Shibboleth Metadata Aggregator (MDA) Evaluation

(Chris Giordano, MOREnet, 03-Apr-2013) w/ replies from Ian Young, MDA Project Lead

Having spent some time over the last couple of days evaluating the Metadata Aggregator tool I believe I've learned enough to share my first impression of the tool. In a nutshell, it appears to work fine and I have no doubt it could be used as part of a pilot. For my testing, I used two copies of our SP metadata as input and tried configuring the tool to process those into a single metadata file based on the example configuration options documented on the project's website.

Briefly, the operation of this tool is based on the concept of a pipeline built up of processing stages. The first stage, reading input files, passes onto successive stages and this process continues until it reaches the final stage responsible for the output of a transformed metadata file. So, using the tool involves defining named pipelines and stages in an XML configuration file and then running a command line tool with arguments that inform the program which named pipeline to process. There's even support for defining multiple pipelines that can be merged together into the final output. With just a little bit of experience using the tool I quickly found myself wanting this capability, for example, to process different metadata files with different rule sets.

Depending on the person's background the configuration process and usage may feel daunting. With regard to the configuration process this is mostly due to the lack of documentation. That is, it's largely straightforward to edit the XML configuration file and declare pipelines and stages but to do so may also require knowing the underlying behavior of the stage being defined. For example, the sample configurations describe how to read in a single metadata file but my first task was to define an input stage for reading in multiple files. I had no reference for doing this. I started with reading Spring documentation, the framework this tool is based on, but ultimately I checked out the Subversion source code and read the Java source to arrive at how to do this. Again, I was back to reading the source code when I needed to understand how to define and configure a filter to white list a set of contact types.

With regard to the actual usage of the tool, I've only briefly looked at a couple of the stages available. But, there are stages for inserting and stripping out elements; filtering and white listing elements (i.e. contact types); and for signing and validation. It would appear virtually everything is in place to satisfy even the most specific needs. But, in the event a custom stage needed to be written, I don't perceive it would be a huge obstacle to overcome. it would require writing Java programming to create a stage having the business logic to perform the task as well as knowing something about the Spring Java Framework. Depending on a person's background, a commitment to Java programming and learning Spring would be a consideration.

With regard to using this tool as part of a pilot, I would recommend defining a use case consisting of just two or three metadata files having sample data to process. Identify what the transformation needs to do for producing the final output file and create that configuration in the tool. Perform everything that can be done using the existing, available stages. This will fully

test all of the tool's capabilities within the context of the pilot as well as ensure the stages operating as expected. It's very possible all the stages needed already exist but, if this is not the case the developers could then be contacted for guidance, if necessary. Either way, beyond a little extra effort to get it configured I have not found anything that would give me pause from using this tool in a production environment today.

For what it's worth, below are some miscellaneous notes I'd like to share.

1) Here is the website to the Metadata Aggregator tool. This page also documents the conceptual overview of the tool:

https://wiki.shibboleth.net/confluence/display/MA1/1+Introduction

2) Here is the web page describing the existing, available stages that can be used for pipeline processing:

https://wiki.shibboleth.net/confluence/display/MA1/3.1+General+Configuration

3) Something I just discovered while looking at this tool is a metadata filter that can be defined in the IdP. Something that might work well alongside the Aggregator tool.

https://wiki.shibboleth.net/confluence/display/SHIB2/IdPMetadataProvider

4) The current CLI release is v0.7 which I used for my testing. Though the introductory documentation refers to both a command line interface (CLI) as well as a RESTful web service, the CLI is what's available as a downloadable distribution from the website. I also checked out the source from the subversion repository and see there is code to support the web service though I didn't see documentation for it's use. I don't know how far along that effort is but, in general, the project appears active. The recent log entry below indicates the developers are preparing for their next development cycle.

```
------------------------------------------------------------------------
r231 | iay | 2013-03-16 05:31:07 -0500 (Sat, 16 Mar 2013) | 1 line

Fix POM versions for 0.8 development.
```

5) Here is my configuration for my white list of contact types. This configuration removes all contact types other than: 'support'. It does this to all of the input files. Below the configuration is the Java code snippet I referenced (directly from the class declaration). I needed to read the code in order to know the property name (designatedTypes) to be able to write this configuration.

```
<bean id="removeInvalidContactPerson"
class="net.shibboleth.metadata.dom.saml.ContactPersonFilterStage">
  <property name="id" value="removeInvalidContactPerson"/>
  <property name="designatedTypes">
  <list>
    <value>support</value>
  </list>
  </property>
</bean>
```

Java snippet:

```
/** Allowed contact person types. */
private Set<String> allowedTypes = ImmutableSet.copyOf(new String[]
{TECHNICAL, SUPPORT, ADMINISTRATIVE, BILLING, OTHER,});

/** Person types which are white/black listed depending on the value of
{@link #whitelistingTypes}. */
private Set<String> designatedTypes = ImmutableSet.copyOf(allowedTypes);
```

```
6) Here is my configuration for reading in multiple SP metadata files.  If
the <constructor-arg> value is a directory the tool will read in all files in
that directory.  Again, I needed to read the code in order understand my
options for reading in multiple source files.  I like this option for it's
flexibility.  A potentially useful technique would be to point the tool to a
directory and symlink in a subset of metadata files for processing.  Metadata
files could be processed in different combinations easily.
```

```
<!-- First, we define the stages for our pipeline -->
<bean id="source"
class="net.shibboleth.metadata.dom.DomFilesystemSourceStage">
  <property name="id" value="source"/>
  <property name="parserPool">
    <bean class="net.shibboleth.utilities.java.support.xml.BasicParserPool"
init-method="initialize"/>
  </property>
  <property name="source">
    <bean class="java.io.File">
      <constructor-arg value="/usr/local/aggregator-cli-0.7.0/metadata"/>
    </bean>
  </property>
</bean>
```

7) Sometimes, I've noticed things from the source of mild interest that reflect the current state of the tool. For example, here's a comment in the source for the command line tool itself. My experience with the tool is that it works fine. The reference to the tool's robustness probably refers to the Java stack trace output that gets displayed when it encounters an error. Something less than friendly like a stack trace would probably alarm some end users.

```
/**
 * A simple driver for the metadata aggregator.
 *
 * This CLI is not terribly robust nor does it really offer much in the way
of features. It's mostly meant for testing purposes and will be replaced
before the 1.0 release of the software.
 */
```

```
Let me know if there's anything further I can assist with.

Regards,

Chris Giordano, MOREnet
```

**** (See Ian Young's responses to Chris's questions below)…*

Ian Young *is the Project Lead for the Shibboleth Metadata Aggregator (MDA) project. The following are his comments to Chris's write-up of the MDA.*

(Response 1 – 04Apr2014):

It's obviously always interesting to see what someone new to the tool runs into. I've had only a quick read through so far, so I may have additional comments later, but for now here are some things that strike me:

**** Chris (or anyone else) should feel free to post questions in the Shib users list if they get stuck. There are a couple of other serious users out there, and I monitor it myself. I'm probably the heaviest user out there (with my UK federation hat on) as well as the project lead.***

**** Although Chris is right that in order to extend the system you *can* use Java, I find a lot of the sort of things I've needed to do in the UK can actually be coded up as XSLT transforms. Once you understand basic XSLT (and I grant that not everyone does!) then it's often much quicker to just whip up an XSLT stage to do something than it would be to code it up in Java. The expectation was always that people would do a lot of that, but that we'd then re-code popular stages as Java and make them available as new pre-defined stages in later releases.***

Hope that helps,

-- Ian

(Response 2 – 10Apr2014):

Just a couple of additional notes to supplement my previous comments now that I've had a chance to read through in more detail.

> Depending on the person's background the configuration process and usage may feel daunting. With regard to the configuration process this is mostly due to the lack of documentation. That is, it's largely straightforward to edit the XML configuration file and declare pipelines and stages but to do so may also require knowing the underlying behavior of the stage being defined. For example, the sample configurations describe how to read in a single metadata file but my first task was to define an input stage for reading in multiple files. I had no reference for doing this.

I agree that we have a bit of a "learning cliff" for new users. The intention was that the "General Configuration" section of the wiki would be a reference for all stages:

https://wiki.shibboleth.net/confluence/display/MA1/3.1+General+Configuration

Each of the stage definitions there has a link to Javadoc for the stage, but even that won't make much sense unless people already know how to translate what they learn there into Spring bean definitions.

I'd welcome any specific suggestions for improvement in this area. So far, I think the thing that has helped people understand the operation of configuration most have been additional examples; I've made the UK metadata machinery (which is a very large example) available to a couple of people to see what happened. It's probably a bit overwhelming for beginners, unfortunately.

At a conceptual level, this article may help:

http://iay.org.uk/blog/2012/08/uk-federation-metadata-aggregation

> With regard to the actual usage of the tool, I've only briefly looked at a couple of the stages available. But, there are stages for inserting and stripping out elements; filtering and white listing elements (i.e. contact types); and for signing and validation. It would appear virtually everything is in place to satisfy even the most specific needs. But, in the event a custom stage needed to be written, I don't perceive it would be a huge obstacle to overcome. it would require writing Java programming to create a stage having the business logic to perform the task as well as knowing something about the Spring Java Framework. Depending on a person's background, a commitment to Java programming and learning Spring would be a consideration.

As I mentioned in my previous comments, it turns out that a lot of the simpler stuff can be done in XSLT rather than going all the way down the road to Java. I use a lot of XSLT in the UK deployment, for example.

> Either way, beyond a little extra effort to get it configured I have not found anything that would give me pause from using this tool in a production environment today.

You'll gather from what I've said above that the UK federation uses the current 0.7 release in production. Although the MDA code base is not "version 1.0" in terms of function, what is there is very stable. The only thing I'd want to warn you about is that as a pre-1.0, the API is not stable. That means that 0.8, 0.9, 1.0 may change the names of beans or change the properties of those beans, requiring configurations coded for 0.7 to be manually updated for later releases.

> For what it's worth, below are some miscellaneous notes I'd like to share.
>
> 1) Here is the website to the Metadata Aggregator tool. This page also documents the conceptual overview of the tool:
>
> https://wiki.shibboleth.net/confluence/display/MA1/1+Introduction
>
> 2) Here is the web page describing the existing, available stages that can be used for pipeline

processing:
>
> https://wiki.shibboleth.net/confluence/display/MA1/3.1+General+Configuration

Another useful place to look for pre-1.0 software would be the JIRA tracker where we keep bug lists and feature requests:

https://issues.shibboleth.net/jira/browse/MDA

> I also checked out the source from the subversion repository and see there is code to support the web service though I didn't see documentation for it's use. I don't know how far along that effort is but, in general, the project appears active.

The web service is planned, but for a later release and almost certainly not 0.8.

> The recent log entry below indicates the developers are preparing for their next development cycle.

True, but that's largely administrative; there are no timescales for an 0.8 release as yet, but as we are occasionally doing things like bug fixes our processes include that work as part of 0.8.

> 7) Sometimes, I've noticed things from the source of mild interest that reflect the current state of the tool. For example, here's a comment in the source for the command line tool itself. My experience with the tool is that it works fine. The reference to the tool's robustness probably refers to the Java stack trace output that gets displayed when it encounters an error. Something less than friendly like a stack trace would probably alarm some end users.

I agree, and there's a JIRA case for that:

https://issues.shibboleth.net/jira/browse/MDA-66

> Let me know if there's anything further I can assist with.

Ditto :-)

- Ian