

SensorGrid Filters

Galip Aydin

03-30-2006

1. A Brief Introduction to Filters

SensorGrid architecture allows us to use the filters individually for simple data processing or chain several filters to solve more complex tasks. The NaradaBrokering middleware helps us connect filters with each other and separate the implementation from the messaging logic. This lets new applications to be seamlessly integrated with the system without having to interfere with the already deployed filters.

Similar to filters in electronics a real-time software filter takes an input message processes this message and outputs the result of this processing.



Figure 1. Simple filter

The input and output are connected to the messaging middleware for receiving and transmitting messages. However in some cases a filter may not broadcast outputs to the broker. For instance a Database Filter may receive the position messages from the broker and inserts these into a database which does not have to publish anything to the messaging middleware.

Following picture depicts use of NaradaBrokering and several filters to create a filter chain.

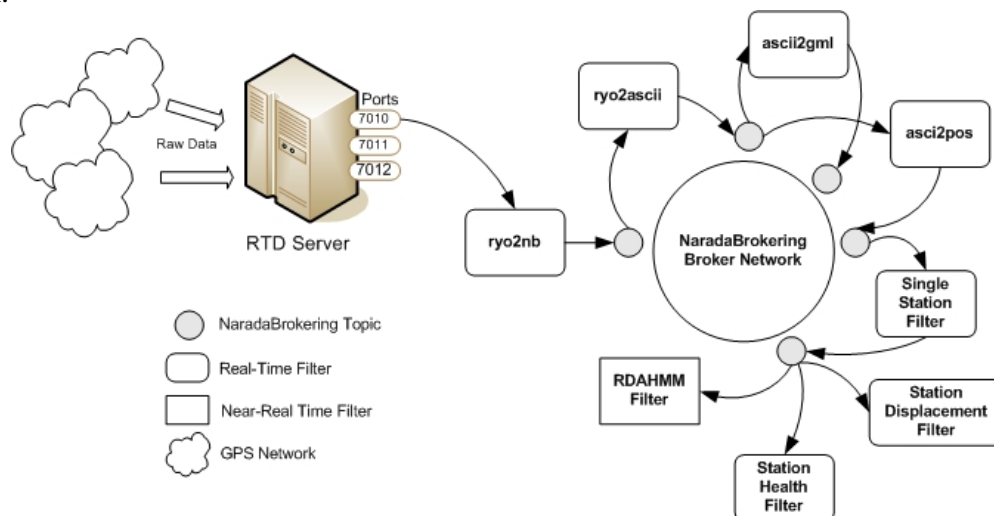
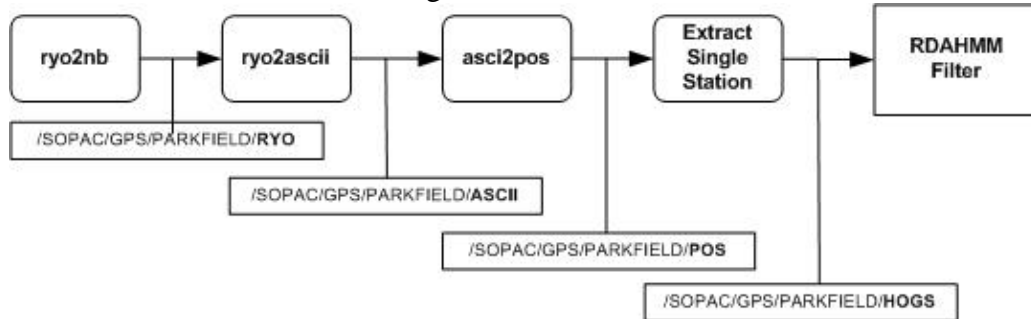


Figure 2. Use of NaradaBrokering to create filter chains.

In this scenario the RDAHMM application is invoked for certain number of GPS messages by the RDAHMM Filter. To do this the RDAHMM Filter needs to accumulate the position information for a single station. Following picture draws the arrangement of the filters for scenario described in Figure 2.



I.

Figure 3. NaradaBrokering topics can be arranged in a hierarchical order.

2. Installing the Filters

First NaradaBrokering server needs to be installed. The software is available for download through <http://www.naradabrokering.org>.

Current SensorGrid software is available at <http://www.crisisgrid.org/files/sensorgrid/>. The source files are under the *src* directory while the required libraries are under the *lib* directory. A *build.xml* file is also available for compiling the source files using *Apache Ant*. Ant build will compile the Java files and create a *classes.ant* directory, also it will create *sensorgrid.jar* archive for compiled classes and copy it under the lib directory.

The easiest way to run the filters is through command prompt (or shell) by appending the required libraries to the Java *CLASSPATH* and invoking with right parameters. I've included several *.bat* files for running filters under Windows environment and shell files for UNIX. These files are under the bin directory.

3. Testing the Filters

First download and extract the sensorgrid.rar (or sensorgrid.tar for Linux) archive. You will need WinRAR for extracting the rar archive (<http://www.rarlab.com>). This archive already contains the sensorgrid.jar file so you don't need to recompile the source code. You can test the filters by simply starting the batch files under the bin directory. Note that you will need a running NaradaBrokering server for this. Please replace the server address in these batch files (gf2.ucs.indiana.edu:3045) with your NB installation. You can keep the rest of the parameters as they are for now.

After installing the NB server and updating the batch files follow these steps for a simple demonstration of real-time GPS message decoding.

- 1- Run *Rtd2narada.bat*
- 2- Run *ryo2pos.bat*
- 3- Run *SimpleFilter.ba*

The first filter will open a socket connection to the RTD Server and start forwarding the raw RYO messages to the topic specified in the batch file (*/SOPAC/GPS/CRTN_01/POS*).

Second filter will subscribe to this topic, convert each RYO message to ASCII, simplify these messages by eliminating some fields, then publish these to another NB topic (*/SOPAC/GPS/CRTN_01/POS*)

The third filter subscribes to a topic specified from the command line and prints the messages it receives. You can see the output of the ryo2pos filter using this filter. The output looks like following.

This message contains the following fields:

Station Name	DSME
Date	2006-03-30
Time	03:33:49PM-EST
X	-2450647.8775
Y	-4758321.881800001
Z	3457381.3452
T	2.310741450434144
Latitude	33.036476806380996
Longitude	-117.24953435325 784
Height	56.13609835281054
End of Message Marker	EOF

You can use the ryo2ascii filter with right parameters to see the whole RYO messages decoded.

4. Integrating Filters with Other Applications

To understand how Filters operate let's look at the SimpleFilter.java file. This filter extends the Filter.java file and inherits the NB publisher and subscriber capabilities.

Receiving Messages from Subscribed Topics

Any class that inherits the Filter class must inherit the *onEvent* method. This method is inherited from NaradaBrokering and is used to receive the messages from the broker:

```
public void onEvent(NBEvent nbEvent) {
    .
    .
    if(nbEvent.getContentPayload()!=null)
        System.out.println(new String(nbEvent.getContentPayload()));
}
```

The `onEvent` method provides the received messages as byte arrays, we can get these by invoking the `getContentPayload()` method.

Publishing Messages to NB

The `Filter.java` class provides a method for publishing data. This method can be called from the inheriting class as in the `SimpleFilter.java`:

```
public void publishData(byte[] data){
    this.publishData(data);
}
```

Writing new Filters

The easiest way to write a new filter is to extend the `Filter.java` base class. This will provide you with the NB subscriber and publisher capabilities. You can simply set the right parameters to initialize the NB clients, process the messages and if necessary publish these messages back to NB.

To initialize the NB subscriber or publisher you need to call the `Filter.java` `initialize` method. This method expects a 5 member String Array as in the `SimpleFilter.java`:

```
String[] argsa = {commType,
                  nbHost,
                  nbPort,
                  subTopic,
                  pubTopic};
```

Here the first argument is the NB Communication Type. This can be `tcp`, `niotcp`, `udp` etc. (we can simply use `niotcp` for this context).

nbHost is the NB server address (i.e. `gf2.ucs.indiana.edu`)

nbPort is the port number for the specified `commType` (i.e. by default 3045 for `niotcp`, 5045 for `tcp`.)

subTopic is the NB topic the filter will subscribe to receive the messages.

pubTopic is the NB topic the filter will publish the generated messages.

In some cases there is no need to subscribe to both topics (see invocation parameters for Simple Filter and `rtd2narada` Filter). In such cases the topic should just be an empty string (“”). For instance the String array used to start the Simple Filter can be as following:

```
String[] argsa = {"niotcp",
                  "gf2.ucs.indiana.edu",
                  "3045",
                  "/SOPAC/GPS/CRTN_01/POS",
                  ""};
```