WS-Policy Framework Overview

The WS-Policy framework provides infrastructure and APIs that allow CXF users and developers to use WS-Policy.

It is compliant with the Web Services Policy 1.5 - Framework and Web Services Policy 1.5 - Attachment specifications.

The framework consists of a core runtime and APIs that allow developers to plug in support for their own domain assertions:

Core

The core is responsible for:

- retrieval of policies from different sources (wsdl documents, external documents)
- computation of effective policies for service, endpoint, operation and message objects
- on-the-fly provision of interceptors based on the effective policies for a particular message
- verification that one of the effective policy's alternatives is indeed supported.

Policy operations such as merge, normalisation, and intersection are based on Apache Neethi.

APIs

AssertionBuilder

The AssertionBuilder API is an interface from Neethi.

AssertionBuilder implementations are loaded dynamically and are automatically registered with the AssertionBuilderRegistry, which is available as a Bus extension. Currently, CXF supports AssertionBuilder and Assertion implementations for the following assertion types:

- `{http://schemas.xmlsoap.org/ws/2005/02/rm/policy}` RMAssertion
- `{http://www.w3.org/2007/01/addressing/metadata}` Addressing
- `{http://www.w3.org/2007/01/addressing/metadata}` AnonymousResponses
- `{http://www.w3.org/2007/01/addressing/metadata}` NonAnonymousResponses
- `{http://cxf.apache.org/transports/http/configuration}` client
- `{http://cxf.apache.org/transports/http/configuration}` server

along with the WS-SecurityPolicy defined assertions.

They are all based on generic Assertion implementations (PrimitiveAssertion, NestedPrimitiveAssertion, JaxbAssertion) that developers can parameterize or extend when developing their own assertions, see Developing Assertions.

PolicyInterceptorProvider

This API is used to automatically engage interceptors required to support domain specific assertions at runtime, thus simplifying interceptor configuration a lot.

```java
public interface PolicyInterceptorProvider extends InterceptorProvider {
    // return the schema types of the assertions that can be supported
    Collection<QName> getAssertionTypes()
}
```

Currently, CXF supports PolicyInterceptorProvider implementations for the following assertion types:

- `{http://schemas.xmlsoap.org/ws/2005/02/rm/policy}` RMAssertion
- `{http://www.w3.org/2007/01/addressing/metadata}` Addressing
- `{http://www.w3.org/2007/01/addressing/metadata}` AnonymousResponses
- `{http://www.w3.org/2007/01/addressing/metadata}` NonAnonymousResponses

along with the WS-SecurityPolicy defined assertions.

In addition, the framework offers an API to refine domain expression(s) (xml elements describing policy subjects within a policy scope) in policy attachments. There is currently only one implementation for EndpointReferenceType domain expressions (matching over the address). Another implementation, using XPath expressions, is in work.
Interaction with the Framework

Components interact with the policy framework mainly in order to:

1. retrieve the assertions pertaining to the underlying message (at least the ones known to the component) so the component can operate on the message accordingly
2. confirm that the assertions pertaining to the underlying message are indeed supported.

Like most other CXF features, the policy framework is itself largely interceptor based. Thus, most interaction with the framework is indirect through the Message object: Policy interceptors make AssertionInfo objects (stateful representations of assertions) available to subsequently executing, policy-aware interceptors by inserting them into the Message object. Extracting the AssertionInfo objects from the Message allows interceptors to perform steps 1. and 2. above:

```java
import org.apache.neethi.Assertion;

public class AssertionInfo {
    ...
    public boolean isAsserted() {...}
    public void setAsserted(boolean asserted) {...}
    public Assertion getAssertion() {...}
}
```

The WS-Addressing and WS-RM interceptors are examples for this style of intercation.

Sometimes, Conduits and destinations also want to assert their capabilities. But they cannot normally wait for Assertion information being made available to them via the Message object:

- Conduits may exhibit message specific behaviour (for example, apply message specific receive timeouts), but decisions made during the initialisation phase may limit their capability to do so.
- And Destinations cannot normally exhibit message or operation specific behaviour at all. But both may still be able to support assertions in the effective endpoint’s policy.

Their interaction with the policy framework therefore typically involves the PolicyEngine through which they obtain the effective policy for the underlying endpoint (for step 1.):

```java
public interface PolicyEngine {
    ...
    EndpointPolicy getClientEndpointPolicy(EndpointInfo ei, Conduit conduit);
    EndpointPolicy getServerEndpointPolicy(EndpointInfo ei, Destination destination);
}

public interface EndpointPolicy {
    ...
    Policy getPolicy();
    Collection<Assertion> getChosenAlternative();
}
```

To perform step 2. they implement the Assertor interface (namely its assertMessage method):

```java
public class Assertor {
    ...
    public boolean canAssert(QName name);
    public void assertMessage(Message message);
}
```

An example for policy aware conduits and destinations in CXF are the HTTP conduit and destination. They do support assertions of element type HTTPClientPolicy and HTTPServerPolicy respectively.