Wrapping ITK Filters in **SCIRun** Modules

Ted Dustman[1]
CVRTI
University
of Utah

Darby J Van Uitert[2]
SCI Institute
University
of Utah

Yarden Livnat[3]
SCI Institute
University
of Utah

January 8, 2004

[1]dustman@cvrti.utah.edu
[2]darbyb@cs.utah.edu
[3]yarden@sci.utah.edu

Contents

1 Introduction

The National Library of Medicine Insight Segmentation and Registration Toolkit[1] (ITK) is open source software that complements **SCIRun** by providing image-processing capabilities, ranging from fundamental algorithms to advanced segmentation and registration tools. The Insight Tool Kit provides these capabilities as *filters*. *SCI* has developed a method of wrapping ITK filters in **SCIRun** modules.

If ITK is present, and **SCIRun** is appropriately configured, **SCIRun** builds a default set of ITK filter-based modules. Users can generate additional modules by writing XML descriptions. This chapter describes the process of wrapping ITK filters in **SCIRun** modules.

Before reading this chapter, users should be familiar with **SCIRun**, the Insight toolkit, and the use of generic programming techniques. Users must have access to the **SCIRun** source code tree and be familiar with its organization.

2 Aim

ITK filters are C++ objects (also known as *process objects*) that receive data on input ports and forward results on output ports. ITK filter objects provide functions for establishing connections to ports and setting filter properties.

SCIRun modules are C++ objects that receive data via input ports and forward results via output ports. **SCIRun** modules provide GUI-based dialogs for modifying module properties.

The aim of **SCIRun**/ITK integration is to map ITK filter data types, ports, properties, and property editing functions to **SCIRun** module data types, ports, properties, and property editing GUI dialogs.

3 Approach

ITK filters are mapped onto **SCIRun** modules via XML description files. Up to three XML files are required for each filter:

ITK Filter Description File
: Description of a filter. Contains no **SCIRun** specific information. Filter descriptions may exist in the ITK distribution.

SCIRun Filter Description File
: Contains **SCIRun** specific information and include a reference to an ITK filter description file and an optional reference to a GUI description file.

[1] http://www.itk.org

GUI Description File
: This file is optional. It describes a filter's GUI. A default GUI will be create if this file does not exist.

Software in the **SCIRun** build system automatically translates these XML files to **SCIRun** modules. XML files are described in Section 4, Section 5, and Section 6, respectively.

4 ITK Filter Description

The ITK filter description XML file describes a filter's inputs, outputs, parameters, parameter modifying member functions, templated data types, and other filter information. This file contains no **SCIRun** specific information.

ITK filter description files are located in directory `SCIRun/src/Packages/Insight/ITK`. ITK filter description files follow the naming convention:

`itk_`*`filter_name`*`.xml`

for example:

`itk_BinaryThresholdImageFilter.xml`

The following XML code illustrates the overall content of an ITK filter description file:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE filter SYSTEM "itk_filter.dtd">
<filter-itk name="filter_name">
   <description>
   ⋮
   </description>
   <templated>
   ⋮
   </templated>
   <datatypes>
   ⋮
   </datatypes>
   <inputs>
   ⋮
   </inputs>
   <outputs>
   ⋮
```

```
    </outputs>
    <parameters>
    ⋮
    </parameters>
    <includes>
    ⋮
    </includes>
  </filter-itk>
```

Each element is discussed below.

4.1 Element <filter-itk>

Element <filter-itk> is the top-level element. All other elements are enclosed in element <filter-itk>. The value of the required "name" attribute is the qualified C++ filter name.

For example:

```
<filter-itk name="itk::ReflectImageFilter">
  ⋮
</filter>
```

4.2 Element <description>

Element <description> should provide a short (several sentences) filter description.

For example:

```
<description>
  Implements a Reflection of an image along a selected
  direction. This class is parameterized over the type
  of the input image and the type of the output image.
</description>
```

4.3 Element <templated>

ITK filters are templated (parameterized) C++ classes and generally support one or two template parameters. The <templated> element describes a filter's template information. Specifically, <templated> contains elements that name a filter's template parameters and list suitable data types for filter instantiations.

```
<templated>
  <template>template_parameter_name1</template>
```

```
  <template>template_parameter_name2</template>
  ⋮
  <defaults>
    <default>data_type_for_parameter_name1</default>
    <default>data_type_for_parameter_name2</default>
    ⋮
  </defaults>
  <defaults>
    <default>data_type_for_parameter_name1</default>
    <default>data_type_for_parameter_name2</default>
    ⋮
  </defaults>
  ⋮
</templated>
```

Each <template> element declares and names a template parameter. Template parameter names may be used in subsequent <input> and <output> elements—see below.

Each <defaults> element corresponds to one filter class instantiation. Each <default> element names a type that corresponds (in order) with previous <template> elements.

For example, the following <templated> element:

```
  <templated>
    <template>InputImageType</template>
    <template>OutputImageType</template>
    <defaults>
       <default>itk::Image&lt;float, 2&gt;</default>
       <default>itk::Image&lt;float, 2&gt;</default>
    </defaults>
  </templated>
```

corresponds to filter class template:

```
  template <class InputImageType, class OutputImageType>
  class AnITKFilterClass...
```

and filter type instantiation:

```
  AnITKFilterClass<itk::Image<float,2>,itk::Image<float,2> >
```

Note that characters "<" and ">" cannot be typed literally into an XML file. The character entities '<' and '>', respectively, must be used.

A <template> element and its corresponding <default> element can describe a parameterized built-in value. In this case, a <template> element's "type" attribute specifies a C++ built-in type, and a corresponding <default> element specifies the type's value:

```
<templated>
  <template type="unsigned int">ImageDimension</template>
  ⋮
  <defaults>
    <default>2</default>
    ⋮
  </defaults>
</templated>
```

A **SCIRun** filter module knows all possible filter instantiations. When executed at runtime, a module uses the filter instantiation with data input and output types matching data types passing through the module's input and output ports.

4.4 Element <datatypes>

Element <datatypes> declares filter parameter data types that are not C++ built-in types. Names of declared data types are used in <parameter> elements that describe filter parameters (see Section 4.7). Element <datatypes> is optional, it is needed only when a parameter is not a C++ built-in type.

The structure of the <datatypes> element follows:

```
<datatypes>
  <data_type_element name="data_type_name">
  ⋮
  /<data_type_element>
  ⋮
</datatypes>
```

where *data_type_element* is the name of a *data type element*.

Each <*data_type_element*> describes the properties of one data type. Element <datatypes> enclose one or more data type elements.

At this time, element <array> is the only supported data type element. The array-like type described by element <array> is subject to the following conditions:

- It must support operator[]()
- Its array elements must be a built-in type

- It must support a member function that returns the array length

The structure of element <array> follows:

```
<array name="FilterType::data_type_name">
  <elem-type>built-in_type</elem-type>
  <size-call>member_function</size-call>
</array>
```

Attribute `name` specifies the name of an array-like data type. "FilterType" is a pseudo qualifier. It represents the full templated type of the current filter. `FilterType` is used in place of the full template name, for example, `itk::MeanImageFilter<InputType,OutputType>`. *Data_type_name* is the name of a type declared or typedef'ed in the filter's class declaration.

Element <elem-type> specifies the array's element type. *Built-in_type* is a C++ built-in type.

Element <size-call> specifies the name of a member function that returns the number of elements in the array.

An example <datatypes> element with an <array> sub-element follows:

```
<datatypes>
  <array name="FilterType::InputSizeType">
    <elem-type>int</elem-type>
    <size-call>GetSizeDimension</size-call>
  </array>
</datatypes>
```

`FilterType::InputSizeType` can be used in a subsequent <param> element.

4.5 Element <inputs>

The <inputs> element describes data types accepted by a filter's input ports, and member functions used to establish input port connections:

```
<inputs>
  <input name="input_name">
    <type>data_type_name</type>
    <call>member_function</call>
  </input>
  ⋮
</inputs>
```

For example:

```
<inputs>
  <input name="SourceImage">
    <type>InputImageType</type>
    <call>SetInput</call>
  </input>
</inputs>
```

Element <input> declares one filter input. The name attribute names an input. The value of name names the corresponding **SCIRun** module input port. The value of name must be unique among all inputs.

Element <type> specifies the data type accepted by a filter's input port. *Data_type_name* can be a name declared in a <template> element (as shown above), or a parameterized data type with parameter names declared in a <template> element. For example, the following template element associates type float with name ScalarType:

```
<templates>
  <template>ScalarType</template>
  <defaults>
    <default>float</default>
  </defaults>
</templates>
```

and ScalarType is used as a template parameter in the following <inputs> element:

```
<inputs>
  <input name="InputImage">
    <type>itk::watershed::SegmentTree&lt;ScalarType&gt;</type>
    <call>SetInputSegmentTree</call>
  </input>
</inputs>
```

Element <call> specifies a member function (commonly SetInput) that establishes an input port connection.

4.6 Element <outputs>

The <outputs> element describes data types transmitted on a filter's output ports, and the member functions used to establish output port connections:

```
<outputs>
  <output name="output_name">
    <type>data_type_name</type>
    <call>member_function</call>
```

```
    </output>
    ⋮
  </outputs>
```

For example:

```
<outputs>
  <output name="DestImage">
    <type>OutputImageType</type>
    <call>GetOutput</call>
  </output>
</outputs>
```

The `name` attribute names an output. The value of `name` names the corresponding **SCIRun** module output port. The value of `name` must be unique among all outputs.

Element <`type`> specifies the data type transmitted on a filter's output port. *Data_type_name* can be a name declared in a <`template`> element (as shown above), or a parameterized data type with parameter names declared in a <`template`> element.

Element <`call`> specifies the member function (commonly `GetOutput`) that establishes an output port connection. If <`call`> element's value is a member function returning a constant pointer, <`call`>'s `const` attribute must be set to the value "yes", for example:

```
<call const="yes">GetSpeedImage</call>
```

4.7 Element <parameters>

A filter's parameters determine its behavior. Filter member functions change parameter values. The <`parameters`> element describes a filter's parameters:

```
<parameters>
  <param>
    <name>parameter_name</name>
    <type>data_type</type>
    <call>member_function</call>
    <default>initial_value</default>
  </param>
  ⋮
</parameters>
```

Each <`param`> element describes one parameter with sub-elements <`name`>, <`type`>, <`call`>, and optional <`default`> providing the parameter's name, data type, corresponding member function, and optional initial value.

For example:

```
<parameters>
  <param>
    <name>upper_threshold</name>
    <type>float</type>
    <call>SetUpperThreshold</call>
    <default>1000.0</default>
  </param>
</parameters>
```

The content of element <type> is either a C++ built-in type or type declared in a <datatypes> element. Element <default> is optional.

If the ITK filter class macro itkBooleanMacro is used to create a boolean parameter, two <call> elements are needed; one for setting a true value, and one for setting a false value:

```
<call value="on">member_function_on</call>
<call value="off">member_function_off</call>
```

For example:

```
<param>
  <name>reverse_expansion_direction</name>
  <type>bool</type>
  <call value="on">ReverseExpansionDirectionOn</call>
  <call value="off">ReverseExpansionDirectionOff</call>
</param>
```

To modify the parameter, each parameter requires a GUI. If a GUI description file is not created, a default GUI is generated that associates a text entry widget with each filter parameter. Section 6 illustrates writing a GUI description for a filter module. When writing a GUI description file, note that the value of the <name> element above must match the value of a corresponding <param> element's name attribute in the GUI description file.

4.8 Element <includes>

Element <includes> lists ITK include files declaring filter classes and data types:

```
<includes>
  <file>include_file<file>
</includes>
```

For example:

```
<includes>
  <file>itkBinaryThreasholdImageFilter.h<file>
</includes>
```

5 SCIRun Filter Description

A **SCIRun** filter description file is associated with each ITK filter. The **SCIRun** filter description references the ITK filter description file, the optional GUI file, and contains **SCIRun** specific information.

SCIRun filter description files are located in directory:

`SCIRun/src/Packages/Insight/Dataflow/Modules/Filters/XML`

SCIRun filter description files follow the naming convention:

`sci_`*`filter_name`*`.xml`

for example:

`sci_BinaryThresholdImageFilter.xml`

The following XML code illustrates the overall structure of a sci-filter description file:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE filter SYSTEM "sci_filter.dtd">
<filter>
  <include href="file_path"/>
  ⋮
  <filter-sci name="SCIRun module name">
    <package>SCIRun_package_name</package>
    <category>SCIRun_category_name</category>
    <instantiations>
    ⋮
    </instantiations>
    <includes>
    ⋮
    </includes>
  </filter-sci>
</filter>
```

Each element is discussed below.

5.1 Element <filter>

Element <<filter>> is the top-level element. All other elements are enclosed in element <<filter>>:

```
<filter>
⋮
</filter>
```

5.2 Element <include>

Element <include> must be used one time to reference the ITK filter description file. It can be used a second time to reference an optional GUI description file.

The path to an ITK filter description file (or a dialog description file) is provided by element <include>'s "href" attribute. The file path is relative to the **SCIRun** Insight package directory (SCIRun/src/Packages/Insight). Note that <include> is an empty element, closed with characters />:

```
<include href="file_path"/>
```

For example:

```
<include href="ITK/itk_WatershedImageFilter.xml"/>
<include href="Dataflow/Modules/Filters/XML/gui_WatershedImageFilter.xml"/>
```

Two <include> element's are used above. The first references the ITK filter description file, the second references a GUI description file.

5.3 Element <filter-sci>

Element <filter-sci> provides module specific information.

```
<filter-sci name="module_name">
  <package>package_name</package>
  <category>category_name</category>
  <instantiations>
  ⋮
  </instantiations>
  <includes>
  ⋮
  </includes>
</filter-sci>
```

The value of the required name attribute determines the name of the **SCIRun** module produced from the filter, and the name of files generated.

For example:

```
<filter-sci name="WatershedImageFilter">
⋮
</filter-sci>
```

generates a module named `WatershedImageFilter`.

Elements <`package`> and <`category`> specify the package and category to which a module belongs:

```
<package>package_name</package>
<category>category_name</category>
```

For example:

```
<package>Insight</package>
<category>Filters</category>
```

Element <`instantiations`> can declare filter instantiations that replace those declared in an ITK filter description file (see Section 4.3):

```
<instantiations use-defaults="off">
  <instance>
    <type name="template_parameter_name1">
      <value>data_type_for_parameter_name1</value>
    </type>
  </instance>
  ⋮
</instantiations>
```

Attribute `use-defaults` is set to "off" to replace ITK filter instantiations with the instantiations specified. It is set to "on" to use ITK filter instantiations.

Each <`instance`> element declares one filter instantiation. Each <`type`> element corresponds to a <`template`> element in an ITK filter description. The value of <`type`>'s `name` attribute must match the content of a corresponding <`template`> element. Element <`value`> declares a data type associated with a template parameter.

For example:

```
<instance>
  <type name="InputImageType">
    <value>itk::Image&lt;float, 2&gt;</value>
  </type>
  <type name="OutputImageType">
    <value>itk::Image&lt;float, 2&gt;</value>
  </type>
 </instance>
 <instance">
  <type name="InputImageType">
```

```
      <value>itk::Image&lt;float, 3&gt;</value>
    </type>
    <type name="OutputImageType">
      <value>itk::Image&lt;float, 3&gt;</value>
    </type>
  </instance>
```

Element <includes> provides a list of include files needed by a module's implementation. Each <file> element provides the path name of one include file:

```
<includes>
  <file>include_file_path_name</file>
  ⋮
</includes>
```

For example:

```
<includes>
  <file>Packages/Insight/Dataflow/Ports/ITKDatatypePort.h</file>
</includes>
```

Include file paths are relative to the **SCIRun** `src` directory (`SCIRun/src`).

6 SCIRun GUI Description

A module GUI allows access to ITK filter parameters. By default, a GUI is created that associates a text entry widget with each filter parameter. By writing a GUI description file, the user can replace all or part of the default GUI.

SCIRun GUI description files are located in directory:

```
SCIRun/src/Packages/Insight/Dataflow/Modules/Filters/XML
```

SCIRun GUI filter description files follow the naming convention:

gui_*filter_name*.xml

for example:

```
gui_BinaryThresholdImageFilter.xml
```

The structure of the GUI description file consists of the top-level element <filter-gui> that encloses one or more <param> elements. Each <param> element associates a filter parameter with a GUI widget:

```
<?xml version="1.0"  encoding="iso-8859-1"?>
<!DOCTYPE filter-gui SYSTEM "gui_filter.dtd">
<filter-gui name="name">
  <param name="parameter_name">
    <widget_element>
    ⋮
    </widget_element>
    ⋮
  </param>
  ⋮
</filter-gui>
```

The value of a <param> element's name attribute must match the value of the corresponding <name> element in the filter description file (see Section 4.7)

One or more widgets can be specified in any order. Widgets are displayed top to bottom in the GUI window in the order specified in the ITK filter description file. Note that a text entry widget is generated for each parameter lacking a <param> element.

Widget_element is one of text-entry, checkbutton, scrollbar, radiobutton, or const. Notice that each *widget_element*, except <const>, must contain a <default> element.

A text entry widget allows the user to enter an arbitrary string of text, and is given an initial (default) value:

```
<text-entry>
  <default>default_string</default>
</text-entry>
```

A check button widget allows the user to select between true and false values. A check button widget is given a default value of 0 (false) or 1 (true):

```
<checkbutton>
  <default>0_or_1</default>
</checkbutton>
```

A scrollbar widget allows the user to select one value from a range of values. A scrollbar widget is given a min value, a max value, an optional scroll step value, and an initial value:

```
<scrollbar>
  <min>min_value</min>
  <max>max_value</max>
  <step>step_value</step>
  <default>default_value</default>
</scrollbar>
```

If the <step> element is not present, the scroll step value defaults to 1.

A radiobutton is a set of button widgets that allow the user to select one value from a set of values. Each value in a set is represented by a button widget. Each button has a label and a value. A <default> element provides <radiobutton>'s initial value. The initial value must be a value specified in a <button> element:

```
<radiobutton>
  <button>
    <label>label_string</label>
    <value>value</value>
  </button>
  ⋮
  <default>initial_value</default>
</radiobutton>
```

A ITK filter parameter may be set to a constant value using element <const>. No GUI widget will be generated for a constant parameter and the user cannot change the value of the parameter.

```
<const value="initial_value</const>
```

7 Putting It All Together

To build **SCIRun** modules from ITK filters:

1. Ensure that **SCIRun** is Configured with support for the Insight package. See the **SCIRun** Installation Guide for details.
2. Place ITK filter description files in `SCIRun/src/Packages/Insight/ITK`
3. Place **SCIRun** filter description files in `SCIRun/src/Packages/Insight/Dataflow/Modules/Filters/XML`
4. Place **SCIRun** GUI description files in `SCIRun/src/Packages/Insight/Dataflow/Modules/Filters/XML`
5. Invoke `gnumake` to build **SCIRun** (see the **SCIRun** Installation Guide for details)

After building **SCIRun**, ITK filter modules can be created by selecting their names from the **Filters** sub-menu of **SCIRun**'s **Insight** package menu.

8 Examples

Examples of ITK, **SCIRun**, and GUI filter description files in increasing order of complexity are:

- DiscreteGuassianImageFilter: Typical inputs and outputs and only 3 parameters. See files:

  ```
  itk_DiscreteGuassianImageFilter.xml
  sci_DiscreteGuassianImageFilter.xml
  gui_DiscreteGuassianImageFilter.xml
  ```

- ThresholdSegmentationLevelSetImageFilter: More parameters and a more complex GUI. See files:

  ```
  itk_ThresholdSegmentationLevelSetImageFilter.xml
  sci_ThresholdSegmentationLevelSetImageFilter.xml
  gui_ThresholdSegmentationLevelSetImageFilter.xml
  ```

- MeanImageFilter: GUI dependent on input image dimension. See files:

  ```
  itk_MeanImageFilter.xml
  sci_MeanImageFilter.xml
  gui_MeanImageFilter.xml
  ```

Example networks using the above modules are located in `SCIRun/src/Packages/Insight/nets`.