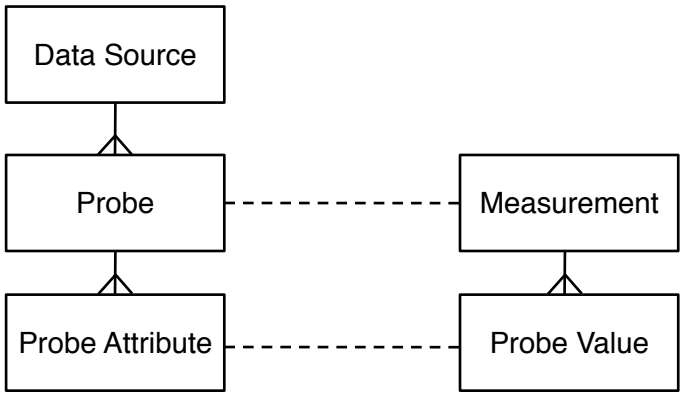
Y2 Snapshot 0.6 of Monitoring Framework

This note gives a brief overview of the Year 2 snapshot 0.6 of the monitoring framework.

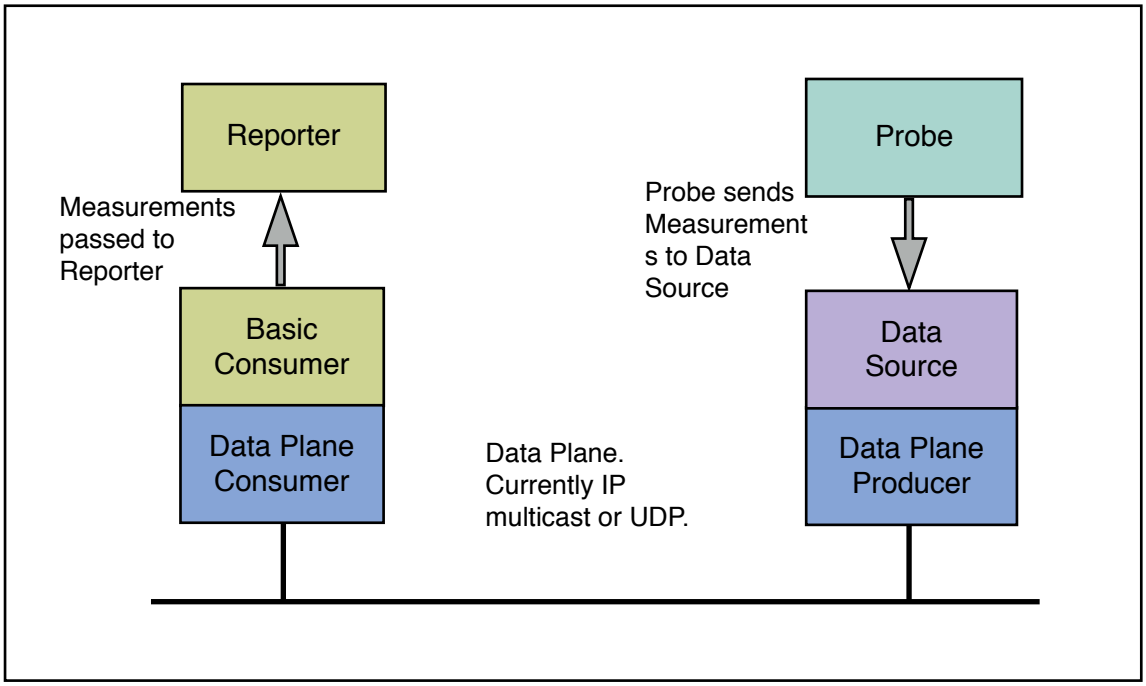
This release implements the basics of the framework as described in the previous documents and discussions we have had. It is written in Java, and is available for anyone to evaluate and modify.

There are implementations of the elements presented in the following model. A DataSource acts as a control point and a container for one or more Probes. Each Probe defines the attributes that it can send in a set of ProbeAttribute objects, that specify the name, the type, and the units of each value that can be sent.

When a Probe sends a Measurement, the values sent are directly related to the Probe Attributes.



The Probe sends each Measurement to the Data Source for transmission. The Data Source passes these measurements onto the Data Plane, where they are encoded into an on-the-wire format, and then sent over the Data Plane distribution network. The receiver of the monitoring data decodes the data and passes reconstructed Measurements to the



monitoring consumer.

In this snapshot, the Data Plane implementation can be done using (i) IP multicast, or (ii) as UDP. The distribution is independent of the core parts of the system, and can easily be replaced by a different Data Plane distribution framework, such as publish/subscribe.

This implementation also supports the Info Plane and the Control Plane, as described in the other monitoring documents. The Info Plane allows the receivers of Measurement data to lookup the fields received to determine their names, types, and units. This data is put into the Info Plane by the Data Sources or by the Service Manager. For this implementation the Info Plane has the inclusion of a distributed information model using a Distributed Hash Table (DHT). This DHT information model has nodes that use a separate network to interact among one another.

Currently, the Control Plane has no implementation but the interface has been defined. This will allow us to design and write a Control Plane plugin at a later date.

Structure

The implementation is structured with 12 main packages. They are presented in alphabetical order below:

Packages	Description
eu.reservoir.monitoring.appl	This package provides classes that are at the application level.
eu.reservoir.monitoring.appl.datarate	This package provides classes that are used to specify data rates for Probes in different ways.
eu.reservoir.monitoring.appl.host.linux	This package provides classes that are used for Probes that monitor Linux hosts.
eu.reservoir.monitoring.appl.vee.sge	This package provides classes that are used for monitoring the Sun Grid Engine (SGE) running inside a virtual machine (VEE).
eu.reservoir.monitoring.core	This package provides classes that are at the core of the monitoring framework, including: Data Source, Probe, Probe Attribute, Measurement, and Probe Value.
eu.reservoir.monitoring.core.list	This package provides classes that are used for embedding List data within a Measurement.
eu.reservoir.monitoring.core.plane	This package provides classes that define the Planes of the monitoring framework.
eu.reservoir.monitoring.core.table	This package provides classes that are used for embedding Table data within a Measurement.
eu.reservoir.monitoring.distribution	This package provides classes that are used for distributing measurement data with the framework.

eu.reservoir.monitoring.distribution.multicast	This package provides classes for the implementation of the Data Plane that utilizes IP Multicast for transmission.
eu.reservoir.monitoring.distribution.udp	This package provides classes for the implementation of the Data Plane that utilizes UDP for transmission.
eu.reservoir.monitoring.im.dht	This package provides classes for the implementation of the Info Plane that utilizes a Distributed Hash Table (DHT)

Testing

To test the framework as it stands so far there are some demo classes. These are in the package: eu.reservoir.demo.

There are a few demos:

i) one that generates an emulated response time
ii) one that monitors the CPU usage, or network usage, or memory usage on a physical host

They provide working examples of how Probes can be written, and how the system can be glued together.

Demo i

Demo i is simple and just sends out an emulated response time from the ResponseTimeEmulator, which acts as a data producer. It uses a simple Probe called RandomProbe to generate these numbers. As a data consumer, there is a program called SimpleConsumer, which receives the measurements and displays them.

The first task is to execute the following on one machine:

```
java eu.reservoir.demo.SimpleConsumer
```

This starts the data consumer.

The second task is to execute the ResponseTimeEmulator on the same machine, like so:

```
java eu.reservoir.demo.ResponseTimeEmulator
```

You should see response times info arriving at the SimpleConsumer.

In demo i, IP multicast is used as the Data Plane for the distribution framework. The default address is 229.229.0.1/2299, but can be changed by passing a new multicast address and port as arguments to these programs. For example:

```
java eu.reservoir.demo.SimpleConsumer  234.1.2.4 7777
```

and:

```
java eu.reservoir.demo.ResponseTimeEmulator  234.1.2.4 7777
```

Using UDP
There is a version of this demo that uses UDP for the Data Plane rather than IP multicast. Try the following to test these:

```
java eu.reservoir.demo.SimpleConsumerUDP
```

and:

```
java eu.reservoir.demo.ResponseTimeEmulatorUDP
```

Looking at the code will show how the different Data Planes are set up.

Demo ii

This is a demo of utility usage monitoring on Linux boxes. It can use one of three Probes, (i) a Probe called CPUInfo, which reads data from /proc/stat and sends CPU usage data out, (ii) a Probe called NetInfo, which reads data from /proc/net/dev and sends network usage data out, or (iii) a Probe called MemoryInfo, which reads data from /proc/meminfo and sends memory usage data out. This demo uses a DataSource which is called BasicDataSource, from the package eu.reservoir.monitoring.appl. It has all the functionality require of a Data Source. The BasicDataSource is embedded in a program called HostMonitor, and it acts as the data producer by connecting to a Data Plane using an IP multicast implementation.

As a data consumer, there is a program called SimpleConsumer, which receives the measurements and displays them.

To execute this demo requires one SimpleConsumer and one or more HostMonitors running on various different machines.

The first task is to execute the following on one machine (say hostA):

```
java eu.reservoir.demo.SimpleConsumer
```

This starts the data consumer.

The second task is to execute the HostMonitor, and to decide which Probe to run. You run both processes on the same machine or on different machines.
To run the CPU Info Probe, do this:

```
java eu.reservoir.demo.HostMonitor -c
```

To run the NetInfo Probe, do this

```
java eu.reservoir.demo.HostMonitor -n
```

To run the MemoryInfo Probe, do this

```
java eu.reservoir.demo.HostMonitor -m
```

You can run HostMonitors on as many machines as required.

You should see cpu usage, network usage, or memory usage info arriving at the SimpleConsumer.

CLASSPATH

The classpath must include all of the classes in the packages mentioned here, plus the dht.jar file.