Refactoring MediaFilterManager for greater reuse and flexibility

Refactoring MediaFilterManager for greater reuse and flexibility

This tutorial will cover the refactoring of the MediaFilter framework for DSpace, freeing Mediafilter configuration to be reused on other parts of the codebase.

The Goals of this tutorial will exemplify the general process that should be applied in refactoring most any part of the codebase to use the DSpace Service Manager.

1. Analyze Old Code to Identify DOmain Model and Business Logic of the Application
2. Create Domain Level Objects and Spring Configuration
3. Abandon original Plugin or hardcoded configuration Approaches, replacing reading of configuration with behavior that is not bound to configuration.

What is the problem we can identify with MediaFilterManager

1. Configuration of the MediaFilters is bound to the execution of the Main method of the class
2. The List of FormatFilters and their configuration are both private, the configuration is held separately fromt eh objects themselves.
3. is hardcoded to a

Configuration of MediaFilters is bound to the execution of the main method:

```
            //retrieve list of all enabled media filter plugins!
            String enabledPlugins = ConfigurationManager.getProperty(MEDIA_FILTER_PLUGINS_KEY);
            filterNames = enabledPlugins.split(",\\s*");
        }

        //initialize an array of our enabled filters
        List<FormatFilter> filterList = new ArrayList<FormatFilter>();


        //set up each filter
        for(int i=0; i< filterNames.length; i++)
        {
            //get filter of this name & add to list of filters
            FormatFilter filter = (FormatFilter) PluginManager.getNamedPlugin(FormatFilter.class, filterNames
[i]);
            if(filter==null)
            {
                System.err.println("\nERROR: Unknown MediaFilter specified (either from command-line or in
dspace.cfg): '" + filterNames[i] + "'");
                System.exit(1);
            }
            else
            {
                filterList.add(filter);

                String filterClassName = filter.getClass().getName();

                String pluginName = null;

                //If this filter is a SelfNamedPlugin,
                //then the input formats it accepts may differ for
                //each "named" plugin that it defines.
                //So, we have to look for every key that fits the
                //following format: filter.<class-name>.<plugin-name>.inputFormats
                if( SelfNamedPlugin.class.isAssignableFrom(filter.getClass()) )
                {
                    //Get the plugin instance name for this class
                    pluginName = ((SelfNamedPlugin) filter).getPluginInstanceName();
                }


                //Retrieve our list of supported formats from dspace.cfg
                //For SelfNamedPlugins, format of key is:
                //  filter.<class-name>.<plugin-name>.inputFormats
                //For other MediaFilters, format of key is:
                //  filter.<class-name>.inputFormats
                String formats = ConfigurationManager.getProperty(
                    FILTER_PREFIX + "." + filterClassName +
                    (pluginName!=null ? "." + pluginName : "") +
                    "." + INPUT_FORMATS_SUFFIX);

                //add to internal map of filters to supported formats
                if (formats != null)
                {
                    //For SelfNamedPlugins, map key is:
                    //  <class-name><separator><plugin-name>
                    //For other MediaFilters, map key is just:
                    //  <class-name>
                    filterFormats.put(filterClassName +
                            (pluginName!=null ? FILTER_PLUGIN_SEPARATOR + pluginName : ""),
                            Arrays.asList(formats.split(",[\\s]*")));
                }
            }//end if filter!=null
        }//end for
```

Identify the structure of the hardcoded configuration that will need to be undone:

1. List of Classnames are in "filter.plugins" property in DSpace configuration

2. List of Input BitstreamFormats that are supported by the filter are configured as a lookup of individual filter property lists "filter.[~mdiggory:FILTER-PLUGIN-CLASS]inputFormats";

List of Format Strings is maintained separately from the fitleres themselves, thus we have two in memory "lookup lists" to deal with in mediafilter manager

```
plugin.named.org.dspace.app.mediafilter.FormatFilter = \
  org.dspace.app.mediafilter.PDFFilter = PDF Text Extractor, \
  org.dspace.app.mediafilter.HTMLFilter = HTML Text Extractor, \
  org.dspace.app.mediafilter.WordFilter = Word Text Extractor, \
  org.dspace.app.mediafilter.PowerPointFilter = PowerPoint Text Extractor, \
  org.dspace.app.mediafilter.JPEGFilter = JPEG Thumbnail, \
  org.dspace.app.mediafilter.BrandedPreviewJPEGFilter = Branded Preview JPEG

#Configure each filter's input format(s)
filter.org.dspace.app.mediafilter.PDFFilter.inputFormats = Adobe PDF
filter.org.dspace.app.mediafilter.HTMLFilter.inputFormats = HTML, Text
filter.org.dspace.app.mediafilter.WordFilter.inputFormats = Microsoft Word
filter.org.dspace.app.mediafilter.PowerPointFilter.inputFormats = Microsoft Powerpoint, Microsoft Powerpoint XML
filter.org.dspace.app.mediafilter.JPEGFilter.inputFormats = BMP, GIF, JPEG, image/png
filter.org.dspace.app.mediafilter.BrandedPreviewJPEGFilter.inputFormats = BMP, GIF, JPEG, image/png
```

Problems we may observe

- Filter formats are complex string requiring parsing by regular expression syntax.
- Names of BitstreamFormats are used as identifiers, but nothing in DSpace enforces that BitstreamFormat names need to be unique.
- Further configuration for each MediaFilter is not possible, if you want greater configurablity, you need to hardwire it into your MediaFilter implementation.
- MediaFilterManager is overly controlling of the FIltering Process, It would be better to let the Filters themselves have more decison making capabilities.

Goals of Spring Configuration

- Create independent configuration of each MediaFilter
- Allow Spring to Provide Autowiring capabilities to inject the Filters for us
- Allow the MediaFilterManager to be "instantiated for use outside of the main function.
- Remove complex naming and regular expression syntax.

Instantiating our new version of MediaFilterManager :

```
    <!-- Place all DSpace core service bean definitions below here  -->
        <bean class="org.dspace.apps.mediafilter.NewMediaFilterManager" autowire="byType"/>
```

Here we will allow our new mediafiltermanager to be autowired by type

FilterMediaManager Architecture

Buisness Logic:

Problem: Filtermedia attempts to regulate multiple levels of processing of the bitstream, leading to overly complex controller logic and the removal of flexibility from the Filter for processing the result.

Solution: Make a DelegateHandler that can be overridden by the application to create MediaFilters that do more generic tasks.

decisionmaking being remove

List of FormatFilters that can be applied to a DSpace Bitstream.

Format Filter

```
public interface FormatFilter
{
    /**
     * Get a filename for a newly created filtered bitstream
     *
     * @param sourceName name of source bitstream
     * @return filename generated by the filter - for example, document.pdf becomes document.pdf.txt
     */
    public String getFilteredName(String sourceName);

    /**
     * @return name of the bundle this filter will stick its generated Bitstreams
```

```
     */
    public String getBundleName();

    /**
     * @return name of the bitstream format (say "HTML" or "Microsoft Word")
     *         returned by this filter look in the bitstream format registry or
     *         mediafilter.cfg for valid format strings.
     */
    public String getFormatString();

    /**
     * @return string to describe the newly-generated Bitstream's - how it was produced is a good idea
     */
    public String getDescription();

    /**
     * @param source input stream
     *
     * @return result of filter's transformation, written out to a bitstream
     */
    public InputStream getDestinationStream(InputStream source)
            throws Exception;

    /**
     * Perform any pre-processing of the source bitstream *before* the actual
     * filtering takes place in MediaFilterManager.processBitstream().
     * <p>
     * Return true if pre-processing is successful (or no pre-processing
     * is necessary).  Return false if bitstream should be skipped
     * for any reason.
     *
     * @param c context
     * @param item item containing bitstream to process
     * @param source source bitstream to be processed
     *
     * @return true if bitstream processing should continue,
     *                         false if this bitstream should be skipped
     */
    public boolean preProcessBitstream(Context c, Item item, Bitstream source)
            throws Exception;

    /**
     * Perform any post-processing of the generated bitstream *after* this
     * filter has already been run.
     * <p>
     * Return true if pre-processing is successful (or no pre-processing
     * is necessary).  Return false if bitstream should be skipped
     * for some reason.
     *
     *
     * @param c context
     * @param item  item containing bitstream to process
     * @param generatedBitstream
     *            the bitstream which was generated by
     *            this filter.
     */
    public void postProcessBitstream(Context c, Item item, Bitstream generatedBitstream)
            throws Exception;
}
```

Interface to all MediaFilters to self register the formats that they supportInterface to allow filters to register the input formats they handle (useful for exposing underlying capabilities of libraries used)

```
public interface SelfRegisterInputFormats
{
    public String[] getInputMIMETypes();

    public String[] getInputDescriptions();

    public String[] getInputExtensions();
}
```

An Abstract MediaFilter class that supports od simple defaults

```
public abstract class MediaFilter implements FormatFilter
{
    public boolean preProcessBitstream(Context c, Item item, Bitstream source) throws Exception
    {
        return true;  //default to no pre-processing
    }

    public void postProcessBitstream(Context c, Item item, Bitstream generatedBitstream) throws Exception
    {
        //default to no post-processing necessary
    }
}
```

Business Logic

A number of configured media filters can be applied to any indvidual Bitstream within an Item to gerneate a resulting bitstream. Note the name of that resulting bitstream's name is generated and controlled byt he FormatFilter and that

MediaFilter Configuration

An array of FormatFilters that may be applied, a list of filterFormats they apply to an a list of Communities or Collections to ignore (skipList)

```

    private static FormatFilter[] filterClasses = null;

    private static Map<String, List<String>> filterFormats = new HashMap<String, List<String>>();

    private static List<String> skipList = null; //list of identifiers to skip during processing


```

Configuration Options

Business Logic (Iterator over DSpace Object Model)

Ways we can clean up the filtering process:

For Loops:

For Loops are strong candidates for refactoring the codebase, the contents of For Loops generally can be copied to new Classes and overriden by the application if neccessary leading to a simpler controller and greater capability to change the codebase and change-up processing.

Reviewing MediaFilterManager we can see a number of ways to easlit cleanup the codebase here. We will go through the codebase and identify the parts that are configured via the existing dspace.cfg and we will replace those:

```
//key (in dspace.cfg) which lists all enabled filters by name
public static final String MEDIA_FILTER_PLUGINS_KEY = "filter.plugins";

//prefix (in dspace.cfg) for all filter properties
public static final String FILTER_PREFIX = "filter";

//suffix (in dspace.cfg) for input formats supported by each filter
public static final String INPUT_FORMATS_SUFFIX = "inputFormats";

...

//separator in filterFormats Map between a filter class name and a plugin name,
//for MediaFilters which extend SelfNamedPlugin (\034 is "file separator" char)
public static final String FILTER_PLUGIN_SEPARATOR = "\034";
```