Create your own Flow Node

David Rybach
rybach@i6.informatik.rwth-aachen.de

12.06.2008

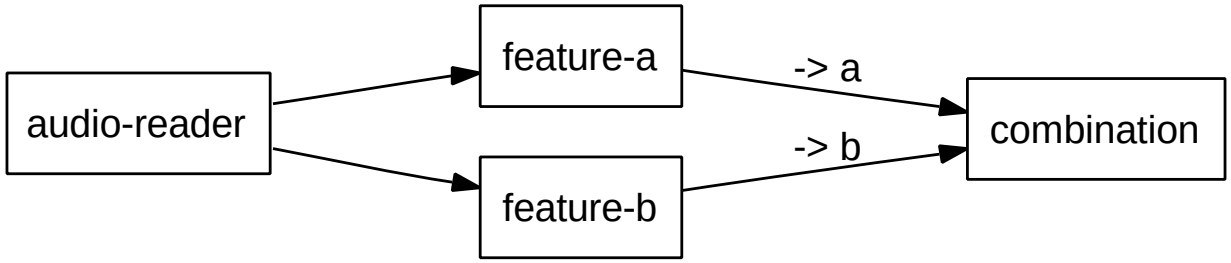
Human Language Technology and Pattern Recognition
Lehrstuhl für Informatik 6
Computer Science Department
RWTH Aachen University, Germany

Outline

- Flow
- Flow Networks
- Implementing new Nodes

Flow

- Flow is a general framework for data processing
- mainly used for feature extraction and alignment generation
- data manipulation (including loading / storing) is done by nodes
- a node has zero or more input and output ports
- nodes are connected by links from one port to another
- a set of nodes combined by links form a network
- a network can be used as node itself
- Flow networks are defined in XML documents
- a node is either a (C++) class or
  a network defined by another flow file
- abstract Example:



Nodes & Links

```
<node name="multiplication" filter="signal-matrix-multiplication-f32" file="$(lda-file)"/>
```

Node properties:

- `name`: name of the node (unique inside the network)
- `filter`: type of the node
  either a registered flow node (e.g. `signal-matrix-multiplication-f32`)
  or another flow network file
- parameters of the node, e.g `file`

```
<link from="sliding-window" to="multiplication"/>
```

Link properties:

- `from` output port of node sending data (syntax: `node-name:port-name`)
- `to` input port of node receiving data

Network Properties

```
<network name="my-network">
  <in  name="features"/>
  <out name="words"/>
  <out name="lattice"/>
  <param name="wer"/>
  <param name="rtf"/>
  <node name="does-it-all" filter="fictional-speech-recognizer-node"
        performance="$(wer)$" speed="$(rtf)" />
  <link from="my-network:features"    to="does-it-all:input"  />
  <link from="does-it-all:first-best" to="my-network:words"   />
  <link from="does-it-all:lattice"    to="my-network:lattice" />
</network>
```

```
  <!-- ... -->
  <node name="asr" filter="asr-node.flow" wer="0.0" rtf="0.01" />
  <link from="feature-extraction" to="asr"/>
  <!-- ... -->
```

- `name`: name of the network (optional)
- `in`: input ports (zero or more)
- `out`: output ports (zero or more)
- `param`: Parameter of the network. Value set by the calling network. Value available in variables `$(param-name)`

Parameters

- given in the node definition:

```
<node name="filterbank" filter="signal-filterbank"
      warping-function="mel" filter-width="268.258"/>
```

- given by the configuration mechanism

```
<node name="dump-node" filter="generic-dump"/>
```

```
[*.feature-extraction]
*.dump-node.file  = data/dump.xml
```

- dynamic parameter expansion:

```
<node name="audio-reader" filter="audio-input-file-$(audio-format)"
      file="$(input-file)" start-time="$(start-time)" end-time="$(end-time)"/>
```

- definition of network parameters

```
<param name="input-file"/>
<param name="start-time"/>
<param name="end-time"/>
```

Example

```
<network name="lda"> <!-- file: lda.flow -->
  <in name="in"/>
  <out name="out"/>

  <node name="sliding-window" filter="signal-vector-f32-sequence-concatenation"/>
  <link from="lda:in" to="sliding-window"/>
  <node name="multiplication" filter="signal-matrix-multiplication-f32" file="$(lda-file)"/>
  <link from="sliding-window" to="multiplication"/>
  <link from="multiplication" to="lda:out"/>
</network>
```

```
<network>
  <out name="features"/>
  <param name="input-file"/>
  <param name="start-time"/>
  <param name="end-time"/>

  <node name="audio-reader" filter="audio-input-file-$(audio-format)"
        file="$(input-file)" start-time="$(start-time)" end-time="$(end-time)"/>

  <node name="feature-a" filter="some-node"/>
  <link from="audio-reader" to="feature-a"/>

  <node name="lda" filter="lda.flow"/>
  <link from="feature-a" to="lda:in"/>

  <link from="lda:out" to="network:features"/>
</network>
```

Attributes

- Nodes communicate the information about the data sent by attributes
- Data type of the data packet has to set in the attribute `datatype`
- Can be observed / debugged by the `dump-attributes` channel

Example:

```
<flow-attributes>
   <flow-attribute name="datatype" value="vector-f32"/>
   <flow-attribute name="file" value="c002/118/0057.wav"/>
   <flow-attribute name="frame-shift" value="0.01"/>
   <flow-attribute name="sample-rate" value="1"/>
   <flow-attribute name="sample-size" value="16"/>
   <flow-attribute name="total-duration" value="30.96"/>
   <flow-attribute name="track-count" value="1"/>
</flow-attributes>
```

Implementing a new Node

- example: running sum node
- input: feature vectors (1 port)
- output: running sum of feature vectors (1 port)

$$f(t) = \sum_{t'=1}^{t} x_{t'}$$

Class Definition

```
class RunningSumNode : public Flow::Node
{
public:
    RunningSumNode(const Core::Configuration &c) :
        Core::Component(c), Flow::Node(c)
    {
        // ...
    }

    // name of the node used inside flow files (attribute 'filter')
    static std::string filterName() {
        return "vector-f32-running-sum";
    }

private:
    static const Core::ParameterFloat paramInitialValue;
    f32 initialValue_;
    bool needInitialization_;
    Flow::Vector<f32> sum_;
};
```

Ports

```
RunningSumNode(const Core::Configuration &c) :
    Core::Component(c), Flow::Node(c)
{
    addInput(0);    // add one input port
    addOutput(0);   // add one output port
}


/* called if a input port name is requested */
virtual Flow::PortId getInput(const std::string &name) {
    // node has only one input port -> map all names to 0
    return 0;
}

/* called if a output port name is requested */
virtual Flow::PortId getOutput(const std::string &name) {
    // node has only one output port -> map all names to 0
    return 0;
}

// better: derive from Flow::SleeveNode (1 input, 1 output)
```

Parameters

```
const Core::ParameterFloat RunningSumNode::paramInitialValue(
    "initial-value", "initial summation value", 0.0);

RunningSumNode::RunningSumNode(const Core::Configuration &c) :
    Core::Component(c), Flow::Node(c)
{
    // ...
    // parameters set via the configuration mechanism
    initialValue_ = paramInitialValue(config);
}

/**
 * setParameter is called each time a parameter is changed
 */
bool RunningSumNode::setParameter(const std::string &name, const std::string &value)
{
    if (paramInitialValue.match(name)) {
        initialValue_ = paramInitialValue(value);
        return true;
    } else {
        return Flow::Node::setParameter(name, value);
    }
}
```

Prepare

```
/**
 * static initialization of the node.
 * called each time an output attribute is requested,
 * i.e. before a new segment starts
 */
bool RunningSumNode::configure()
{
    Core::Ref<Flow::Attributes> a(new Flow::Attributes());

    // retrieve the attributes from the incoming link at port 0
    getInputAttributes(0, *a);

    // check the datatype of the incoming data
    if(!configureDatatype(a, Flow::Vector<f32>::type())) {
        error("wrong data type. expected '%s', got '%s'",
              Flow::Vector<f32>::type()->name().c_str(),
              a->get("datatype").c_str());
        return false;
    }

    // set a flag to reinitialize at segment start
    needInitialization_ = true;

    // leave the attributes unchanged, just pass it to the next node
    return putOutputAttributes(0, a);
}
```

Work!

```
/**
 * perform the data processing.
 * called each time a data packed is requested
 */
bool RunningSumNode::work(Flow::PortId p)
{
    Flow::DataPtr< Flow::Vector<f32> > in;
    // retrieve data from incoming port 0
    if (!getData(0, in)) {
        // no more input data
        // pass end of data token to the next node
        return putData(0, in.get());
    }

    if (needInitialization_) {
        needInitialization_ = false;
        sum_.resize( in->size() );
        std::fill(sum_->begin(), sum_->end(), initialValue_);
    }

    std::transform(in->begin(), in->end(), sum_.begin(), sum_.begin(), std::plus<f32>());

    // create a new data packed to be sent
    Flow::DataPtr< Flow::Vector<f32> > out( new Flow::Vector<f32>(sum_) );

    // sent data
    return putData(0, out.get());
}
```

Announce the new Node

```
class Application::Application()
{
    // ...
    registerFilter<RunningSumNode>();
}
```

Thank you for your attention

David Rybach

rybach@i6.informatik.rwth-aachen.de

http://www-i6.informatik.rwth-aachen.de/