# Building a Sensor Network Controller

Michael Pigg
Chariot Solutions

November 5, 2010

This work is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States License.

# Chariot Solutions

Practical, smart software development powered by Java, open source and emerging technologies

# Agenda

- What are we building?
- Why use OSGi?
- Look at interesting code features
- Run the system

# An engineer* walks into a room ...

*Software guy with old, dusty electrical technology degree, really

and notices it's too cold.

He walks into another and notices it's too warm.

# Hire someone to fix it?

# No Way!

# Build a system to monitor temperature in multiple rooms.

# What should it do?

# Collect temperature data from multiple locations

# Record data for later analysis

# Requirements

No wires

# Easily add new capabilities

# Proposed Solution

- Hardware
  - Custom wireless sensor module
- Software
  - Controller software based on Apache Felix

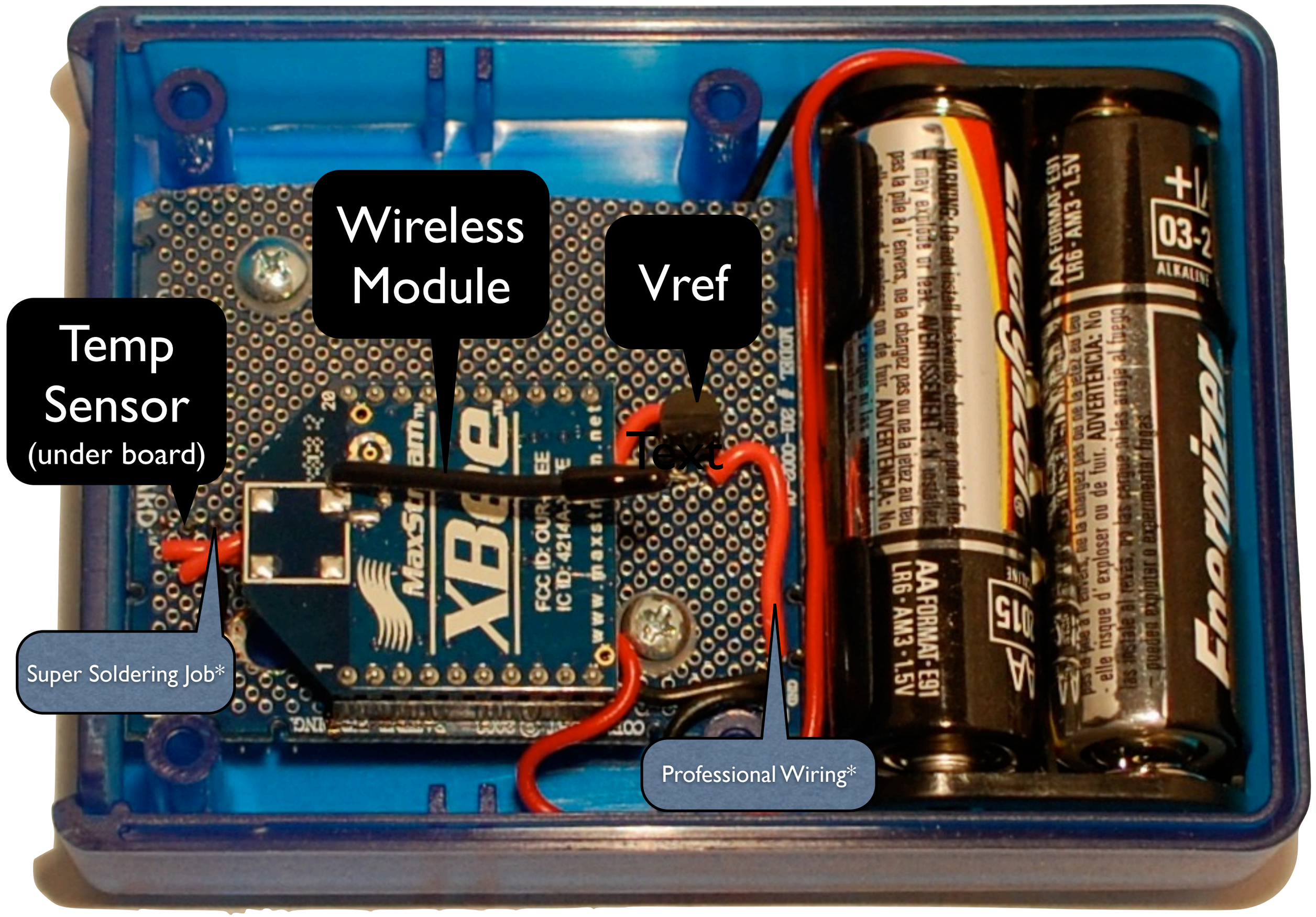
# Why OSGi and Felix?

- Highly modular architecture
  - Capable of being updated at runtime
- Already implemented solutions
  - console shell commands
  - configuration implementation
  - web-based configuration UI

# Hardware

# Sensor Module

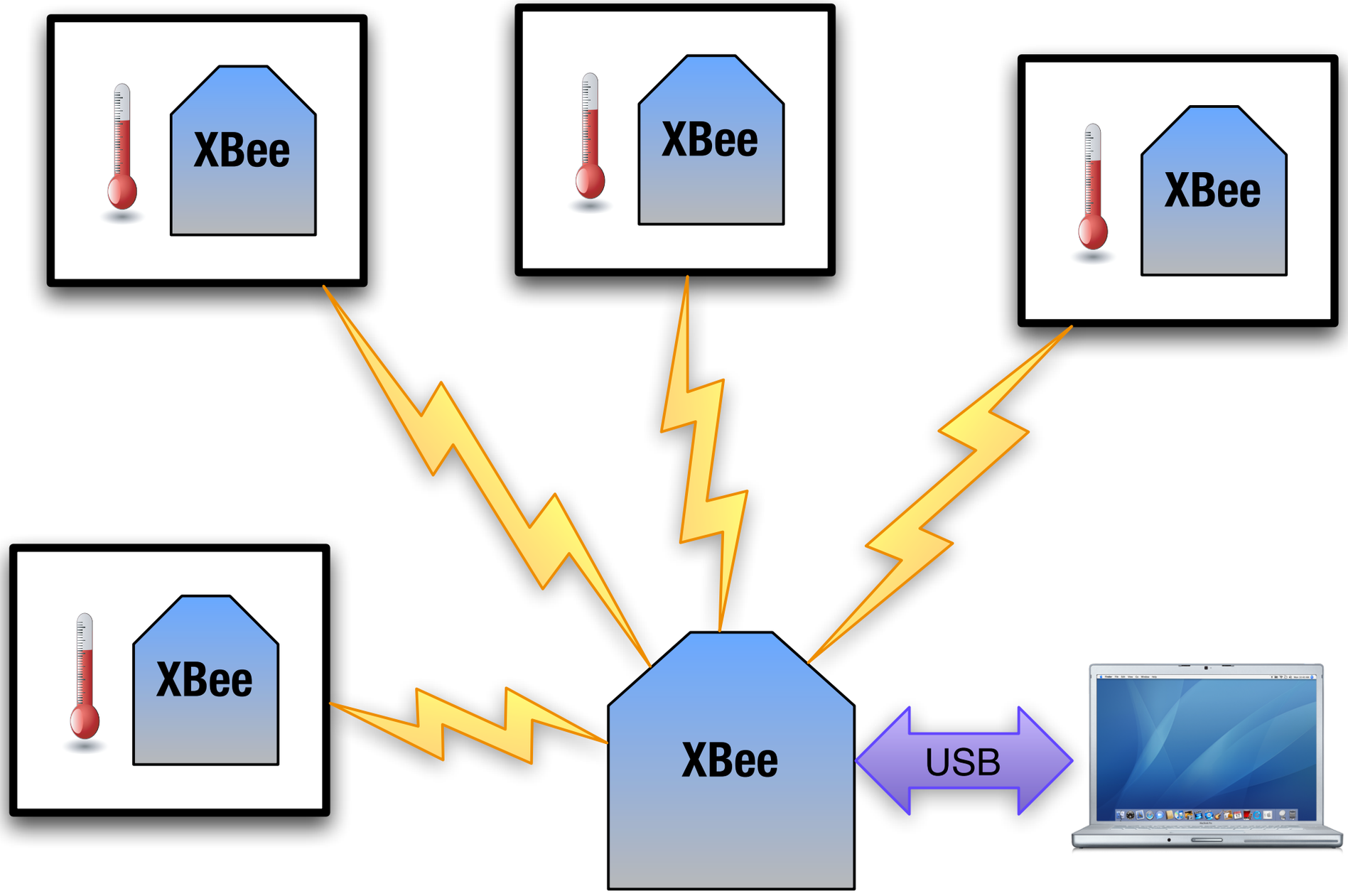
- Digi XBee wireless mesh network module
- Has integrated ADC
- Operates in voltage range of 2 AA batteries
- Relatively low power consumption


Wireless Module
Vref
Temp Sensor (under board)
Text
Super Soldering Job*
Professional Wiring*

*Not so much

# Life of a Node

- Sleep for 5 minutes, then wake up
- Sample ADC port
- Send sample data over network
- Go back to sleep


XBee
XBee
XBee
XBee
XBee
USB

# Software

# XBee Communication

# Connect XBee to PC

- XBee to USB
- Appears as serial port

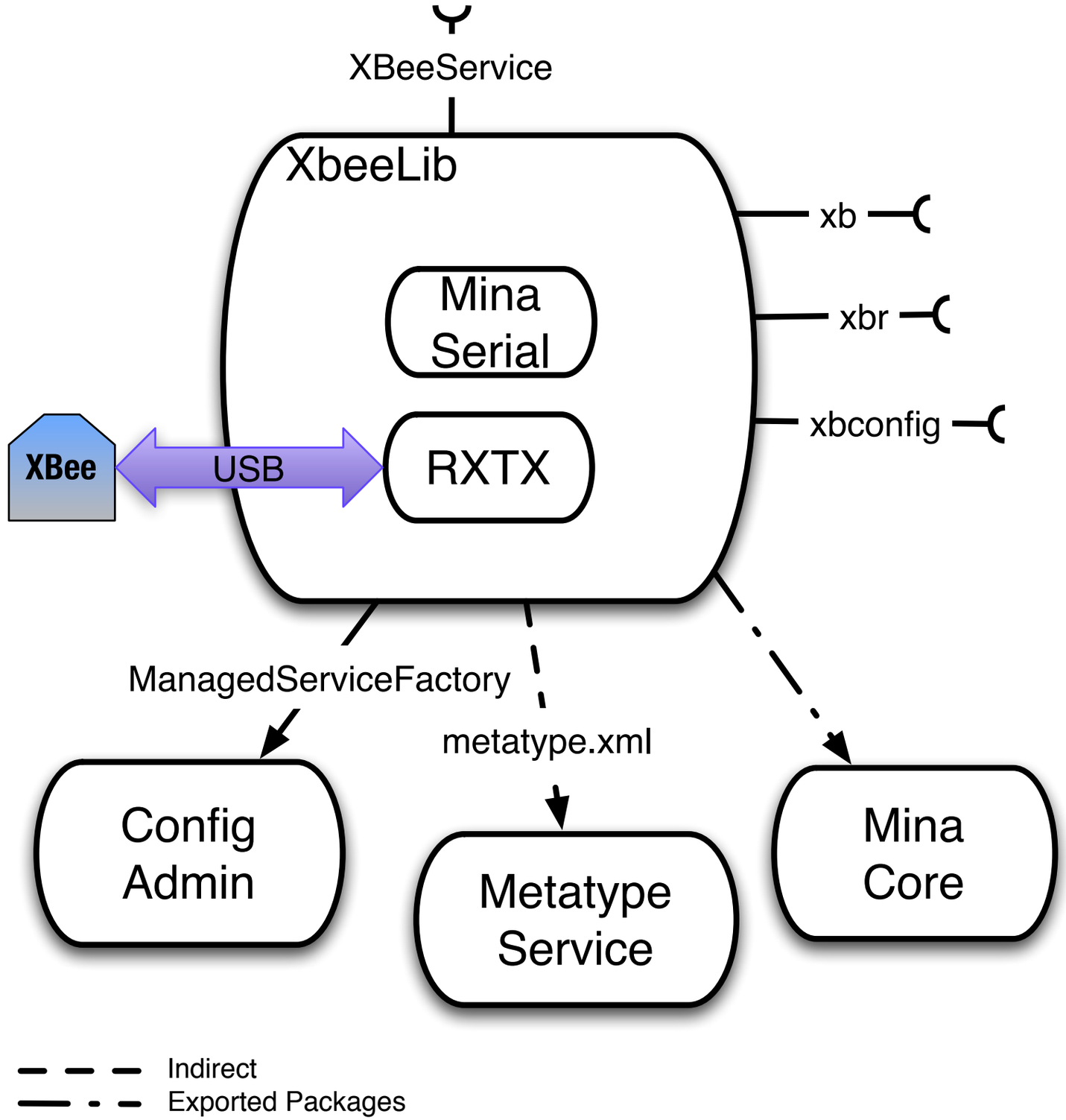
# XBee has custom protocol

# XBeeLib

- Implementation of XBee API protocol
- Open source, BSD license
- http://kenai.com/projects/xbeelib

# Apache Mina

- Eases implementation of custom protocol encoder/decoder
  - Support for fragmented packets
- Comes with serial port transport


XBeeService
XbeeLib
xb
xbr
xbconfig
Mina
Serial
RXTX
XBee
USB
ManagedServiceFactory
metatype.xml
Config
Admin
Metatype
Service
Mina
Core
Indirect
Exported Packages

# RXTX requires native library

# Where to put native library?

# How to get native library installed?

# Bundle-NativeCode

- Parameters
  - path to native code in bundle JAR
  - osname (MacOS, Linux, etc.)
  - processor (x86, PowerPC, etc.)
  - osversion

```
<plugin>
   <groupId>org.apache.felix</groupId>
   <artifactId>maven-bundle-plugin</artifactId>
   <version>2.0.1</version>
   <extensions>true</extensions>
   <configuration>
      <instructions>
         <Import-Package>*</Import-Package>
         <Export-Package>net.michaelpigg.xbeelib.protocol,net.michaelpigg.xbeelib</Export-Package>
         <Embed-Dependency>rxtx,mina-transport-serial</Embed-Dependency>
```

**`<Bundle-NativeCode>lib/rxtx/mac/x86_64/librxtxSerial.jnilib; osname=MacOSX; processor=x86_64</Bundle-NativeCode>`**

```
         <Bundle-Activator>net.michaelpigg.xbeelib.impl.BundleActivator</Bundle-Activator>
      </instructions>
   </configuration>
</plugin>
```

# XBeeLib dependencies

- Mina and most other dependencies are deployed as bundles
- Mina serial transport and RXTX are embedded into XBeeLib bundle
  - RXTX is not OSGi-ready
  - Mina serial depends on RXTX

```
<plugin>
   <groupId>org.apache.felix</groupId>
   <artifactId>maven-bundle-plugin</artifactId>
   <version>2.0.1</version>
   <extensions>true</extensions>
   <configuration>
      <instructions>
         <Import-Package>*</Import-Package>
         <Export-Package>net.michaelpigg.xbeelib.protocol,net.michaelpigg.xbeelib</Export-Package>
         <Embed-Dependency>rxtx,mina-transport-serial</Embed-
         Dependency>
         <Bundle-NativeCode>lib/rxtx/mac/x86_64/librxtxSerial.jnilib; osname=MacOSX; processor=x86_64</Bundle-
         NativeCode>
         <Bundle-Activator>net.michaelpigg.xbeelib.impl.BundleActivator</Bundle-Activator>
      </instructions>
   </configuration>
</plugin>
```

# First Contact

- XBeeLib provides simple shell commands for sending commands to XBee modules
  - xb - send command to local XBee
  - xbr - send command to remote XBee

# Shell Commands

- Felix provides the Gogo shell as of 3.0
- Bundles contribute commands by registering a service with two properties
  - osgi.command.scope
  - osgi.command.function

```
public void start(BundleContext context) throws Exception {
  Hashtable cmdProps = new Hashtable();

  // these commands are in scope “xbee”
  cmdProps.put(CommandProcessor.COMMAND_SCOPE, “xbee”);

  // “xb” and “xbr” are functions that can act as commands
  cmdProps.put(CommandProcessor.COMMAND_FUNCTION,
    new String[] {“xb”, “xbr”});

  context.registerService(
      ToXbeeCommand.class.getName(),
      new ToXbeeCommand(context),
      cmdProps);
}
```

# Listing Shell Commands

```
g! help
felix:bundlelevel
   .....
felix:which
gogo:cat
   ....
gogo:until
obr:deploy
   ....
obr:source
xbee:listports
xbee:xb
xbee:xbconfig
xbee:xbr
xbee:xmon
xbee:xsample
g!
```

# Using Commands

Send node discover command

```
g! xb nd
ND response: Frame ID = 109;Status = OK;Address =
0013a200403c5e93;Signal strength = 32;

g! x1 = "0013a200403c5e93"
0013a200403c5e93

g! xbr $x1 is
IS response: Frame ID = 11;Status = OK;Data: 010200013c

g!
```

Request sample

# Transform Data

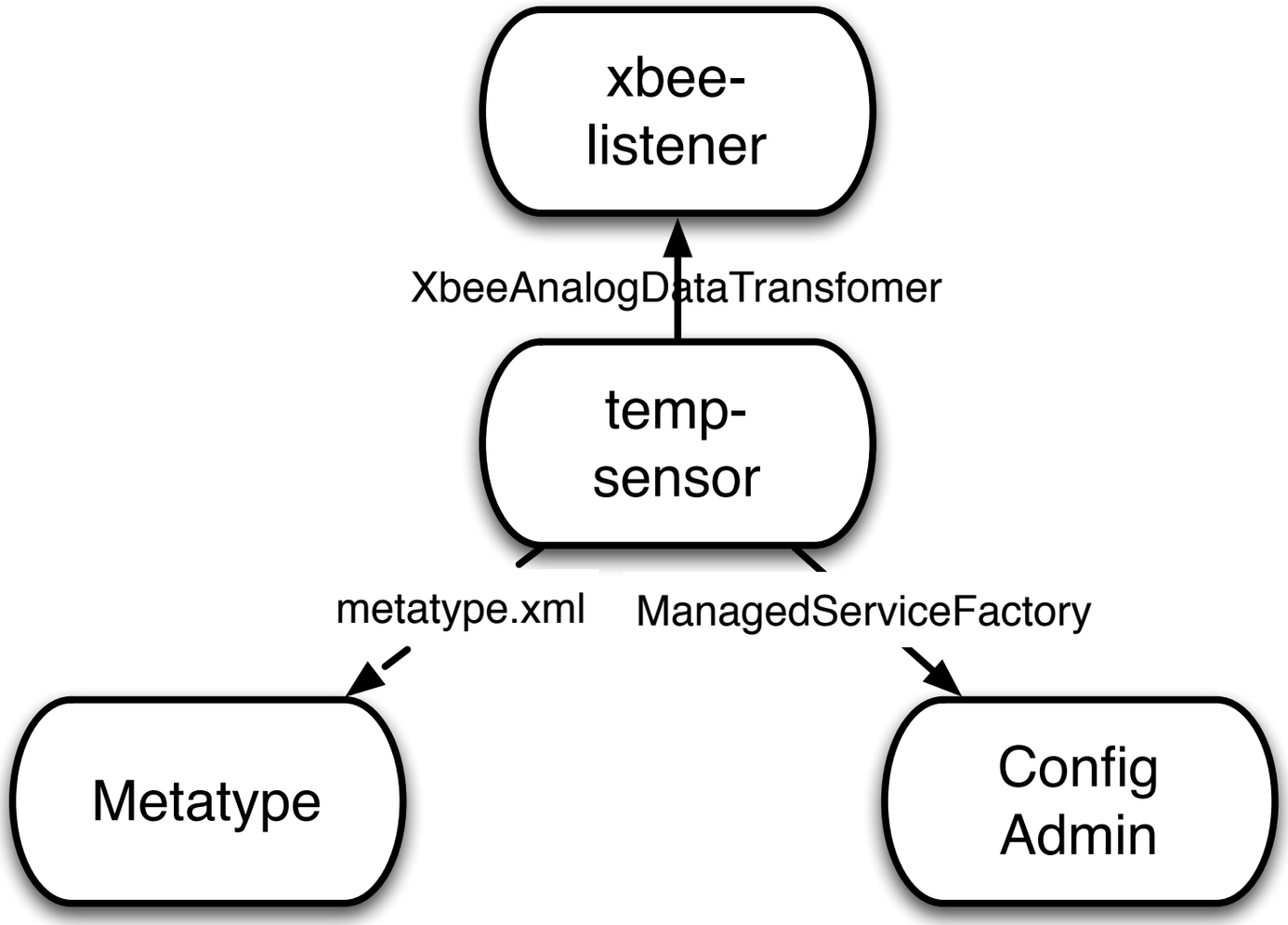
# Incoming Data

- Number resulting from ADC sample on XBee

# XbeeAnalogData Transformer Interface

- Implementations convert raw ADC reading to useful data
- TemperatureSensorTransformer does this for our temperature sensor hardware

# xbee-temp-sensor bundle



# Sample with no Transformer

```
g! xsample $x1
Command returned with status OK
Raw data in response:1200dd
Number of samples 1
Digital I/O
     No digital I/O data in response.
Analog Data
     1: DD(221)
g!
```

# Configuring Temperature Sensors

# Config file?

NO!

# Configuration Admin Service

- Stores configuration for a service
- Sends configuration to service
  - at startup
  - when configuration changes

# ManagedService or ManagedServiceFactory?

- ManagedService
  - Configures at most one instance
- ManagedServiceFactory
  - Configures one or more instances

# We need to configure multiple sensors

# Implementation

- Make up a PID
- Implement ManagedServiceFactory
- Register implementation as a service with the chosen PID

```
public class XbeeTemperatureSensorFactory implements ManagedServiceFactory {

    public void updated(String pid, Dictionary properties) throws ConfigurationException {
        // get configuration properties and copy to registration properties
        String address = properties.get(LOCATION_ADDRESS);
        String name = properties.get(LOCATION_NAME);
        Dictionary<String, String> registrationProperties = new Hashtable<String, String>();
        registrationProperties.put( LOCATION_NAME, name);
        registrationProperties.put( LOCATION_ADDRESS, address);

        // create new transformer
        TemperatureSensorTransformer transformer = new TemperatureSensorTransformer();

        // register newly configured transformer
        ServiceRegistration registration = bundleContext.registerService(
                            XbeeAnalogDataTransformer.class.getName(),
                            transformer,
                            registrationProperties);
    }
}
```

```
public class BundleActivator implements BundleActivator {
    // factory PID - will be used as base PID for configured instances
    public static String PID = "com.pigglogic.phomenet.xbee.sensor.temperature";

    public void start(BundleContext context) throws Exception {
        // create ManagedServiceFactory instance
        sensorFactory = new XbeeTemperatureSensorFactory(context);
        Dictionary properties = new Hashtable<String, String>();
        // set SERVICE_PID property to our factory PID
        properties.put(Constants.SERVICE_PID,  PID);
        // register factory instance
        factoryRegistration = context.registerService(
                ManagedServiceFactory.class.getName(),
                sensorFactory,
                properties);
}
```

# What is allowed in configuration?

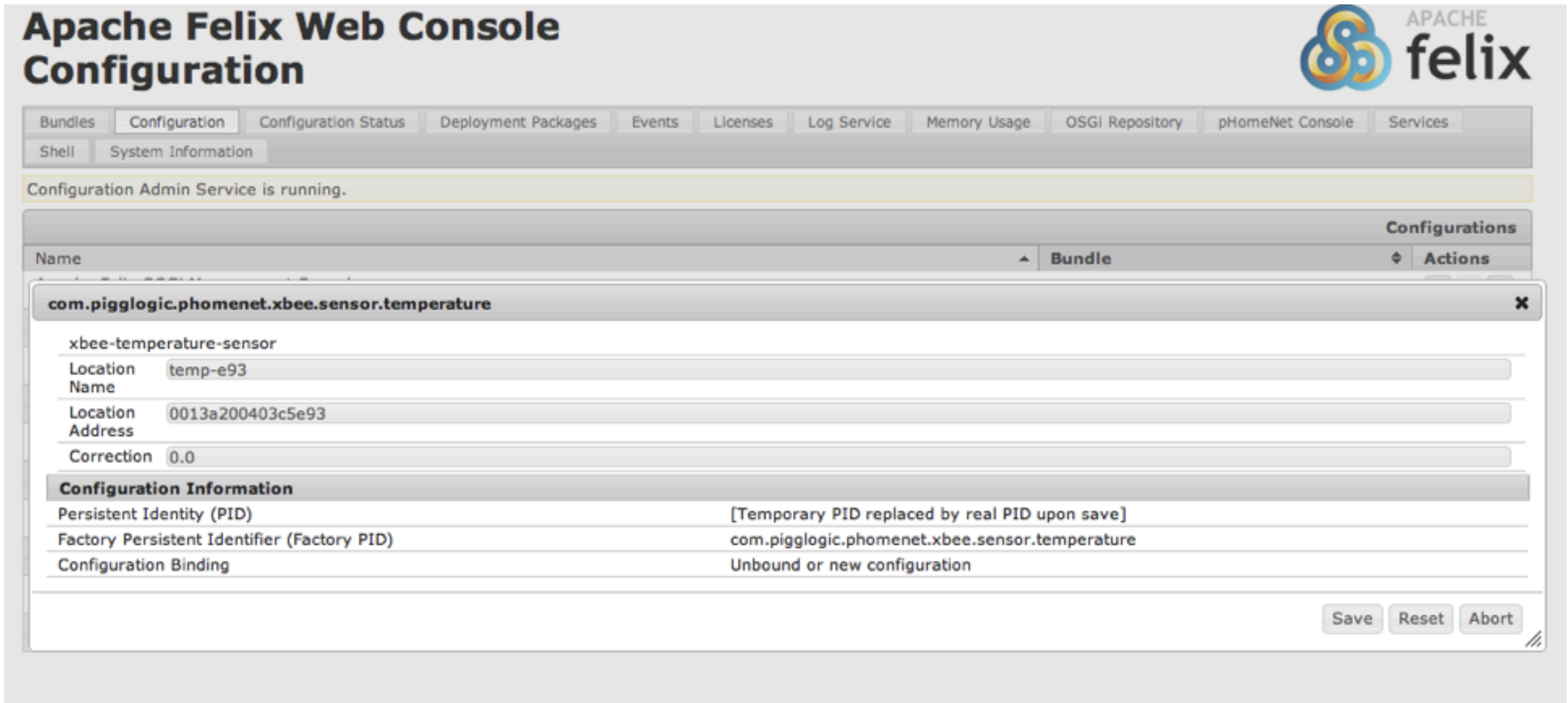
- What are the attribute names?
- What type of data do they take?
- Are they required?

# Metatype Specification

- Specify attributes used in configuration
  - name, type, required, default value, etc.
- XML file in OSGI-INF/metatype
- Felix web console will use this data to dynamically create a configuration UI

```
<metatype:MetaData xmlns:metatype="http://www.osgi.org/xmlns/metatype/v1.0.0">

   <OCD description="xbee-temperature-sensor"
      name="com.pigglogic.phomenet.xbee.sensor.temperature"
      id="com.pigglogic.phomenet.xbee.sensor.temperature">

   <AD name="Location Name"  id="location.name" required="true" type="String"
         default="Temperature Sensor"/>
   <AD name="Location Address"  id="location.address" required="true" type="String"/>
   <AD name="Correction"  id="location.correction" required="false" type="Double" default="0.0"/>

   </OCD>

    <Designate pid="com.pigglogic.phomenet.xbee.sensor.temperature"
          factoryPid="com.pigglogic.phomenet.xbee.sensor.temperature">
       <Object ocdref="com.pigglogic.phomenet.xbee.sensor.temperature"/>
    </Designate>

</metatype:MetaData>
```

# Configure Sensor

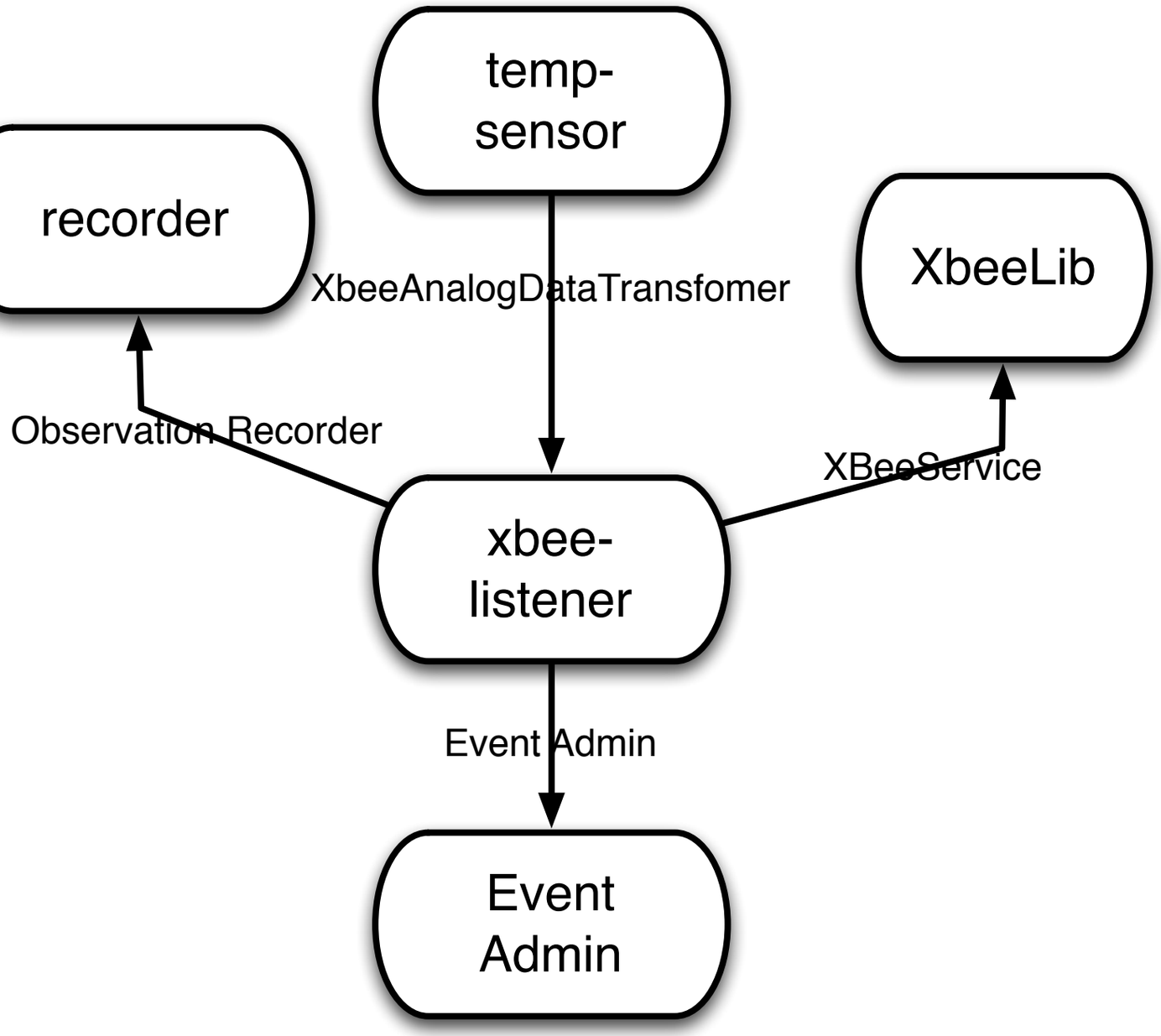


# Sample with Transfomer

```
g! xsample $x1
Command returned with status OK
Raw data in response:12013e
Number of samples 1
Digital I/O
     No digital I/O data in response.
Analog Data
     1: 13E(318)
Transformer reports value of -11.609971
```

# Collecting and Recording Data

# xbee-listener bundle



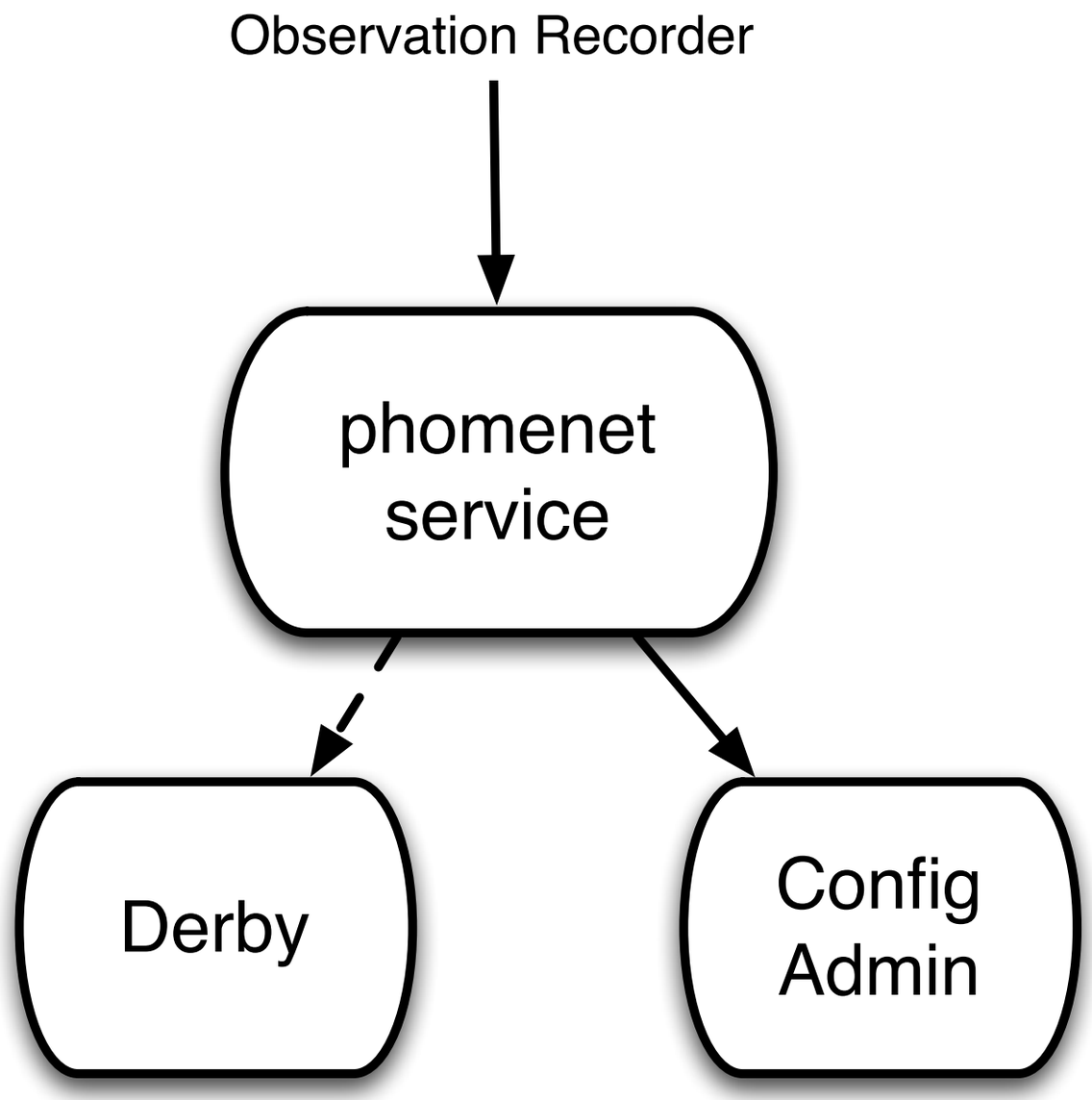
# Piping Hot Data

- xbee-listener gets data from XBeeService, but what to do with it?
- Needs to find a transformer that will handle the data
- There should be a number of transformers registered
- Chooses based on source address

# Recording data

- xbee-listener records transformed data
- Finds an instance of ObservationRecorder service

# Default Storage



# NoSql?

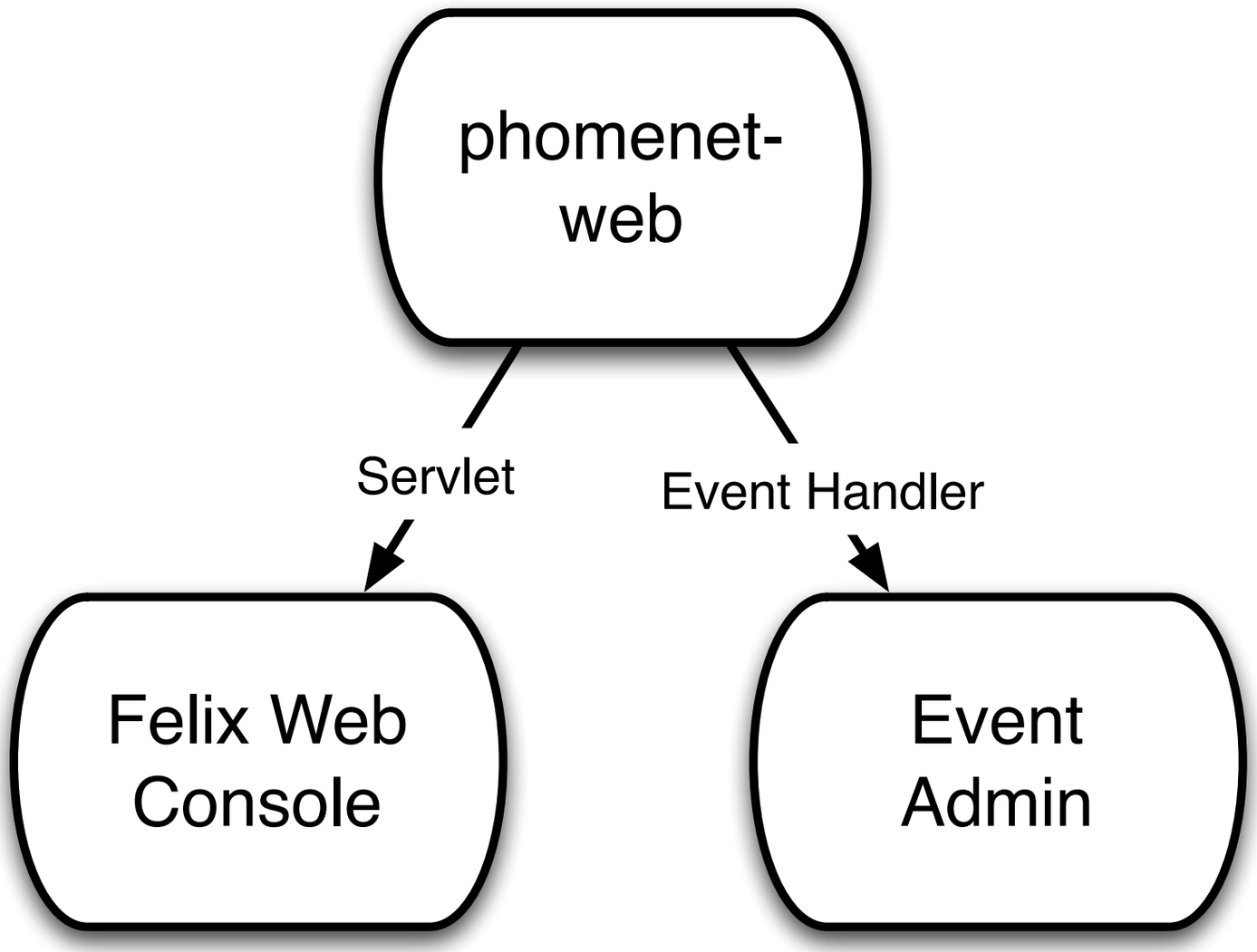
- What if we wanted to be cool and store data in Cassandra?
- We could provide a different bundle that implements ObservationRecorder
  - but uses Cassandra instead of Derby

# Visualizing Sensor Data

# Show me a GUI!

- Web page GUI
- Show latest observation for each sensor

# phomenet-web



# Observation Events

- xbee-listener posts an OSGi event when data is transformed
- Event topic is “phomenet/Observation/Temperature/<address>”

# OSGi Event Admin

- Bundles can post events
- Bundles can register to be notified of posted events

# Posting an Event

```
// build Map containing event data
Map<String, Object> eventProperties = new HashMap<String, Object>();
eventProperties.put("location.name", locationName);
final String addressString = frame.getSourceAddress().toString();
eventProperties.put("location.address", addressString);
 eventProperties.put("observedValue", value);

// create Event with topic and properties
final Event event = new Event(
     "phomenet/Observation/Temperature/" + addressString,
     eventProperties);

 // call postEvent with event
 eventAdmin.postEvent(event);
```

# Listening to Events

```
public void start(BundleContext context) throws Exception {

   Hashtable<String, Object> props = new Hashtable<String, Object>();

   // Felix web console extension properties
   props.put("felix.webconsole.label", "phomenet");
   props.put("felix.webconsole.title", "pHomeNet Console");

   // property to say that we want all Observation events
   props.put(EventConstants.EVENT_TOPIC, "phomenet/Observation/*");

   // register under Servlet and EventHandler interfaces
   webConsoleRegistration = context.registerService(
      new String[] {Servlet.class.getName(), EventHandler.class.getName()},
      new PhomenetConsolePlugin(context), props);
}
```

# Extending Web Console

- Register a class that extends HTTPServlet
- Add properties
  - felix.webconsole.label - used in URL
  - felix.webconsole.title - title of page

# Registering Extension

```
public void start(BundleContext context) throws Exception {

    Hashtable<String, Object> props = new Hashtable<String, Object>();

    // Felix web console extension properties
    props.put("felix.webconsole.label", "phomenet");
    props.put("felix.webconsole.title", "pHomeNet Console");

    // property to say that we want all Observation events
    props.put(EventConstants.EVENT_TOPIC, "phomenet/Observation/*");

    // register under Servlet and EventHandler interfaces
    webConsoleRegistration = context.registerService(
            new String[] {Servlet.class.getName(), EventHandler.class.getName()},
            new PhomenetConsolePlugin(context), props);
}
```

# Resources

- XBeeLib - kenai.com/projects/xbeelib
- pHomeNet - kenai.com/projects/phomenet

# Resources

- XBee 802.15.4 OEM module
  - http://www.digi.com/products/wireless/point-multipoint/xbee-series1-module.jsp#overview
- Droids XBee USB board
  - http://www.droids.it/990_002.html
- MCP9700 Temperature Sensor
  - http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en022289