Overview of USAP Next Generation RVDAS

David Pablo Cohn / Valerie Warner
Scott Walker - Manager

Reader → Transform → Writer

Presented at RVTEC 2016 – 01 Nov 2016
Scripps Institution of Oceanography
An architecture, not a system

(systems change as requirements change; proper architecture lets you put together whatever system meets current requirements)

Reader → Transform → Writer

Read from serial port, prefix with timestamp and instrument id, write to file
Everyone’s needs are different **now**
Everyone’s needs will be different in 5 years

Solution: small set of Lego-like components that can be easily “snapped together” to create what you need

Read from serial port, prefix with timestamp and instrument id, write to file
Everyone’s needs are different **now**
Everyone’s needs will be different in 5 years

Solution: small set of Lego-like components that can be easily “snapped together” to create what you need
Readers, Transforms and Writers

Readers
- SerialReader
- NetworkReader
- FileReader

Transforms
- PrefixTransform
- TimestampTransform
- ParseXMLTransform
- ParseNMEATransform

Writers
- FileWriter
- NetworkWriter
- DatabaseWriter
- DisplayWriter
Dead Simple API
So it’s easy to create your own
Readers/Writers/Transforms as needed

```
reader = SerialReader(instrument='Knudsen')
transform = PrefixTransform(prefix='knud',
                            add_timestamp=True)
writer = FileWriter(logfile='/var/logs/NBP1511/knud')

while True:
    in_record = reader.read_record()
    out_record = transform.transform(in_record)
    writer.write_record(out_record)
```
(We’re using Python, by the way)
Snap pieces together to build what you need
Add new modules as we get new requirements

```
import xml.etree.ElementTree as ET

class DASRecord(models.Model):
    record_type = models.CharField(max_length=15)
    version = models.FloatField(blank=True)
    board = models.ForeignKey('Board', blank=True, null=True)
    board_index = models.IntegerField(blank=True, null=True)
    device = models.ForeignKey('Device', blank=True, null=True)
    serial = models.ForeignKey('Serial', blank=True, null=True)
    data = models.ForeignKey('Data', blank=True, null=True)


class OSUDASRecordWriter:
    def __init__(self):
        pass

    def write_record(self, record):
        tree = ET.parse_xml(record)
        record = DASRecord.from_xml(tree)
        record.save()
```
Add new modules as we get new requirements

NetworkReader ➔ OSUDASRecordWriter

```
<?xml version="1.0" encoding="UTF-8"?>
<OSU_DAS_Record Type="Data" Version="2.0">
    <SUDS_received>
        <Timestamp TimeZone="UTC">
            <Date Format="YYYY-MM-DD">2016-01-25</Date>
            <Time Format="HH:MM:SS">01:00:00</Time>
            <Count Units="Seconds" Epoch="1970-01-01 00:00:00 UTC">1453683600</Count>
        </Timestamp>
    </SUDS_received>
    ......
```

```
listener.py --read_network :6221 \ 
    --write_osu_dasrecords
```
Listeners – a snap-together tool

```bash
listener.py --read_serial Knudsen \ 
    --data_id knud \ 
    --logfile /data/logger/uw/NBP1511knud \ 
    --network :6221 \ 
    --database rvdas@bonnie:test
```
Chaining Listeners

listener.py --read_serial Knudsen \
   --data_id knud \
   --logfile /data/logger/uw/NBP1511knud \
   --network :6221

listener.py --read_logfile /data/logger/uw/NBP1511knud \
   --database rvdas@bonnie:nbp1511

listener.py --read_network :6221 \
   --display
Displays: Browser-based “pull” architecture

DisplayWriter
• Receives data records, parses out and caches values
• Serves XMLHttpRequests for updates (moving to websockets for efficiency)

Javascript DisplayWidget
• Renders dynamic Table/Dial/Chart
• Updates via websocket / XMLHttpRequests to DisplayWriter
DisplayWidgets are configure-drag-drop

```json
{
    'name': 'dial_wind',
    'widget': 'DASDialWidget',
    'angle': 'NBPSUSWindDir',
    'options': {
        'magnitude': 'NBPSUSWindSpd',
        'title': 'Stbd Relative Wind',
        'height': 300,
        'width': 300,
        'circle_color_range': ['green', 'orange', 'firebrick', 'purple'],
        'circle_color_domain': [2, 15, 30, 45],
    },
}
```
### Control Architecture – a Database Approach

<table>
<thead>
<tr>
<th>Id</th>
<th>Current state</th>
<th>Desired state</th>
<th>Sensor connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCOD</td>
<td>observed: PCOD running</td>
<td>desired: PCOD running</td>
<td>PCOD (serial)</td>
</tr>
<tr>
<td>adcp</td>
<td>observed: adcp not running</td>
<td>desired: adcp not running</td>
<td>adcp (serial /dev/ttyr05, 9600,</td>
</tr>
<tr>
<td>ctdd</td>
<td>observed: ctdd not running</td>
<td>desired: ctdd not running</td>
<td>ctdd (serial)</td>
</tr>
<tr>
<td>eng1</td>
<td>observed: eng1 running</td>
<td>desired: eng1 running</td>
<td>eng1 (serial)</td>
</tr>
<tr>
<td>flr1</td>
<td>observed: flr1 not running</td>
<td>desired: flr1 not running</td>
<td>flr1 (serial /dev/ttyr17, 19200,</td>
</tr>
<tr>
<td>gp02</td>
<td>observed: gp02 not running</td>
<td>desired: gp02 not running</td>
<td>gp02 (serial /dev/ttyr03, 4800,</td>
</tr>
<tr>
<td>grv1</td>
<td>observed: grv1 not running</td>
<td>desired: grv1 not running</td>
<td>grv1 (serial /dev/ttyr14, 9600,</td>
</tr>
<tr>
<td>gyr1</td>
<td>observed: gyr1 not running</td>
<td>desired: gyr1 not running</td>
<td>gyr1 (serial /dev/ttyr04, 4800,</td>
</tr>
<tr>
<td>hdas</td>
<td>observed: hdas not running</td>
<td>desired: hdas not running</td>
<td>hdas (serial /dev/ttyr27, 9600,</td>
</tr>
</tbody>
</table>
Control Architecture – a Database Approach

Django is “a database-driven web development framework”*

*translation: goop that lets us connect code to web pages and a database
Control Architecture – a Database Approach

LoggerManager

- Reads desired state from database
- Checks observed state from system
- Starts/stops/modifies processes to reconcile

```
manage_loggers.py -loggers pguv,rtmp,s330,knud \
--manage_interval 1
```
## Control Architecture – a Database Approach

**DAS Logger Dashboard**

Last update Mon Oct 10 01:41:54 2016

<table>
<thead>
<tr>
<th>Logger</th>
<th>Write logfile</th>
<th>Write network</th>
<th>Write database</th>
</tr>
</thead>
<tbody>
<tr>
<td>seap</td>
<td>Stop</td>
<td>/home/rvdas/logs/NBPsampleseap</td>
<td>Write</td>
</tr>
<tr>
<td>mwx1</td>
<td>Run</td>
<td>/home/rvdas/logs/NBPsamplemwx1</td>
<td>Write</td>
</tr>
<tr>
<td>tsg2</td>
<td>Run</td>
<td>/home/rvdas/logs/NBPsampleshtag2</td>
<td>Stop</td>
</tr>
<tr>
<td>gp02</td>
<td>Run</td>
<td>/home/rvdas/logs/NBPsamplegp02</td>
<td>Write</td>
</tr>
<tr>
<td>ffr1</td>
<td>Run</td>
<td>/home/rvdas/logs/NBPsamplesffr1</td>
<td>Write</td>
</tr>
<tr>
<td>hdas</td>
<td>Run</td>
<td>/home/rvdas/logs/NBPsamplehdas</td>
<td>Write</td>
</tr>
<tr>
<td>svp1</td>
<td>Run</td>
<td>/home/rvdas/logs/NBPsamplesvp1</td>
<td>Write</td>
</tr>
<tr>
<td>eng1</td>
<td>Run</td>
<td>/home/rvdas/logs/NBPsampleeng1</td>
<td>Write</td>
</tr>
<tr>
<td>kmd</td>
<td>Run</td>
<td>/home/rvdas/logs/NBPsampleskmd</td>
<td>Write</td>
</tr>
</tbody>
</table>
Control Architecture – a Database Approach

LoggerMonitor - watches database, LoggerManager, filesystem and processes; raises alarm if things look stale

```
monitor_loggers.py \
   --sleep_seconds 5 \
   --stale_interval 60 \
   --email rvdas_alerts@nbp.usap.gov
```
This is all early days – what’s next?

You have chronic mahjobbis crappus but that’s not why you puked.

Have you been exposed to any user interfaces designed by engineers?

Yes.

You have interface poisoning. You’ll be dead in a week.

Better Control UX
This is all early days – what’s next?

Better ways to compose Readers/Transforms/Writers

Maybe Scratch-like visual interface?
This is all early days – what’s next?

Drag-and-drop creation of entire displays
This is all early days – what’s next?

Beta deployment at sea
This is all early days – what’s next?

Getting other teams and vessels involved
  (come find us to talk)