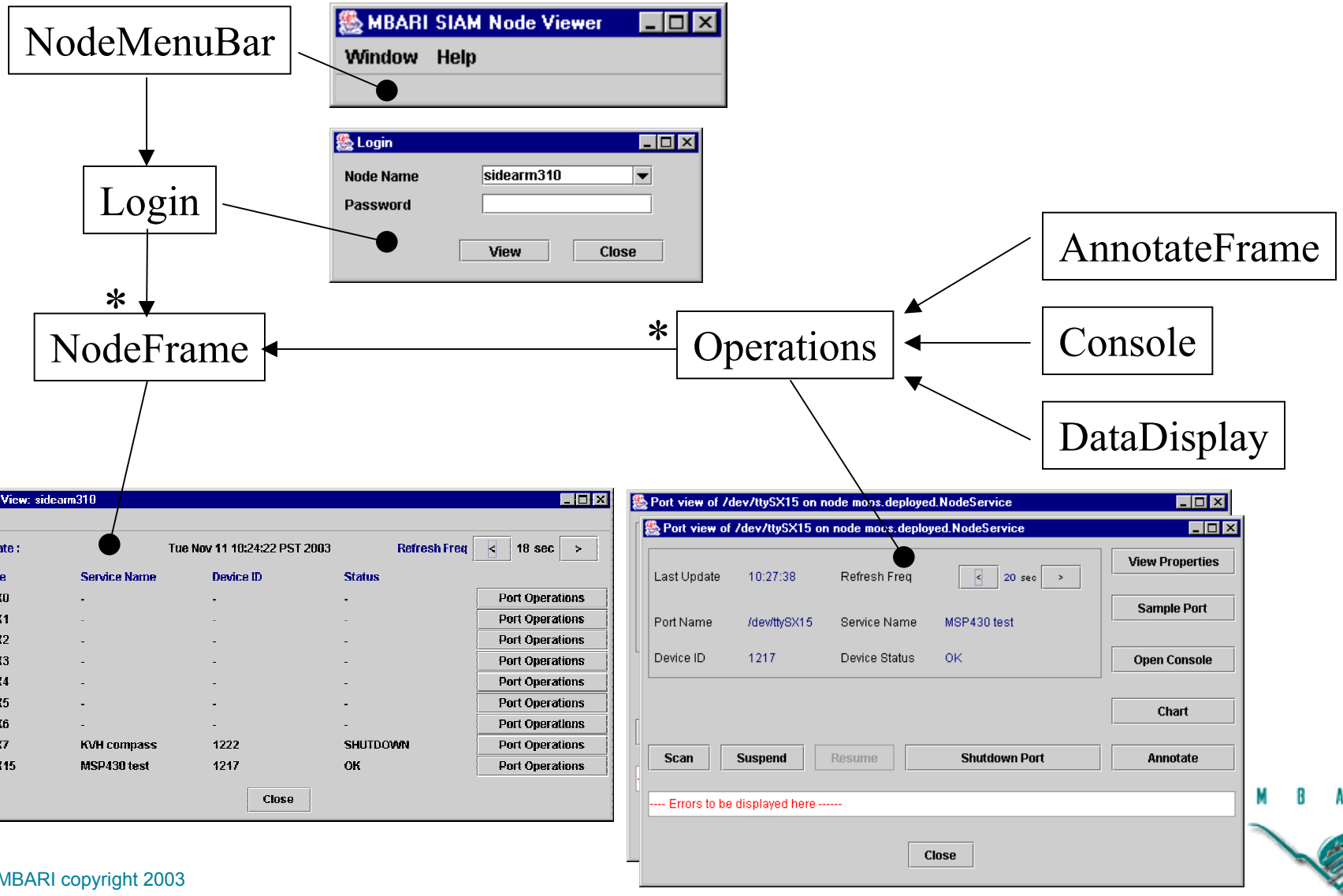
Synopsis:
conPort <nodeURL> <port>

Description:
Establish a virtual serial console to the instrument attached to the specified port. This operation takes control of the serial port and will interrupt scheduled sampling. The session will timeout after a period of inactivity and return control to the instrument service associated with the port.

SIAM GUI

- Implemented using open source JFreeChart classes
- Provides basic operations in test and integration context
- Demonstrate system functions
- Provides easy access to system health and status information

SIAM GUI



SIAM GUI

With the GUI, you can:

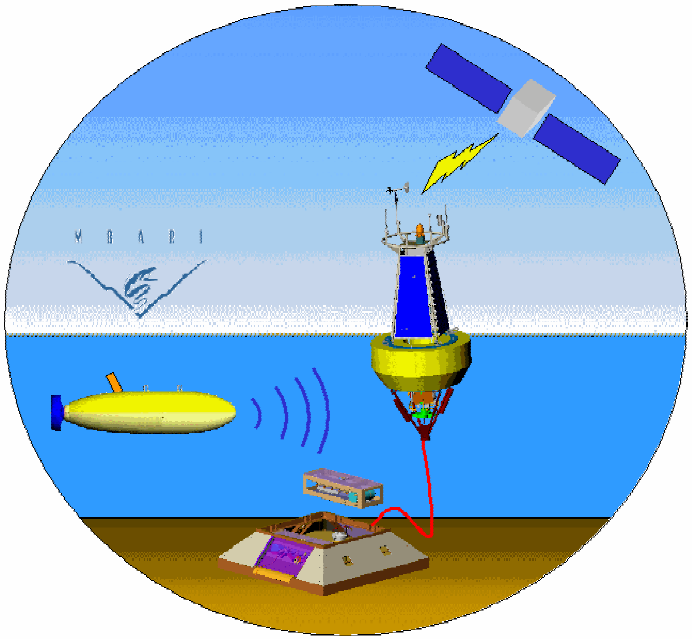
- View node configuration and port status
- Shutdown, Scan ports
- Establish an instrument console
- Request an immediate (or most recent) sample from an instrument
- Plot environmental data in real time
- Insert a message packet into instrument data stream
- View instrument service properties

User interfaces

Open Issues

- Class loading over radio link
  - Need to cache classes on workstation
- GUI Features
  - Scheduling
  - Instrument service property changes
  - Generalized charting mechanism
  - Use data streams from portal instead of polling
  - Code clean-up
  - Feedback

Transition to Operations

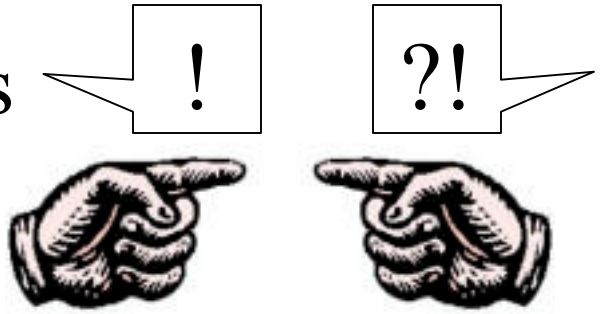


Kent Headley

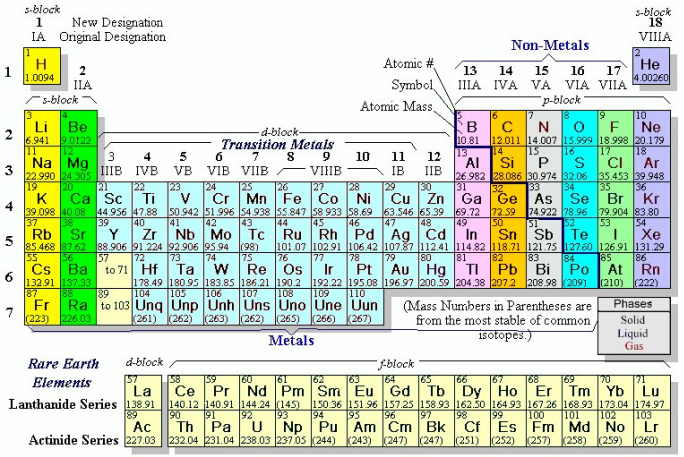
M B A R I

Transition to operations

- Operational responsibilities
  - Deployed Subsystem
  - Operations Subsystem



- Transition elements
  - Testing
  - Strategies
  - Training
  - Resources



Primary requirement source(s)

- *SIAM User Stories*
  - Operations user stories
- *SIAM Behaviors Required by SSDS:*
  - Test plan

Proposed responsibilities

	Software engineering	Support engineering	Science Users	Observatory Support
Instrument Services	d	ad	du	a
Metadata	d	ad	ad	u
Portal	d	ad	u	a
Platform	d	ad	u	a
Instruments	u	ad	u	a
User Interfaces	du	ad	u	a

a: administrator (Affects operational changes)
d: developer (Participates in modification, new development)
u: user

Transition elements: testing

SIAM Test Plan

- Deploy early and often
  - AOSN
  - Parking lot deployment
  - 2004 MTM, CIMT
- 23-6 Bench system
  - Available for user testing
- Demos/tests depicting user stories
- Automated test procedures
- Unit testing…

Transition elements: strategies

Strategies for making a smooth transition

- Deploy early and often
- Ramp up participation by Observatory Support in early deployments:
  - Parking lot
  - MOOS test mooring (MTM)
  - CIMT
- Commit user resources for participation in XP process
- Support Engineering participation
  - Development (especially at system interfaces)
  - System integration
  - Transition

Transition elements: training

Proposed SIAM training workshop

- 10 days budgeted for training in 2004
- Objectives
  - Provide working knowledge of system operations
  - Expose weaknesses in system implementation
- Multi-day, hands-on workshop format
- Content:
  - System overview
  - Interfaces and utilities
  - Hardware configuration
  - Software configuration
  - Test procedures
  - Basic Operations
  - Troubleshooting
- Could be tailored to science users and application developers

Transition elements: resources

Technical resource deliverables

Documentation (resource request)

- 20 days budgeted for documentation in 2004
- Portable documents, formatted for easy printing
  - PDF
- Architectural overview
- Javadoc used for code documentation (UML?)
- Op-sheets, procedural manual (HOWTO)
- Utilities guide
- Troubleshooting guide
- Puck developer's kit?

Technical support

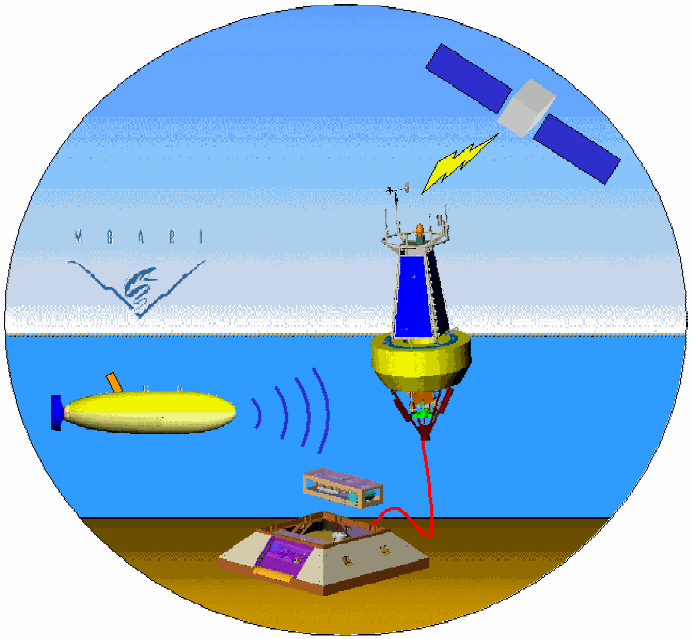
Transition open issues

- Bandwidth arbitration
  - There will be many users potentially competing for limited bandwidth resources:
    - Portal
    - Observatory support
    - Science users
    - Software applications (possibly asynchronously and in large numbers)
  - There are currently no access control policies in place
  - Who/what will enforce access control policies?

Transition open issues

- Unfinished core design elements
  - Power management
  - Device coordination
  - Environmental alarm handling
  - Event propagation and response
  - Adaptive sampling
  - Metadata management
  - Security
- Documentation and training
- Unit testing – more, better
- Know-it-all(s) needed (no one has end-to-end knowledge of system)

Discussion



M B A R I