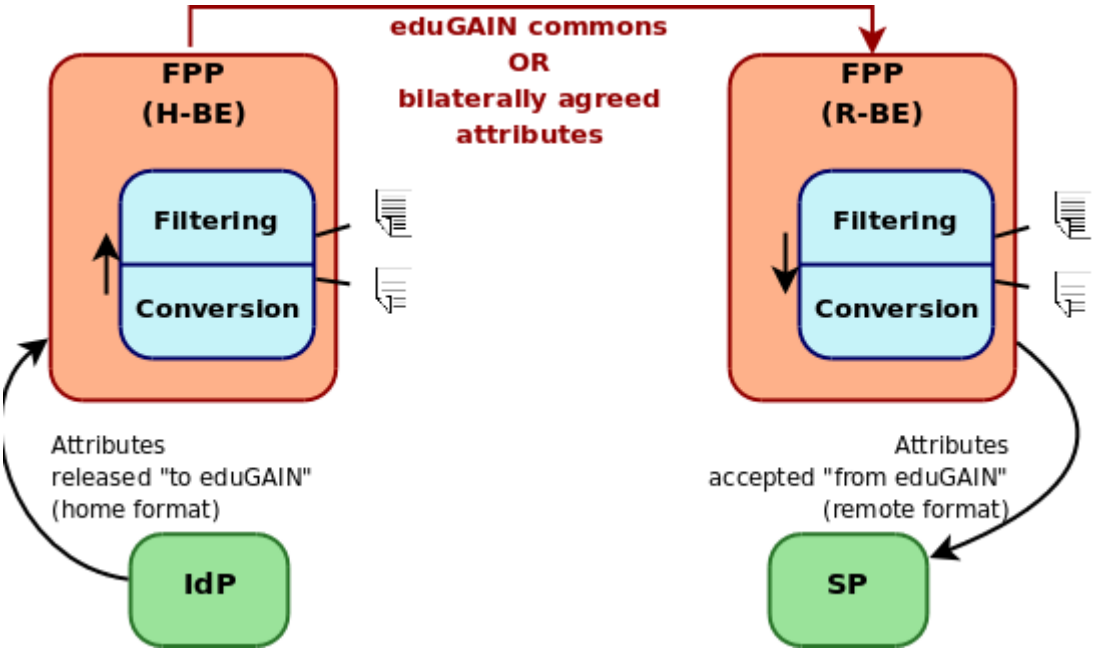
Attribute Conversion for eduGAIN

JRA5 Attribute Conversion allows a Bridging Element administrator to define rules to transform attributes being released or received. The same logic can work in both Home and Remote Bridging Elements.

Introduction



Attributes are travelling on the wire in eduGAIN-defined format, ie. SAML. Naming attributes and defining their contents might be a standardization task of eduGAIN operators; however it should be possible for federations to agree on custom set of attributes *beyond* "*eduGAIN commons*".

Attribute Conversion only adds attributes (or values) to the attribute set; use Attribute Filtering for filtering out unnecessary attributes. It also means that if no rules match an attribute, then it will go to the filter unmodified - so conversion works with a **default by-pass policy**.

Attribute conversion rule concepts

Most of the rules are based on standard regular expressions and Unified Expression Language.

Each rule works on the actual attribute set which is not necessarily the initial set, as each rule can alter the set (ie. by changing values or names, adding new attributes to the set). This means that the **order of the rules is important**.

Every rule consists of two parts: condition and action. The condition element is used to determine whether this particular rule is to be processed or not. Thus, the rule action is only processed when all the conditions are met (a rule without any conditions is processed by default).

The condition engine now only supports regular expression -based matching rules. There are three types of matching rules

- local federation peer's identifier (LocalProviderMatch)
- remote federation peer's identifier (RemoteProviderMatch)
- attribute values (AttributeMatch)

The rule's action is to create new attributes (or to modify existing ones). Please refer to the detailed BasicRule, MergeRule, SplitRule documentation below.

Attribute conversion rule types

BasicRule

The Basic rule is the simplest attribute conversion rule type. It can create one attribute and optionally use one attribute and regular expressions to transform attribute values.

Basic Rule can create static attributes. You can achieve this by omitting the Condition node. The `replaceValues` parameter is true by default, so if you want to append values to (probably) existing attributes, you must declare it using `replaceValues="false"`. Also note that you can use multiple AttributeValue nodes.

```
<BasicRule>
  <Description>Create static attribute (or append to existing if attribute with this name
already exists)</Description>
  <Attribute attributeName="eduPersonScopedAffiliation" replaceValues="false">
    <AttributeValue>staff@niif.hu</AttributeValue>
    <AttributeValue>member@href.hu</AttributeValue>
  </Attribute>
</BasicRule>
```

The next rule is using remote provider matching to determine whether the remote side has an identifier of 'urn:geant:edugain:be:' and any Hungarian domain appended to it.

```
<BasicRule>
  <Description>Create static attribute for some remote providers</Description>
  <Condition>
    <RemoteProviderMatch>^urn:geant:edugain:be:[^:]+\.hu$</RemoteProviderMatch>
  </Condition>
  <Attribute attributeName="homeOrganization">
    <AttributeValue>niif.hu</AttributeValue>
  </Attribute>
</BasicRule>
```

This example shows how to rename an attribute without converting its values. Note that you must use `AttributeMatch` without regular expressions to achieve this.

```
<BasicRule>
  <Description>Rename attribute uid to edupersonPrincipalName</Description>
  <Condition>
    <AttributeMatch attributeName="uid"/>
  </Condition>
  <Attribute attributeName="edupersonPrincipalName">
    <AttributeValue>${uid}</AttributeValue>
  </Attribute>
</BasicRule>
```

The next example demonstrates the use of regular expression matching groups.

```
<BasicRule>
  <Decription>Transform 'o=org,c=country'-style OrgDN to DNS-based
homeOrganization</Decription>
  <Condition>
    <AttributeMatch attributeName="edupersonOrgDN" id="regex">o=(.*),c=(.*)</AttributeMatch>
  </Condition>
  <Attribute attributeName="homeOrganization">
    <AttributeValue>${regex[1]}.${regex[2]}</AttributeValue>
  </Attribute>
</BasicRule>
```

This last example needs some more explanation. When you want to reference the regular expression matching groups (enclosed by parentheses), you must define the reference name with the 'id' parameter of `AttributeMatch`. Then, use `${id[0]` to refer to the whole regular expression match (ie. the whole attribute value), and `${id[0]}` to refer to the Nth. matching group of the regular expression.

Info

You cannot reference regular expressions from another rule.

MergeRule

The merge rule can merge two or more attributes into one. The attributes whose values you want to merge is declared using the InputAttribute node. You can also use the condition node, but only with `RemoteProviderMatch` and `LocalProviderMatch` (`AttributeMatch` is ignored).

This example shows how to combine two attribute values:

```
<MergeRule>
  <Description>Merges the uid and homeOrganization to edupersonPrincipalName</Description>
  <InputAttribute attributeName="homeOrganization" />
  <InputAttribute attributeName="uid" />
  <Attribute attributeName="edupersonPrincipalName" replaceValues="true">
    <AttributeValue>${uid}@${homeOrganization}</AttributeValue>
  </Attribute>
</MergeRule>
```

You can also use regular expressions, as with **BasicRule**.

SplitRule

The **SplitRule** is very similar to the **MergeRule**, the only difference is that the **SplitRule** contains one `InputAttribute` and more `Attribute` nodes.

```
<SplitRule>
  <Description>Split the edupersonScopedAffiliation to edupersonAffiliation and
homeOrganization</Description>
  <InputAttribute attributeName="edupersonScopedAffiliation" id="scopedAffiliation"
>^([^@]+)@(.+)$</InputAttribute>
  <Attribute attributeName="edupersonAffiliation">
    <AttributeValue>${scopedAffiliation[1]}</AttributeValue>
  </Attribute>
  <Attribute attributeName="homeOrganization">
    <AttributeValue>${scopedAffiliation[2]}</AttributeValue>
  </Attribute>
</SplitRule>
```

CustomRule

If you need to create new attributes from program (eg. appending generated identifiers), you can use the **CustomRule** type.

```
<CustomRule className="org.test.MyCustomRuleImpl">
  <Configuration>
    <!-- any xml here -->
  </Configuration>
</CustomRule>
```

CustomRule class must implement the `net.geant.edugain.attributes.rules.Rule` interface, configuration can be read with the DOM API. Please refer to the Attribute Converter JavaDOC, and see the test package as it contains a sample implementation.

Negating matches

If your federation has *optional* attributes then sometimes it is desirable to process rules **only if a particular attribute does not exist.** Therefore it is possible to append a `negate` boolean attribute (setting it to **true**) to the `<*Match>` nodes (inside the `<Condition>` element) to revert the match. It means that the rule is only processed if there is no match for the given value.

The following example creates `preferredLanguage` only if it is not set by the IdP (or by the peer's home bridging element):

```
<BasicRule>
  <Decription>Create preferredLanguage only if source has not supplied it</Decription>
  <Condition>
    <AttributeMatch attributeName="preferredLanguage" negate="true"/>
  </Condition>
  <Attribute attributeName="preferredLanguage">
    <AttributeValue>hu, en-gb;q=0.8, en;q=0.7</AttributeValue>
  </Attribute>
</BasicRule>
```

Using attribute name mapper

For interoperability, SAML AttributeStatement carries attribute names with URN-style attribute naming scheme. For example, the 'mail' logical attribute name can be named as `'urn:mace:dir:attribute-def:mail'`, or `'urn:oid:0.9.2342.19200300.100.1.3'`. Shibboleth2 further encourages federations to use the latter form (ie. the LDAP oid).

The eduGAIN Attribute Converter library comes with an attribute name mapping subsystem. With the help of the attribute name mapper, system administrators can **write the attribute converter configuration independently of the currently used attribute name format in AttributeStatement**.

Attribute name mapper concepts

As the attribute conversion sits between two federations (and probably two attribute naming schemes), there are two types of physical attributes: the 'input' and 'output' attributes. Note that these notation is different in Home and Remote BEs: Home BE releases attributes to the eduGAIN federation, Remote BE releases attributes to the local federation. So the eduGAIN format is the **'output' attribute format of the Home BE, and the 'input' format of the Remote BE.**

The following example shows the difference between logical and physical attribute names.

Physical input attribute name	Logical attribute name	Physical output attribute name
urn:mace:dir:attribute-def:mail	**mail** {: rowspan="2"}	urn:mace:dir:attribute-def:mail {: rowspan="2"}
urn:oid:0.9.2342.19200300.100.1.3	&#8288 {: style="padding:0"}	&#8288 {: style="padding:0"}

Configuration of the attribute name mapper

The configuration xml of the attribute name mapper consists of one or more AttributeDefinition nodes. Each AttributeDefinition node specifies one logical attribute name (called 'id') and one physical AttributeName - which is the physical 'output' attribute name format. You can also specify AttributeNamespace (for SAML1.0).

The 'input' attribute names are configured using the 'Attribute' node. Note that the output name is also interpreted as input name, so it is not necessary to declare it twice.

```
<AttributeDefinition
 Id="mail"
 AttributeName="urn:mace:dir:attribute-def:mail">
  <Attribute AttributeName="urn:oid:0.9.2342.19200300.100.1.3" />
</AttributeDefinition>
```

Using logical attribute names

When attribute name is referenced in a conversion rule, attribute mapper tries to interpret it as logical attribute name. Note that the logical attribute names are case-insensitive.

Using SAML1.1 Namespaces

EduGAIN Bridging Elements use SAML1.1 as communication protocol. SAML1.1 forces the use of Attribute NameSpace. You can specify the exact namespace in the attribute name mapping configuration using the 'AttributeNameSpace' attribute (with both the AttributeDefinition and Attribute nodes). When you use AttributeNameSpace, the input matching considers two attributes with the same name but different namespace different. The output attributes will contain the namespace information.

Attribute Filtering

Concepts

At Home BE, Filtering normally gets its incoming attribute set from Conversion; at Remote BE, it gets incoming attributes from the other bridging element.

From a technical viewpoint, Attribute Filtering is just a Rule extension to Conversion, so you can use most of the features of Converter, especially regular expressions and matching conditions. One major difference is that **only explicitly allowed attributes can pass through**, so you have to list all the attributes that you want to support in eduGAIN.

Filter uses name mappers in the same way as Converter. So you should define your attributes there before you start using 'friendly' attribute names here.

Allowing and denying attributes

Three main rules of the filtering:

1. Default action is to deny ALL attributes.
2. You can allow/deny whole attributes or specific values of the attributes.
3. The first rule decides. If you allowed something, you can not deny it later and vice versa. So start with the special rules and leave the generic rules to the end.

You can allow an attribute by using `<AllowAttribute>` element and deny it with `<DenyAttribute>`. Each element can optionally have child elements `<AttributeValue>`, which means that the action is only performed on certain values of the attribute.

> **Info**
>
> An attribute is removed from the set if its last value is removed. It means that it's not possible to pass through attributes without at least one value.

Using conditions

You can use the `<Condition>` node in a filter rule just like with converter. The syntax is the same. So if you omit the `Condition` element then the rule is evaluated unconditionally.

There is one slight difference: in FilterRule, **AttributeMatch is always evaluated on the original input attribute set**. It means that you can reference attributes in conditions even if they were allowed or denied before. (This is what you would normally expect, though.)

You can allow or deny multiple attributes within one `<FilterRule>`. Note that the rule only applies if all the conditions within its `<Condition>` element evaluate to true.

Examples

```
<?xml version="1.0" encoding="UTF-8"?>
<AttributeFilter  xmlns='urn:geant:edugain:attribute-mangling:1.0'
   xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
   xsi:schemaLocation='urn:geant:edugain:attribute-mangling:1.0 AttributeMangling.xsd'>

   <FilterRule>
       <Description>Unconditional allowing and denying. The main rule is to deny ALL
attributes.</Description>
       <AllowAttribute attributeName="mail" />
       <AllowAttribute attributeName="cn" />
       <AllowAttribute attributeName="eppn" />
       <DenyAttribute attributeName="userPassword" />
       <DenyAttribute attributeName="uid" />
       <DenyAttribute attributeName="homeOrganization" />
       <AllowAttribute attributeName="schacHomeOrganization" />
   </FilterRule>

   <FilterRule>
       <Description>Use conditions - allow only specific attribute values</Description>
       <Condition>
           <LocalProviderMatch>^urn:.*\.hu$</LocalProviderMatch>
       </Condition>
       <AllowAttribute attributeName="eduPersonEntitlement">
           <AttributeValue>^.*@.*\.hu$</AttributeValue>
       </AllowAttribute>
   </FilterRule>
```

```
    <FilterRule>
        <Description>Use conditions - reference any attribute</Description>
        <Condition>
            <AttributeMatch attributeName="homeOrganization">niif.hu</AttributeMatch>
        </Condition>
        <AllowAttribute attributeName="eduPersonEntitlement">
            <AttributeValue>^.*@niif\.hu$</AttributeValue>
        </AllowAttribute>
    </FilterRule>

</AttributeFilter>
```

Integration

You can integrate Attribute Conversion and Filtering into your Bridging Element by using these java code snippets. (Of course, edugain.jar and converter.jar need to be placed on the classpath.)

Initialization time

This code needs to be invoked in BE initialization time (and not in runtime, as XML configuration parsing is a time-consuming process).

```
  // get a reference to the AttributeConverterFactory singleton object
  AttributeConverterFactory factory = AttributeConverterFactory.getInstance();
  // set the configuration file paths (which paths can be set in web.xml for example)
  factory.setAttributeConverterFilePath("path-to-converter.xml");
  factory.setAttributeFilterFilePath("path-to-filter.xml");
  factory.setAttributeNameMapperFilePath("path-to-namemapper.xml");

  // create converter and filter objects
  try {
    AttributeConverter converter = factory.createAttributeConverter();
    AttributeFilter filter = factory.createAttributeFilter();
  } catch (ConfigurationException ex) {
    // handle configuration errors (missing files, not valid xmls and more issues)
    log.error(ex);
  }
```

Runtime

This code is invoked in BE runtime. You should have a List of AttributeValues, which was either received from the IdP or from the H-BE. You will get the output attribute set after invoking `process()` method. Note that `process()` takes two more arguments: `remote` and `local`. These represent the local and remote peers that your BE bridges together. Use of these identifiers is optional, you can pass `null`.

> **Figyelem**
>
> If you do not pass `local` or `remote` then rules containing `LocalProviderMatch` or `RemoteProviderMatch` will **NOT** be executed.

```
String remote = "remote-federation-peer-identifier";
String local = "local-federation-peer-identifier";

// get Attributes from the assertion
List<AttributeValues> input = ...;

// Call converter and filter in order.
// Home BE should call converter first, remote BE should call filter first.
if (isHomeBE) {
  List<AttributeValues> output = converter.process(input, remote, local);
  output = filter.process(output, remote, local);
} else {
  List<AttributeValues> output = filter.process(input, remote, local);
  output = converter.process(output, remote, local);
}

// process output here, create new assertion, etc.
```

Testing

XMLTest.sh

Attribute conversion library comes with XML-based test framework. The test can be invoked by the `net.geant.edugain.attributes.xmltest.AttributeConverterTest` main class. There is the **`XMLTest.sh`** script attached to the project, which makes it easy to execute the testing framework.

```
$ ./XMLTest.sh

Attribute Converter Test usage

java AttributeConverterTest
  [-debug]
  [AttributeConverter.xml](-converterconfig)
  [AttributeFilter.xml](-filteringconfig)
  [AttributeNameMapping.xml](-attributenameconfig)
  [output.xml](-output)
  input.xml
```

Example test configuration

The `xmltest/` directory contains the following examples

`AttributeNameMapper.xml` converter name mapper configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<AttributeMapper
   xmlns='urn:geant:edugain:attribute-mapper:1.0'>

    <AttributeDefinition
    Id="mail"
    AttributeName="urn:mace:dir:attribute-def:mail">
        <Attribute AttributeName="urn:oid:0.9.2342.19200300.100.1.3" />
    </AttributeDefinition>

    <AttributeDefinition
    Id="edupersonAffiliation"
    AttributeName="urn:mace:dir:attribute-def:eduPersonAffiliation" />

    <AttributeDefinition
    Id="homeOrganization"
    AttributeName="urn:mace:dir:attribute-def:homeOrganization" />

    <AttributeDefinition
    Id="cn"
    AttributeName="urn:mace:dir:attribute-def:cn">
        <Attribute AttributeName="urn:oid:2.5.4.3"/>
    </AttributeDefinition>
```

```
</AttributeMapper>
```

`AttributeConverter.xml` converter configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<AttributeConverter
   xmlns='urn:geant:edugain:attribute-mangling:1.0'>
    <BasicRule>
        <Description>Create static attribute</Description>
        <Attribute attributeName="eduPersonAffiliation" replaceValues="false">
            <AttributeValue>staff@niif.hu</AttributeValue>
        </Attribute>
    </BasicRule>
    <BasicRule>
        <Description>Create static attribute for some remote providers</Description>
        <Condition>
            <RemoteProviderMatch>^urn:geant:edugain:be:[^:]+\.hu$</RemoteProviderMatch>
        </Condition>
        <Attribute attributeName="homeOrganization">
            <AttributeValue>niif.hu</AttributeValue>
        </Attribute>
    </BasicRule>
    <MergeRule>
        <Description>Merges the common name to the email address(es)</Description>
        <InputAttribute attributeName="cn" />
        <InputAttribute attributeName="mail" />
        <Attribute attributeName="mail" replaceValues="true">
            <AttributeValue>${cn} &lt;${mail}&gt;</AttributeValue>
        </Attribute>
    </MergeRule>
</AttributeConverter>
```

`AttributeTest.xml` test input xml

```
<?xml version="1.0" encoding="UTF-8"?>
<AttributeTest
   xmlns='urn:geant:edugain:attribute-test:1.0'
   Remote="urn:geant:edugain:be:niif.hu" >
    <Attribute AttributeName="urn:oid:0.9.2342.19200300.100.1.3">
        <AttributeValue>adam.lantos@niif.hu</AttributeValue>
```

```
        <AttributeValue>hege@niif.hu</AttributeValue>
    </Attribute>
    <Attribute AttributeName="urn:mace:dir:attribute-def:eduPersonAffiliation">
        <AttributeValue>staff</AttributeValue>
    </Attribute>
    <Attribute AttributeName="urn:oid:2.5.4.3">
        <AttributeValue>Adam Lantos</AttributeValue>
    </Attribute>
</AttributeTest>
```

Output of the example

```
$ ./XMLTest.sh \
  -converterconfig xmltest/AttributeConverter.xml \
  -attributenameconfig xmltest/AttributeNameMapper.xml \
  xmltest/AttributeTest.xml

AttributeConverter.<init> (80) : Creating new rule: class
net.geant.edugain.attributes.rules.BasicRule
AttributeConverter.<init> (81) : Rule Description: Create static attribute
AttributeConverter.<init> (80) : Creating new rule: class
net.geant.edugain.attributes.rules.BasicRule
AttributeConverter.<init> (81) : Rule Description: Create static attribute for some remote
providers
AttributeConverter.<init> (80) : Creating new rule: class
net.geant.edugain.attributes.rules.MergeRule
AttributeConverter.<init> (81) : Rule Description: Merges the common name to the email
address(es)
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<AttributeTest xmlns="urn:geant:edugain:attribute-test:1.0">
   <Attribute AttributeName="urn:mace:dir:attribute-def:eduPersonAffiliation">
       <AttributeValue>staff</AttributeValue>
       <AttributeValue>staff@niif.hu</AttributeValue>
   </Attribute>
   <Attribute AttributeName="urn:mace:dir:attribute-def:mail">
       <AttributeValue>Adam Lantos &amp;lt;adam.lantos@niif.hu&amp;gt;</AttributeValue>
       <AttributeValue>Adam Lantos &amp;lt;hege@niif.hu&amp;gt;</AttributeValue>
   </Attribute>
   <Attribute AttributeName="urn:mace:dir:attribute-def:homeOrganization">
       <AttributeValue>niif.hu</AttributeValue>
```

```
    </Attribute>
    <Attribute AttributeName="urn:mace:dir:attribute-def:cn">
        <AttributeValue>Adam Lantos</AttributeValue>
    </Attribute>
</AttributeTest>
```

Real-life examples

> **A szócikk vagy fejezet még megírásra vár**
>
> Please post your BE configuration here.

Hungarnet

Home

Remote

Változat #3
document-uploader hozta létre 2025-08-07 12:04:07 CEST
cziernorbert frissítette 2026-04-13 15:36:36 CEST