Creating Metadata types

The exisiting set of Metadata can be extended with own types.

The are two types of metadata:

- Base Metadatypes (extends AbstractBaseMetaAttribute)
- Combined Metadatatypes (extends AbstractCombinedMetaAttribute)

Important:

- If you want to use combination of existing types (e.g. the new type and TimeInterval) you could define a combined metadata ~~or let Odysseus do the work for you. In this case you will need to run Odysseus with the JDK and not JRE (i.e. tools.jar must be available).~~
- Because there is a one to one mapping between a metadata interface and a metadata type, **all metadata types must be final**.

Creating a new base metadata type

In our example we will use the trust metadata (de.uniol.inf.is.odysseus.trust). This is just a double value.

To create new base metadata type you first need to create an interface to set and retrieve the values that should be contained in this metadata. This interface must extend IMetaAttribute.

```
public interface ITrust extends IMetaAttribute {

	void setTrust(double trustValue);
	double getTrust();

}
```

This interface must be implemented by the content holding class:

```
public final class Trust extends AbstractBaseMetaAttribute implements ITrust {

	private static final String TRUSTVALUE = "trustvalue";
	private static final String NAME = "Trust";

	@SuppressWarnings("unchecked")
	public final static Class<? extends IMetaAttribute>[] classes = new Class[] { ITrust.class };

	public static final List<SDFMetaSchema> schema = new ArrayList<>(classes.length);

	static {
		List<SDFAttribute> attributes = new ArrayList<>();
		attributes.add(new SDFAttribute(NAME, TRUSTVALUE, SDFDatatype.DOUBLE));
		schema.add(SDFSchemaFactory.createNewMetaSchema(NAME, Tuple.class, attributes, ITrust.class));
	}

	@Override
	public List<SDFMetaSchema> getSchema() {
		return schema;
	}

	@Override
	public Class<? extends IMetaAttribute>[] getClasses() {
		return classes;
	}

	private static final long serialVersionUID = -426212407481918604L;
	private double trustValue;

	public Trust() {
		this.trustValue = 1;
	}

	public Trust(Trust trust) {
		this.trustValue = trust.trustValue;
	}

	@Override
```

```
	public String getName() {
		return NAME;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj) {
			return true;
		} else if (!(obj instanceof Trust)) {
			return false;
		}
		return this.trustValue == ((Trust) obj).trustValue;
	}

	@Override
	public void retrieveValues(List<Tuple<?>> values) {
		@SuppressWarnings("rawtypes")
		Tuple t = new Tuple(1, false);
		t.setAttribute(0, Double.valueOf(this.trustValue));
		values.add(t);
	}

	@Override
	public void writeValue(Tuple<?> value) {
		this.trustValue = ((Double) (value.getAttribute(0))).doubleValue();
	}

	@SuppressWarnings("unchecked")
	@Override
	public <K> K getValue(int subtype, int index) {
		switch (index) {
		case 0:
			return (K) Double.valueOf(this.trustValue);
		default:
			return null;
		}
	}

	@Override
	public void setTrust(double value) {
		this.trustValue = value;
	}

	@Override
	public double getTrust() {
		return this.trustValue;
	}

	@Override
	protected IInlineMetadataMergeFunction<? extends IMetaAttribute> getInlineMergeFunction() {
		return new TrustMergeFunction();
	}

	@Override
	public ITrust clone() {
		return new Trust(this);
	}

	@Override
	public String toString() {
		return "" + this.trustValue;
	}

}
```

There are quite a lot of methods that need to be implemented. This is necessary to allow combination of metadata types.

```
	@SuppressWarnings("unchecked")
	public final static Class<? extends IMetaAttribute>[] classes = new Class[] { ITrust.class };

	public static final List<SDFMetaSchema> schema = new ArrayList<>(classes.length);

	static {
		List<SDFAttribute> attributes = new ArrayList<>();
		attributes.add(new SDFAttribute(NAME, TRUSTVALUE, SDFDatatype.DOUBLE));
		schema.add(SDFSchemaFactory.createNewMetaSchema(NAME, Tuple.class, attributes, ITrust.class));
	}

	@Override
	public List<SDFMetaSchema> getSchema() {
		return schema;
	}

	@Override
	public Class<? extends IMetaAttribute>[] getClasses() {
		return classes;
	}
```

First of all, this class must define, which metatype it provides (here ITrust.class, line 2) and provide a schema for the metadata. This is the same as for operators where an output schema need to be defined with one difference: The created schema must be a Metaschema (line 9).

Then a default constructor and a copy constructor must be defined, and a method that returns the name of the metadata:

```
	private double trustValue;

	public Trust() {
		this.trustValue = 1;
	}

	public Trust(Trust trust) {
		this.trustValue = trust.trustValue;
	}

	@Override
	public String getName() {
		return NAME;
	}
```

The next part contains methods to set and get the values:

```
	@Override
	public void setTrust(double value) {
		this.trustValue = value;
	}

	@Override
	public double getTrust() {
		return this.trustValue;
	}
```

The following methods must be used to handle cases with tuples. This methods are important to allow sending metadata via network or writing to a file.

retrieveValues requires, that the metadata is returned inside a tuple. So if there are multiple values (e.g. with latency) use different positions inside the tuple. Important use for reading and writing the same positions!

writeValue ist used to set the value of the metadata object from a tuple object (same position 0 as in retrieve).

getValue is used to get a single value from the metadata, index is the same as the position inside the tuple (see Latency for a complexer example). subtype is not used in base metadata. This is only used in combined metadata.

```
	@Override
	public void retrieveValues(List<Tuple<?>> values) {
		@SuppressWarnings("rawtypes")
		Tuple t = new Tuple(1, false);
		t.setAttribute(0, Double.valueOf(this.trustValue));
		values.add(t);
	}

	@Override
	public void writeValue(Tuple<?> value) {
		this.trustValue = ((Double) (value.getAttribute(0))).doubleValue();
	}

	@SuppressWarnings("unchecked")
	@Override
	public <K> K getValue(int subtype, int index) {
		switch (index) {
		case 0:
			return (K) Double.valueOf(this.trustValue);
		default:
			return null;
		}
	}
```

Metadata must be combined, e.g. in a join or in an aggregation. The method getInlineMergeFunction returns a function that can do the merge.

```
	@Override
	protected IInlineMetadataMergeFunction<? extends IMetaAttribute> getInlineMergeFunction() {
		return new TrustMergeFunction();
	}
```

This merge function must be provided as follows:

```
public class TrustMergeFunction implements IInlineMetadataMergeFunction<ITrust> {

	@Override
	public void mergeInto(ITrust result, ITrust inLeft, ITrust inRight) {
		result.setTrust(Math.min(inLeft.getTrust(), inRight.getTrust()));
	}

	@Override
	public IInlineMetadataMergeFunction<? super ITrust> clone() {
		return this;
	}

	@Override
	public Class<? extends IMetaAttribute> getMetadataType() {
		return ITrust.class;
	}

}
```

There must be implemented three methods:

getMetadataType() must deliver the metadata type for which this function will be used. (line

clone() must return an instance of this function. If the function has a state, it is important to create a new function. Typically, this is not the case, so just return this.

mergeInto is the function that is called by the framework to merge two metadata values. The result must be written to the inout parameter result (this is necessary for easy handling of combined metadata). In this case, the new trust value is the mininum of both values.

Finally, this metadata must be registered. This is done with an OSGi component:

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0" name="de.uniol.inf.is.odysseus.trust.Trust">
   <implementation class="de.uniol.inf.is.odysseus.trust.Trust"/>
   <service>
      <provide interface="de.uniol.inf.is.odysseus.core.metadata.IMetaAttribute"/>
   </service>
</scr:component>
```

and registered inside the MANIFEST.MF (line 11)

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Trust Metadata
Bundle-SymbolicName: de.uniol.inf.is.odysseus.trust
Bundle-Version: 1.0.0.qualifier
Require-Bundle: de.uniol.inf.is.odysseus.core,
 de.uniol.inf.is.odysseus.core.server,
 de.uniol.inf.is.odysseus.transform;bundle-version="1.0.0",
 de.uniol.inf.is.odysseus.ruleengine;bundle-version="1.0.0"
Export-Package: de.uniol.inf.is.odysseus.trust
Service-Component: OSGI-INF/Trust.xml
Automatic-Module-Name: de.uniol.inf.is.odysseus.trust
Bundle-ActivationPolicy: lazy
Import-Package: org.slf4j
```

Finally, there should be operators that could set the values.

For examples see:

- https://git.swl.informatik.uni-oldenburg.de/projects/ODY/repos/odysseus_core/browse/server/trust/src/de/uniol/inf/is/odysseus/trust/physicaloperator/TrustUpdater.java
- https://git.swl.informatik.uni-oldenburg.de/projects/ODY/repos/odysseus_core/browse/server/trust/src/de/uniol/inf/is/odysseus/trust/logicaloperator/TrustAO.java
- https://git.swl.informatik.uni-oldenburg.de/projects/ODY/repos/odysseus_core/browse/server/trust/src/de/uniol/inf/is/odysseus/trust/transform/TTrustAORule.java

Creating a combined metadata type

Different metadata types can be used together. For this, in the best case, a single class should be provided, that combines this data. Typically, this class just delegates to the base classes. In the following is an example with trust and timeinterval:

```
public class IntervalTrust extends AbstractCombinedMetaAttribute implements ITimeInterval, ITrust {

	private static final long serialVersionUID = 1599620389994530920L;

	@SuppressWarnings("unchecked")
	public final static Class<? extends IMetaAttribute>[] classes = new Class[] { ITimeInterval.class,
ITrust.class };

	@Override
	public Class<? extends IMetaAttribute>[] getClasses() {
		return classes;
	}

	public static final List<SDFMetaSchema> schema = new ArrayList<>(classes.length);

	static {
		schema.addAll(TimeInterval.schema);
		schema.addAll(Trust.schema);
	}

	@Override
	public List<SDFMetaSchema> getSchema() {
		return schema;
	}

	@Override
	public String getName() {
		return "IntervalTrust";
	}
```

The new class must extend AbstractCombinedMetaAttribute and implements the base interfaces for this combination.

The classes for this metadata is the same set of interfaces, and should be in lexical order.

The schema is simply a list containing the subschema of the containing metadata.

```
	private final ITrust trust;
	private final ITimeInterval timeInterval;

	public IntervalTrust() {
		this.trust = new Trust();
		this.timeInterval = new TimeInterval();
	}

	public IntervalTrust(IntervalTrust other) {
		this.trust = (ITrust) other.trust.clone();
		this.timeInterval = (ITimeInterval) other.timeInterval.clone();
	}

	@Override
	public IntervalTrust clone() {
		return new IntervalTrust(this);
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj) {
			return true;
		} else if (!(obj instanceof IntervalTrust)) {
			return false;
		}
		IntervalTrust other = (IntervalTrust) obj;
		return this.trust.equals(other.trust) && this.timeInterval.equals(other.timeInterval);
	}
```

Also similiar to the basetypes, there must be a default constructor, a copy constructor a clone-method and an equals method.

```
	// --------------------------------------------------------------------------
	// Methods that need to merge different types
	// --------------------------------------------------------------------------

	@Override
	public void retrieveValues(List<Tuple<?>> values) {
		this.timeInterval.retrieveValues(values);
		this.trust.retrieveValues(values);
	}

	@Override
	public void writeValues(List<Tuple<?>> values) {
		this.timeInterval.writeValue(values.get(0));
		this.trust.writeValue(values.get(1));
	}

	@Override
	public List<IInlineMetadataMergeFunction<? extends IMetaAttribute>> getInlineMergeFunctions() {
		List<IInlineMetadataMergeFunction<? extends IMetaAttribute>> list = new ArrayList<>();
		list.addAll(this.timeInterval.getInlineMergeFunctions());
		list.addAll(this.trust.getInlineMergeFunctions());
		return list;
	}

	@Override
	public <K> K getValue(int subtype, int index) {
		switch (subtype) {
		case 0:
			return this.timeInterval.getValue(0, index);
		case 1:
			return this.trust.getValue(0, index);
		default:
			return null;
		}
	}

	@Override
	public String toString() {
		return "( i= " + this.timeInterval.toString() + " | " + " l=" + this.trust + ")";
	}
```

The next block contains methods to retrieve and set the combined values and merge functions. As you can see, here is always delegated to the base metadata. Important here is to keep the right order!

In getValue, the subtype, says from which metadata the value should be retrieved.

The last block just contains the getter and setter for the metadata and is always delegated to the base types:

```
        // -------------------------------------------------------------------------
        // Delegates for timeInterval
        // -------------------------------------------------------------------------

        @Override
        public PointInTime getStart() {
                return this.timeInterval.getStart();
        }

        @Override
        public PointInTime getEnd() {
                return this.timeInterval.getEnd();
        }

        @Override
        public void setStart(PointInTime point) {
                this.timeInterval.setStart(point);
        }

        @Override
        public void setEnd(PointInTime point) {
                this.timeInterval.setEnd(point);
        }

        @Override
        public void setStartAndEnd(PointInTime start, PointInTime end) {
                this.timeInterval.setStartAndEnd(start, end);
        }

        @Override
        public int compareTo(ITimeInterval o) {
                return this.timeInterval.compareTo(o);
        }

        // -------------------------------------------------------------------------
        // Delegates for trust
        // -------------------------------------------------------------------------

        @Override
        public double getTrust() {
                return this.trust.getTrust();
        }

        @Override
        public void setTrust(double trustValue) {
                this.trust.setTrust(trustValue);
        }
}
```

When Odysseus at Runtime does not find such a combined metadata it will create a new one and compile it. This is slower and does not always work. So it is better to create a combination.

You could use: GenerateMetadataClassCode (https://git.swl.informatik.uni-oldenburg.de/projects/ODY/repos/odysseus_core/browse/common/core/src/de/uniol/inf/is/odysseus/core/metadata/GenerateMetadataClassCode.java) as a generator for the combined metadata.

Remark: Typically, it is best to put each combination of metadata to an own bundle to avoid to many dependencies.

Finally, this class need to be registered as metadata, too:

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0" name="de.uniol.inf.is.odysseus.interval_trust.
IntervalTrust">
   <implementation class="de.uniol.inf.is.odysseus.interval_trust.IntervalTrust"/>
   <service>
      <provide interface="de.uniol.inf.is.odysseus.core.metadata.IMetaAttribute"/>
   </service>
</scr:component>
```