Orchard-Maler Assertion Proposal

Document Name and Version

draft-orchard-maler-assertion-00

Publication Date

15 June 2001

Authors

David Orchard, dorchard@jamcracker.com

Eve Maler, eve.maler@east.sun.com

Abstract

This document offers a proposal for SAML assertions and the XML structure that conveys them to and from SAML Authorities. The structure is simple, easily implementable, and intuitive to XML Schema-aware developers, allowing for faster time to development. Many parts of this proposal borrow concepts that are much more fully defined in the core-07 proposal.

Introduction

This document offers a proposal for SAML assertions and the XML structure that conveys them to and from SAML Authorities. The structure is simple, easily implementable, and intuitive to XML Schema-aware developers, allowing for faster time to development.

Many parts of this proposal borrow concepts that are much more fully defined in the core-07 proposal. We have tried to capture all TBD design issues here; many of them roughly correspond to the numbered issues currently faced by the TC.

Definitions

The following definitions are used in this proposal:

- **Request**: A SAML-compliant XML structure (“compound”) that asks for a particular SAML Authority to produce assertions.
- **Response**: A SAML-compliant XML structure (“compound”) that encodes the assertions produced by a SAML Authority on request.
- **Assertions package**: A grouping of atomic assertions (“molecule”). The core-07 proposal called this an “assertion.”
- **Assertion**: A single declaration of fact (“atom”). The core-07 proposal called this a “claim.”
- **Metadata**: Properties of an XML structure that apply equally to all parts of it. For example, an assertion has metadata that identifies who issued it and when, and a request has metadata indicating what version of SAML was used to encode it.

Document Conventions

XML **element** and **attribute** names are shown in bold; typically these elements would be declared to have complex types that are anonymous. XML *complex type* names that are abstract and do not necessarily correspond directly to elements are shown in italic.

The class diagram notation uses UML; *abstract class* names are italicized in correspondence with XML abstract complex types. In the diagram, parent elements are shown above their child elements. The cardinality shown on each relationship line represents the number of child elements allowed inside each instance of the parent element. Order of child elements within a parent element is not precisely shown in this diagram, though the schema mostly uses sequential content models.

XML Design Principles

The proposed design adheres to the following principles for XML structure design.

1. **Top typing**: Use XML Schema complex typing to identify commonalities as high up in the XML tree as possible. This allows XML validators to function as “free error checkers” on assertions and improves performance of streaming tools. With suitable definition of subtypes, we believe it is possible to use any style of

querying (not just XML Query) with SAML, and so the decision on querying style can be made independently of this principle.

2. **Isolate extensions**: Use XML Namespaces and XML Schema to isolate extensibility features where possible, so that schema modules can be used to ensure compliance with extensions and so that extensions can be uniquely referred to with XML namespace names. This makes it easier to describe conformance to extensions.
3. **Existing vocabularies**: Consider reusing existing XML vocabularies where they exist, are well supported, and directly address a SAML need. For example, if SAML needed a facility for marking up error messages, it should prefer XHTML to a new SAML-specific vocabulary.
4. **Elements vs. attributes**: Tend towards attributes for metadata and "single-field" information, and elements for any content that has distinguishable subparts.

SAML Message Architecture

SAML-encoded information can be conveyed as a whole message in its own right ("standalone SAML"), without being embedded in another XML structure such as a purchase order. The form it takes for this purpose is either a request message or a response message. It is presumed that a SAML message is conveyed by some external means of transport/messaging (which could include an XML-based messaging protocol such as SOAP); this is in the purview of the Bindings subcommittee.

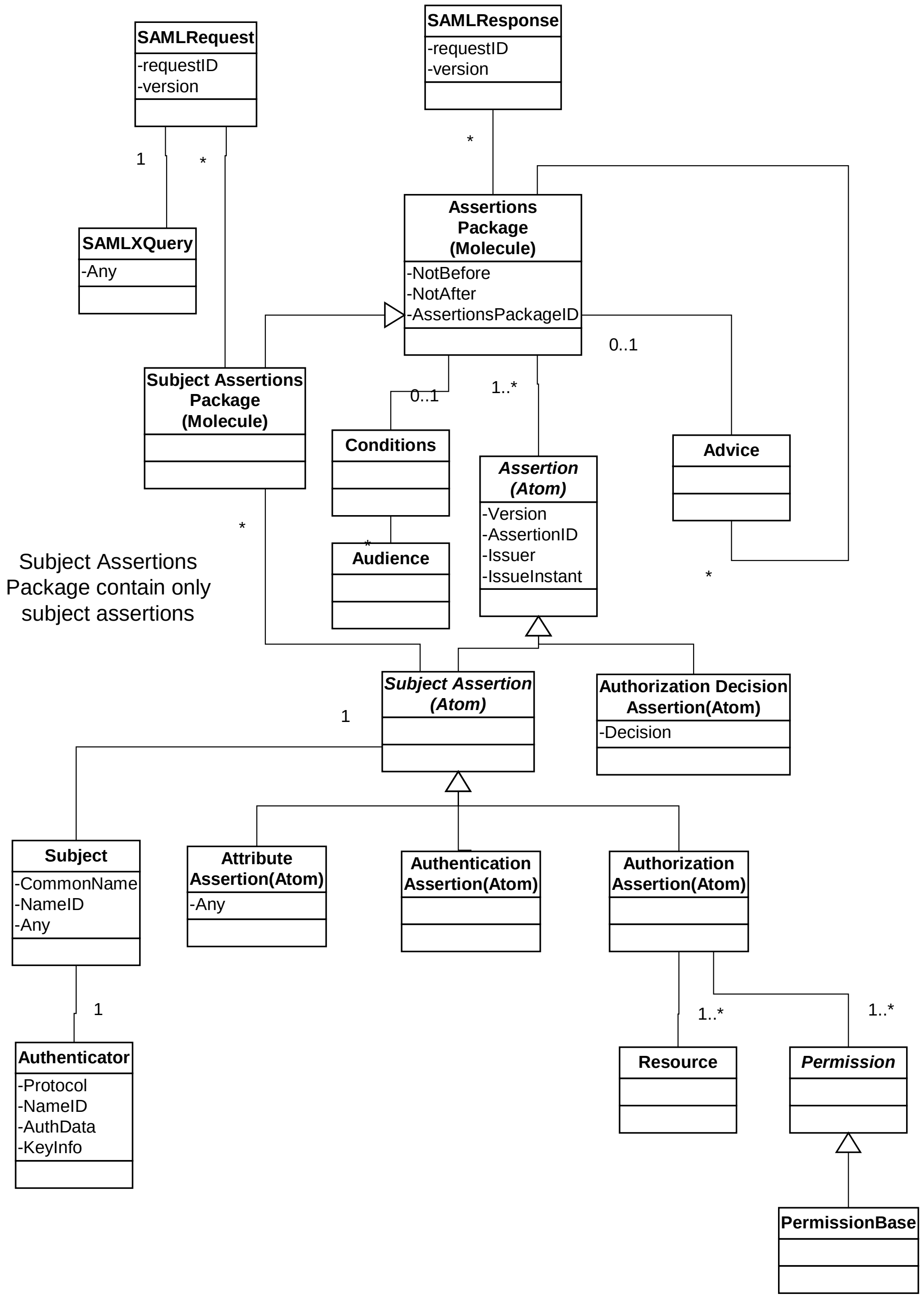
Because it may be necessary to embed SAML assertions inside other XML structures ("embedded SAML"), we anticipate that these higher-level request and response structures might not always be used. Thus, there is a **Version** attribute both on **SAMLRequest** and **SAMLResponse**, and also on all the individual assertion elements.

The class diagram below shows the outlines of the entire structure.

Issues:

1. What provision do we need to make for digitally signing requests and responses? What subparts need to be signed individually?
2. Where a particular binding chooses to extract some of the SAML-native information and present it in out-of-band layers, how should the SAML schema handle the possibility of missing information? Can it be assumed that the process of extracting the information is done after validation on the producing end and that there is a process of re-introducing it into the SAML stream before the consuming end validates it?
3. How should unique IDs be handled? Currently we have put generic ***ID** attributes in the places where we think IDs should be, and have not said what the constraints on their content or handling are. We have also proposed that the **Issuer** attribute contain a fully qualified DNS domain name. If an issuer/serial number pair is chosen, it would require the ***ID** attributes to become ***SerialNumber** attributes.
4. In the case of "embedded SAML," would single assertions be embedded, or would whole assertions packages be embedded? This decision will have an affect on the pattern of metadata available on these two layers.

5. How would Policies be added to the model for XACML queries?



SAMLRequest Element

The request message element, **SAMLRequest**, is a collection of XML-encoded SAML information that is intended to be sent to a SAML Authority. It puts the actual query in the required **SAMLXQuery** element, and may also supply zero or more **SubjectAssertionsPackage**s as auxiliary input.

The query operates over all the assertions available to the SAML Authority being queried, *plus* the assertions provided as auxiliary input. It is expected that implementations will store the assertion information in proprietary mechanisms, such as various RDBMS tables, LDAP tables, files, etc. Thus a query is made against a “virtual” model.

The request has metadata indicating the version of SAML (**Version**) in which the message is encoded and a unique identifier for the request (**RequestID**).

Rationale:

The need for providing assertions as auxiliary input is demonstrated by the dotted-line relationships in our domain model, in which (for example) Authentication Assertions can serve as input to Attribute Authorities that ultimately generate Attribute Assertions. Each assertions package has the opportunity to provide its own **Conditions** and **Advice**.

Issues:

1. Should a SAML request allow for additional non-SAML auxiliary information, akin to **Advice**?
2. Should the request ID be handled differently? A “requester” field (similar to **Issuer**) might be needed on the request as a whole if a two-part unique identified system is used.

SAMLXQuery Element

The main content of a **SAMLRequest** is the query itself, the **SAMLXQuery** element.

This document proposes the use of a subset of XML Query, including FLWR expressions (FOR, LET, WHERE, RETURN) and OPERATIONS, but not functions, conditionals, filtering, or custom data types.

Rationale:

The element was given the name **SAMLXQuery** because it is a SAML-specific subsetting of a query in XML Query form. It is the only element, other than the two top-level message elements, that has “SAML” in the name.

The XML Query approach is being proposed for the following reasons:

- It achieves a higher level of reuse of other specifications, following design principle #3.
- It will tend to increase developer productivity because XML Query engines already exist.
- It allows developers to focus on the data model rather than the query syntax.
- It allows arbitrary new kinds of queries to be generated without changes in the SAML specification or deployed SAML-compliant systems.

Issues:

1. What form should the query take? The most recent Focus telecon listed three possible directions to go with this: allow only specific forms of request that have no variability in them (not really a query at all), a SAML-specific query language along the lines of core-07’s **Respond** element identifiers, and a (subset of a) generalized query language such as XML Query.

2. If we go with the XML Query approach, we are assuming that subsetting is required. Is the subsetting necessary? How should this subsetting be done? Should the subset be enforced in the SAML schema by making the query elements be SAML-native elements? This would allow greater control over the inbound elements and help conformance, but would not give us the same reuse benefits because they would no longer be in the XML Query namespace.
3. Even if XML Query is used, should there be in addition a shorthand notation for common query structures, along the lines of core-07's **Respond** element? An analogy is that Xpath has a short-hand and long-hand syntax. Most people use the short-hand syntax.

SubjectAssertionsPackage Element

The auxiliary input to a SAML request is an optional **SubjectAssertionsPackage** element, which contains one or more assertions of the *SubjectAssertion* type; in addition to inheriting metadata attributes, these assertions all share the characteristic that they require a **Subject** element as their first subelement. SAML should be able to be extended to add new assertions of this type. The **SubjectAssertionsPackage** element is a subtype of **AssertionsPackage**, and inherits metadata attributes from it.

Rationale:

Following design principle #1, The *SubjectAssertion* type factors out the commonalities in an important set of assertions, those that are subject-centric. Such assertions may require handling that is different from non-subject-centric assertions, and therefore this deserves its own type. We anticipate that some extension assertions (for example, session assertions) will want to be of this subtype.

Issues:

1. Should **SubjectAssertionsPackage** inherit **Conditions** and **Advice** as well as the metadata attributes?

SAMLResponse Element

The response message, **SAMLResponse**, is a collection of XML-encoded SAML information intended to be the output of a SAML Authority. It contains a set of one or more **AssertionsPackage**s generated in response to a request, optionally preceded by a **Conditions** elements and optionally followed by an **Advice** element.

The response has metadata attributes indicating:

- The version of SAML (**Version**) in which the message is encoded
- A reference to the unique identifier for the request that it is responding to (**RequestID**)

The **Conditions** element provides auxiliary data that is specific to the package on which it appears. Currently, this consists only of a series of **Audience** elements, each of which contains a string identifying the relevant audience. SAML Authorities are required to understand and process the contents of any **Conditions** element provided; if they do not understand, they must produce an error.

The **Advice** element provides auxiliary data that is not required for understanding and processing the package. It can contain any content that is not from the SAML namespace.

Rationale:

This structure allows one or more packages because they may have different **NotBefore** and **NotAfter** values.

This structure disallows repetition of the **Conditions** and **Advice** elements because a single element is enough to contain whatever conditions or advice is necessary, and there are no metadata attributes on these elements that would benefit from multiple instances with different attribute values.

Issues:

1. How should error conditions for responses be handled?
2. Is the **Audience** information in scope for SAML? (Core-07 describes it as a URI that points to a document that identifies the terms and conditions for audience membership.)
3. Is there any other information that SAML should allow in **Conditions**? Should non-SAML namespaces be allowed here?

AssertionsPackage Element

The content of a SAML response is set of **AssertionsPackage** elements, which contains one or more assertions of the *Assertion* type. The **AssertionsPackage** type provides metadata attributes:

- **AssertionsPackageID**: a unique identifier for this package.
- **NotBefore**: The time instant before which the assertions contained within are not valid.
- **NotAfter**: The time instant after which the assertions contained within are not valid.

Rationale:

The **AssertionsPackage** element is useful as a grouping mechanism for several assertions of different kinds whose validity interval metadata is shared in common. For example, a “combination authority” that is capable of producing several different kinds of assertions may produce them all at once in response to a request, and then provide the validity information on the package element that contains them all.

Issues:

1. Is a “binding assertion” needed as a native SAML assertion?
2. Given that individual assertions might be embedded in other XML documents, and given that the AttributeAssertion element implicitly allows multiple attributes in a single assertion, should the **NotBefore** and **NotAfter** attributes go on the assertion level instead of on the package level? There wouldn’t seem to be too much point to the package level if this were done.

Individual Assertion Structures

Individual assertions can be of the *Assertion* type, which provides the following metadata attributes:

- **Version**: The version of SAML used to encode this assertion.

- **AssertionID**: a unique identifier for this assertion.
- **Issuer**: The fully qualified DNS domain name of the issuer.
- **IssueInstant**: A timestamp indicating when the one or more assertions contained within were issued.

SAML can be extended to add new assertions of the *Assertions* type.

Some SAML assertions are further subtyped as being of the *SubjectAssertion* type. SAML can be extended to add new assertions of this type. In addition to having the metadata attributes, these assertions inherit **Subject** as their first child element.

Rationale:

The **Version** attribute is available here because individual assertions might be embedded in other XML structures, such as purchase orders, and an assertion element might thus be a "top-level" SAML element in that context.

AttributeAssertion Element

The **AttributeAssertion** element is of the *SubjectAssertion* type. In addition to its inherited metadata attributes and **Subject** child element, it can contain any amount of non-SAML-namespace elements that convey the attribute data. SAML-compliant systems need to negotiate the attributes they understand by means of XML Schemas.

Rationale:

Following design principle #2, namespaces are used to manage extensibility. XML Schemas allow for complete flexibility in the content model of attributes. This is much more suitable for extensibility than the alternatives of name/value pairs or structured strings.

AuthenticationAssertion Element

The **AuthenticationAssertion** element is of the *SubjectAssertion* type. It contains nothing beyond its inherited metadata attributes and **Subject** child element.

Rationale:

There is only one **Subject** element allowed because conveying multiple authentications is less likely than the scenario of conveying only one of them, and if it is necessary to convey multiple ones, then a package can be used.

AuthorizationAssertion Element

The **AuthorizationAssertion** element is of the *SubjectAssertion* type. In addition to its inherited metadata attributes and **Subject** child element, it contains a **Resource** element and one or more **Permission** elements.

Rationale:

See the issues below.

Issues:

3. Authorization "assertions" seem to be needed only as a way to express policy "facts," and they don't really have a place in our domain model (unless decision assertions eventually turn out to use the basic form described here: subject,

permissions, resource). Should authorization assertions be a kind of auxiliary data, rather than being seen as assertions?

4. What should the structure of the **Resource** element be? Should it be an attribute or an element? It's pretty clear that it probably wants to be a URI reference, but are there any restrictions on what kinds of URI reference? Do we have to say anything about equality rules for resource URIs? Should the **Resource** element allow for plural values?
5. What should the structure of the **Permission** element be? Should its range of possible permissions be extensible?

AuthorizationDecisionAssertion Element

The **AuthorizationDecisionAssertion** element is of the *Assertion* type. It inherits metadata attributes, and has an additional attribute, **Decision**, which provides the decision in response to the request whose ID is referenced in the **SAMLResponse** ancestor of this element.

Rationale:

See the issue below and the information about AuthorizationAssertion above.

Issues:

1. Should decision assertions have a structure more like authentication assertions, repeating the subject, resource and permissions that are being approved? In this case, how would a "no" answer be conveyed?

Subject Element

The **Subject** element appears in the assertions of *SubjectAssertion* type. It contains zero or more **Authenticator** elements, and has two attributes: **CommonName** and **NameID**. The **Authenticator** element has only the following attributes:

- **Protocol**
- **NameID**
- **AuthData**
- **KeyInfo**

Issues:

1. We borrowed the core-07 design for the **Subject** element. We need to understand this structure better, and also there are outstanding TC issues regarding subjects, indexical references, and so on that affect this element directly.
2. Should there be an ID reference from **Subject** to the relevant **AttributeAssertion**?
3. Should **Authenticator** be called **ValidationOfBinding** instead?

Summary of Extensibility Features

Implementations are offered flexibility in the following areas:

- Arbitrary queries against the data model are allowed.

- Arbitrary attribute information is permitted in the **AttributeAssertion** element. Attributes can be in whatever form the implementations agree upon, so long as they can be constrained by a schema and can be represented by an XML Query.
- Additional *Assertion* and *SubjectAssertion* types are allowed to appear. An example might be a **SessionAssertion**, which would be a subtype of *SubjectAssertion*.

Issues:

1. Is it a requirement that other schemas can redefine SAML components? This may make sense in the assertions bindings. For example, a **SOAP-SEC:Assertion** could be redefined from *s0:Assertion*. This will make a difference in how the SAML schema's target namespace is handled.
2. There are other questions about extensibility that appear in the various issues lists above.

Summary of Differences from core-07

1. Removal of Responds element
2. Removal of Bindings and Claims elements, replace with new structures including subject, object, permissions
3. Change of attributes from list of strings to open model
4. Create top-level assertion type with subtypes
5. Move the resource from the claims/bindings/authorization/resource to resource
6. Move the permission from the claims/bindings/authorization/permission to permission

Request Methods

The following are some sample requests that need to be supported by SAML. Some of these came from Tim Moses's recent post.

1. Can Alice read finance?
2. Can Alice read finance with an attribute Assertion?
3. Can Alice read finance with Role Admin?
4. The requestor requests an authentication assertion that will be accepted by an identified secondary domain. The requestor, in its request, identifies the target domain. The responder returns an indication of its success or failure and the resulting assertion or a reference to an assertion (in the event of success) that it stores for later retrieval.
5. The requestor requests an attribute assertion that will be accepted by another (unspecified) secondary domain. The request specifies the requested attributes. For instance, a group name, a role, a signing authority or a security clearance. The responder returns an indication of its success or failure. If it indicates success, it may return the requested assertion or a more general version of the requested assertion. If it indicates failure, it may return nothing or a more constrained version of the requested assertion.

6. The requestor sends a reference to an authentication or attribute assertion to the responder, indicating that it wants the corresponding assertion to be returned. If successful, the responder returns the assertion.
7. The requestor sends a description of an assertion that it would like the responder to locate, retrieve and return. If successful, the responder returns a success indication and an assertion that either exactly meets the requirements or is more general. If unsuccessful, the responder returns a failure indication and (optionally) one or more assertions that are more specific than the one specified. The sample used is Alice trying to read finance, and if she can't read finance or *, then return if she can read finance/f1
8. The requestor sends a question concerning the authorization status of a subject in relation to a specified resource. The subject may be identified by name, by an authentication assertion or by a reference to an authentication assertion. If necessary, the responder locates and retrieves the specified (or a suitable) assertion) and evaluates it in relation to the resource. It can reply in one of three ways: "Yes", "No" or "No, but if you had asked this (more specific) question, the answer would have been 'yes'".

Issues:

1. We need to agree on what types of requests are in scope, and (in each case) which type of SAML Authority they would be addressed to and what the expected response content is. Does the above list capture what we want?

W3C XML Schema Design principles

This section describes the principles used in creating the SAML XML Schema. Many of the principles are from

1. Named types used, rather than anonymous types http://www.xfront.com/ElementVersusType.html
2. The xml schema best practice design pattern of variable content containers using abstract type and type substitution is used, http://www.xfront.com/VariableContentContainers.html

Issues:

1. Should the dangling type pattern be used? This allows removal of the xsi:type attribute. Or can XML Schema SubstitutionGroups be used.
2. Should the ANY content model be used for extension of assertion, as per http://www.xfront.com/ExtensibleContentModels.html?
3. Is it a requirement that other schemas can redefine SAML components? This may make sense in the assertions bindings. For example, a SOAP-SEC:Assertion could be redefined from s0:Assertion. If this is the case, then the chameleon pattern of http://www.xfront.com/ZeroOneOrManyNamespaces.html should be used.
4. Would AttributeGroups be useful for the Assertions attributes

5. Should we make all the single-use complex types anonymous? It's distracting to see name="SAMLQuery" type="s0:SAMLqueryType" and then have a named complex type, when we haven't said we want extensibility for this type.
6. Should the use of local element names with complexTypes be changed to global element names?

Schema and Example Documents

A large number of documents are included here to normatively define the schema, illustrate various extensions, and show samples.

Complete Assertions Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.oasis.org/tbs/1066-12-25/"
xmlns="http://www.w3.org/2000/10/XMLSchema"
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
xmlns:s0="http://www.oasis.org/tbs/1066-12-25/"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
elementFormDefault="unqualified">
    <!-- Schema for all Assertions -->
    <xsd:element name="SAMLRequest" type="s0:SAMLRequestType"/>
    <xsd:complexType name="SAMLRequestType">
        <xsd:sequence>
            <xsd:element name="SAMLXQuery" type="s0:SAMLXQuery" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="s0:SubjectAssertionsPackage" minOccurs="0" maxOccurs="unbounded"/>

        </xsd:sequence>
        <xsd:attribute name="RequestID" type="s0:RequestIDType"/>
        <xsd:attribute name="Version" type="s0:VersionType"/>
    </xsd:complexType>

    <xsd:element name="SAMLResponse" type="s0:SAMLResponseType"/>
    <xsd:complexType name="SAMLResponseType">
        <xsd:sequence>
            <xsd:element ref="s0:AssertionsPackage" minOccurs="1" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="RequestID" type="s0:RequestIDType"/>
        <xsd:attribute name="Version" type="s0:VersionType"/>
    </xsd:complexType>

    <xsd:complexType name="SAMLXQuery" mixed="true">
        <xsd:choice>
            <xsd:any namespace="##any" processContents="skip"/>
        </xsd:choice>
    </xsd:complexType>

    <xsd:element name="AssertionsPackage">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Conditions" type="s0:ConditionsType" minOccurs="0" maxOccurs="1"/>
                <xsd:element name="Assertion" type="s0:AssertionType" minOccurs="1"
maxOccurs="unbounded"/>
                <xsd:element name="Advice" type="s0:AdviceType" minOccurs="0" maxOccurs="1"/>
                <!-- Basic Information -->
            </xsd:sequence>
            <xsd:attribute name="AssertionsPackageID" type="s0:AssertionIDType"/>
            <xsd:attribute name="NotBefore" type="timeInstant"/>
            <xsd:attribute name="NotAfter" type="timeInstant"/>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="SubjectAssertionsPackage">
        <xsd:complexType>
        <xsd:complexContent>
            <xsd:restriction>
            <xsd:sequence>
                <xsd:element name="Assertion" type="s0:SubjectAssertionType" minOccurs="0"
maxOccurs="unbounded"/>
                <!-- Basic Information -->
            </xsd:sequence>
            <xsd:attribute name="RequestID" type="s0:AssertionIDType"/>
            </xsd:restriction>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>


    <xsd:element name="Assertion" type="s0:AssertionType"/>
    <xsd:complexType name="AssertionType" abstract="true">
        <xsd:sequence>
            <!-- Basic Information -->
        </xsd:sequence>
        <xsd:attribute name="Version" type="s0:VersionType"/>
        <xsd:attribute name="AssertionID" type="s0:AssertionIDType"/>
        <xsd:attribute name="Issuer" type="s0:IssuerType"/>
        <xsd:attribute name="IssueInstant" type="timeInstant"/>
        </xsd:complexType>


    <xsd:complexType name="SubjectAssertionType">
        <xsd:complexContent>
            <xsd:extension base="s0:AssertionType">
                <xsd:sequence>
                    <xsd:element name="Subject" type="s0:SubjectType" minOccurs="1" maxOccurs="1"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <xsd:complexType name="AuthenticationAssertionType">
        <xsd:complexContent>
            <xsd:extension base="s0:SubjectAssertionType">
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <xsd:complexType name="AttributeAssertionType">
        <xsd:complexContent>
            <xsd:extension base="s0:SubjectAssertionType">
                <xsd:sequence>

                    <!-- the namespace should be any, but I'm doing this to make sure the parser validates at least
                        the namespace name -->
                    <xsd:any namespace="http://www.oasis.org/tbs/1066-12-25/s/" processContents="strict"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:element name="AuthorizationDecisionAssertion" type="s0:AuthorizationDecisionAssertionType"/>
    <xsd:complexType name="AuthorizationDecisionAssertionType">
        <xsd:complexContent>
            <xsd:extension base="s0:AssertionType">
                <xsd:sequence>
                    <xsd:element name="Decision" type="s0:DecisionType"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="AuthorizationAssertionType">
        <xsd:complexContent>
            <xsd:extension base="s0:SubjectAssertionType">
                <xsd:sequence>
                    <xsd:element name="Resource" minOccurs="1" type="string"/>
                    <xsd:element ref="s0:Permission" minOccurs="1" maxOccurs="unbounded"/>
                    <xsd:any namespace="##any" processContents="strict"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:simpleType name="DecisionType">
        <xsd:restriction base="string">
            <xsd:enumeration value="Permit"/>
            <xsd:enumeration value="Deny"/>
            <xsd:enumeration value="Indeterminate"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:element name="Permission" type="s0:PermissionType" abstract="true"/>

    <xsd:complexType name="PermissionType">
        <xsd:simpleContent>
            <xsd:restriction base="string"/>
        </xsd:simpleContent>
    </xsd:complexType>

    <xsd:element name="BasePermission" type="s0:PermissionBaseType" substitutionGroup="s0:Permission"/>
    <xsd:complexType name="PermissionBaseType">
        <xsd:simpleContent>
            <xsd:restriction base="string">
                <xsd:enumeration value="R"/>
                <xsd:enumeration value="W"/>
                <xsd:enumeration value="Use"/>
                <xsd:enumeration value="Admin"/>
            </xsd:restriction>
        </xsd:simpleContent>
    </xsd:complexType>

    <xsd:simpleType name="VersionType">
        <xsd:restriction base="string"/>
    </xsd:simpleType>
    <xsd:simpleType name="AssertionIDType">
        <xsd:restriction base="string"/>
    </xsd:simpleType>

    <xsd:simpleType name="RequestIDType">
        <xsd:restriction base="string"/>
    </xsd:simpleType>
    <xsd:simpleType name="IssuerType">
        <xsd:restriction base="string"/>
    </xsd:simpleType>
    <xsd:element name="Subject" type="s0:SubjectType"/>
    <xsd:complexType name="SubjectType">
        <xsd:sequence>
            <xsd:element name="CommonName" type="string" minOccurs="0"/>
            <xsd:element name="NameID" type="uriReference" minOccurs="0"/>
            <xsd:element ref="s0:Authenticator" minOccurs="0"/>
            <xsd:any namespace="##any" processContents="lax"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="Authenticator">
        <xsd:complexType>
```

```
                <xsd:sequence>
                    <xsd:element name="Protocol" type="string" minOccurs="0" maxOccurs="unbounded"/>
                    <xsd:element name="NameID" type="uriReference"/>
                    <xsd:element name="Authdata" type="string"/>
                    <xsd:element name="KeyInfo" type="string"/>
                    <!-- ds:KeyInfo"/> -->
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="Conditions" type="s0:ConditionsType"/>
        <xsd:complexType name="ConditionsType">
            <xsd:sequence>
                <xsd:element name="Audiences" type="string" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>

        <xsd:element name="Advice" type="s0:ConditionsType"/>
        <xsd:complexType name="AdviceType">
                <xsd:sequence>
                    <xsd:element name="Assertion" type="s0:AssertionType" minOccurs="0"
maxOccurs="unbounded"/>
                </xsd:sequence>
            </xsd:complexType>
</xsd:schema>
```

Sample Authorization Decision Assertion

```
<?xml version="1.0" encoding="UTF-8"?>
<s0:Assertion xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xmlns:s0="http://www.oasis.org/tbs/1066-12-25/"
xsi:schemaLocation="http://www.oasis.org/tbs/1066-12-25/ D:\AllMaterial\OASIS-Sec-TC\Assertions.xsd"
xsi:type="s0:AuthorizationDecisionAssertionType"
AssertionID="http://www.bizexchange.test/assertion/AE0221"
Issuer="URN:dns-date:www.bizexchange.test:2001-01-03:19283">
    <Decision>Deny</Decision>
</s0:Assertion>
```

Sample Attribute Assertion

```
<?xml version="1.0" encoding="UTF-8"?>
<s0:Assertion xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xmlns:s0="http://www.oasis.org/tbs/1066-12-25/"
xmlns:s1="http://www.oasis.org/tbs/1066-12-25/s/"
xsi:schemaLocation="http://www.oasis.org/tbs/1066-12-25/ D:\AllMaterial\OASIS-Sec-TC\Assertions.xsd"
xsi:type="s0:AttributeAssertionType"
AssertionID="http://www.bizexchange.test/assertion/AE0221"
 Issuer="URN:dns-date:www.bizexchange.test:2001-01-03:19283"
xmlns:someOtherNs="http://www.example.org/something">
<Subject>
    <NameID>mailto:Alice@bizex.test</NameID>
</Subject>
<s1:Role>Admin</s1:Role>

</s0:Assertion>
```

Sample Assertions Repository

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<s0:AssertionsPackage
xmlns:s0="http://www.oasis.org/tbs/1066-12-25/"
xmlns:s1="http://www.oasis.org/tbs/1066-12-25/s/"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:schemaLocation="http://www.oasis.org/tbs/1066-12-25/
D:\AllMaterial\OASIS-Sec-TC\sampleExtensions1.xsd
http://www.oasis.org/tbs/1066-12-25/s/
D:\AllMaterial\OASIS-Sec-TC\sampleExtensions2.xsd" >
<!-- Sample File, named SampleAuthorityAssertionsList.xml -->
<!-- Test file for executing SAML Queries against -->
<!-- This file would be a virtual file in a real system -->

<!-- The following extensions are shown: -->
<!-- 1. Custom attributues for a user, in a different namespace -->
<!-- 2. Customer required rights, in the same namespace -->

<!--ToDo: XMLSpy does not seem to validate the Any contents -->
    <Assertion xsi:type="s0:AttributeAssertionType">
        <Subject>
            <NameID>mailto:Alice@bizex.test</NameID>
        </Subject>
        <s1:Role xsi:type="s1:Role">Admin</s1:Role>
    </Assertion>
    <!-- Alice can Read and Write-->
    <Assertion xsi:type="s0:AuthorizationAssertionType">
        <Subject>
            <NameID>mailto:Alice@bizex.test</NameID>
        </Subject>
        <Resource>
            http://store.carol.test/finance
        </Resource>
        <s0:BasePermission>R</s0:BasePermission>
    </Assertion>

    <!-- Users with Role Admin can Admin the resource -->
    <Assertion xsi:type="s0:AuthorizationAssertionType">
        <Subject>
            <someOtherNs:Role>Admin</someOtherNs:Role>
        </Subject>
        <Resource>
            http://store.carol.test/finance
        </Resource>
        <s0:BasePermission>Admin</s0:BasePermission>

    </Assertion>
        <!-- Alice can Write -->
    <Assertion xsi:type="s0:AuthorizationAssertionType">
        <Subject>
            <NameID>mailto:Alice@bizex.test</NameID>
        </Subject>
        <Resource>
            http://store.carol.test/finance2
        </Resource>
        <s0:ExtensionPermission>Provision</s0:ExtensionPermission>

    </Assertion>
</s0:AssertionsPackage>
```

Sample Extensions #1 – sampleExtensions1.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xsd:schema targetNamespace="http://www.oasis.org/tbs/1066-12-25/"
xmlns="http://www.w3.org/2000/10/XMLSchema" xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
xmlns:s0="http://www.oasis.org/tbs/1066-12-25/" elementFormDefault="unqualified">
    <xsd:include schemaLocation="D:\AllMaterial\OASIS-Sec-TC\Assertions.xsd"/>

    <!-- Sample Extensions #1 shows an addition Permission -->
    <xsd:element name="ExtensionPermission" type="s0:PermissionExtensionType"
substitutionGroup="s0:Permission"/>
    <xsd:complexType name="PermissionExtensionType">
        <xsd:simpleContent>
            <xsd:restriction base="string">
                <xsd:enumeration value="Provision"/>
            </xsd:restriction>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:schema>
```

Sample Extensions #2 – sampleExtensions2.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.oasis.org/tbs/1066-12-25/s"
xmlns="http://www.w3.org/2000/10/XMLSchema"
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
xmlns:s1="http://www.oasis.org/tbs/1066-12-25/s"
elementFormDefault="unqualified">

<!-- sampleExtensions #2 shows a custom attribute, role -->
    <xsd:element name="Role">
    <xsd:simpleType>
        <xsd:restriction base="string">
            <xsd:enumeration value="User"/></xsd:restriction>
    </xsd:simpleType>
</xsd:element>

</xsd:schema>
```

Sample Request #1

```
<?xml version="1.0" encoding="UTF-8"?>
<s0:SAMLQuery xmlns:s0="http://www.oasis.org/tbs/1066-12-25/"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance" xsi:schemaLocation="http://www.oasis.org/tbs/1066-
12-25/ D:\AllMaterial\OASIS-Sec-TC\Assertions.xsd">
    <!-- example 2.1.4.  Can Alice read finance? -->
    <SAMLXQuery>
        <AssertionsPackage>
        FOR $S IN document("SampleAuthorityAssertionsList.xml")
        WHERE $S/Resource = "http://store.carol.test/finance"
        AND $S/Subject/NameID = "mailto:Alice@bizex.test"
        AND $S/Permission = "Admin"
        RETURN $S
        </AssertionsPackage>
    </SAMLXQuery>
```

```
</s0:SAMLQuery>
```

Sample Result #1

```
<?xml version="1.0" encoding="UTF-8"?>
<s0:AssertionsPackage xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xmlns:s0="http://www.oasis.org/tbs/1066-12-25/"

xsi:schemaLocation="http://www.oasis.org/tbs/1066-12-25/ D:\AllMaterial\OASIS-Sec-TC\Assertions.xsd">
    <!-- Example 2.1.5 -->
    <Assertion xsi:type="s0:AuthorizationDecisionAssertionType"
AssertionID="http://www.bizexchange.test/assertion/AE0221" Issuer="URN:dns-date:www.bizexchange.test:2001-
01-03:19283">
        <Decision>Permit</Decision>
    </Assertion>
</s0:AssertionsPackage>
```

Sample Request #2

```
<?xml version="1.0" encoding="UTF-8"?>
<s0:SAMLRequest
xmlns:s0="http://www.oasis.org/tbs/1066-12-25/"
xmlns:s1="http://www.oasis.org/tbs/1066-12-25/s/"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:schemaLocation="http://www.oasis.org/tbs/1066-12-25/ D:\AllMaterial\OASIS-Sec-TC\Assertions.xsd">
    <!-- example 2.1.4 Can Alice read finance with an attribute Assertion-->
    <SAMLXQuery>
        <AssertionsPackage>
            FOR $S IN document("SampleAuthorityAssertionsList.xml")
            WHERE $S/Resource = "http://store.carol.test/finance"
            AND $S/Subject/NameID = "mailto:Alice@bizex.test"
            AND $S/Permission = "READ"
        <Assertion>
            RETURN $S/Decision
        </Assertion>
        </AssertionsPackage>
    </SAMLXQuery>
        <s0:SubjectAssertionsPackage>
        <Assertion xsi:type="s0:AttributeAssertionType">
            <Subject>
                <NameID>mailto:Alice@bizex.test</NameID>
            </Subject>
            <s1:Role>Admin</s1:Role>
        </Assertion>
    </s0:SubjectAssertionsPackage>
</s0:SAMLRequest>
```

Sample Request #7

```
<?xml version="1.0" encoding="UTF-8"?>
<s0:SAMLQuery
xmlns:s0="http://www.oasis.org/tbs/1066-12-25/"
xmlns:s1="http://www.oasis.org/tbs/1066-12-25/s/"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:schemaLocation="http://www.oasis.org/tbs/1066-12-25/ D:\AllMaterial\OASIS-Sec-TC\Assertions.xsd">
    <!-- 7.   The requestor sends a description of an assertion that it would like the responder to locate, retrieve
and return.  If successful, the responder returns a success indication and an assertion that either exactly meets the
requirements or is more general.  If unsuccessful, the responder returns a failure indication and (optionally) one or
more assertions that are more specific than the one specified. The sample used is Alice trying to read finance, and
if she can't read finance or *, then return if she can read finance/f1-->
<!-- this example isn't quite right yet -->
    <SAMLXQuery>
        <AssertionsPackage>
```

```
                <Assertion>
                FOR $S IN document("SampleAuthorityAssertionsList.xml")
                WHERE ( $S/Resource = "http://store.carol.test/finance" OR $S/Resource =  "http://store.carol.test/*")
                AND $S/Subject/NameID = "mailto:Alice@bizex.test"
                AND $S/Permission = "READ"
                return $S/Decision
                IF $S/Decision != "YES" then
                      FOR $T IN document("SampleAuthorityAssertionsList.xml")
                      WHERE $T/Resource = "http://store.carol.test/finance/f1"
                      AND $T/Subject/NameID = "mailto:Alice@bizex.test"
                      AND $T/Permission = "READ"
                      IF $T/Decision = "YES" then return
                            $T/Decision
                </Assertion>
          </AssertionsPackage>

    </SAMLXQuery>
</s0:SAMLQuery>
```

Appendix:

Discussion of Xquery

(Following are notes by Dave on advantages of XQuery.)

The key benefits to using XQuery are:

- Generic syntax, which allows for tighter cardinalities in SAML domain model (these 2 are linked)
- Arbitrary return values, no need for a responds element.
- Arbitrary searches and results including wildcards and booleans.
- The ability to add new queries without revving the server software. This pushes the ability to change the queries to the client.
- The Assertions class diagram is simplified as the assertions are for facts only, rather than queries.

The disadvantages of Xquery are:

- Developers have to learn another specification rather than just saml
- The Xquery syntax is too general for the queries that SAML needs, a very restrictive and simple syntax would be adequate.
- Implementations are going to have to map Xquery syntax onto their own repositories
- The xml syntax for Xquery is quite verbose and difficult.

The response to the disadvantages:

- The developers are going to have to learn a syntax anyways, why not use an industry standard one with tooling and high probability of developer knowledge re-use.
- It seems that many people want complex queries and also we don't want to overly restrict the queries allowed. Should it happen that the requests/queries are very general, than this might be revisited.
- Implementations are going to have to map Xquery or any other syntax onto their repositories. Wouldn't mapping a general syntax rather than a specific syntax be easier for vendors?
- The xml syntax for xquery is verbose, but probably any kind of general syntax will be verbose. Xquery has these as issues before it. Presumably they will be better equipped to create a simple xml syntax for queries than SAML will be.

IMHO, the biggest advantage of the use of XQuery is that it decouples the clients from the servers from a query perspective. New queries from the client can be added without requiring a spec and server software change. It allows extensibility from the clients. It pushes the ability to change queries from the server to the client. Under the PHB model, any time we want to modify a query, we have to update the protocol (particularly the responds element), the client and the server. Using Xqueries, just the client gets updated.

So the big question is: do we want strongly-typed queries, meaning the spec & software get reved every time there's a new query, or do we want weakly-typed queries.

There are 2 alternatives to Xquery:

1. a generic assertion/claim like PHB has, with a results element.
2. Subtype each of the items in the class diagram for an input query, making the cardinalities optional.

One of the reasons why the PHB style claim is so open-ended, is so that it can be used as a template for the query. Say you want to find an authorization assertion (OM model) for a given subject/object combination. It's got both subject, object, and permission. Now Permission is required in OM model. In PHB model, Permission is optional. The reason is so that you can leave permission blank in a PHB query. This is the whole point about cardinalities, that in phb's model you can never have cardinalities (as they might be left blank for the query) whereas in OM you can because they are just used for the return.

Now you could model it as a set of AssertionTemplates with no cardinalities, and then subtype to Assertion with cardinalities, but that adds even more types. (option #2)

Further, because of the template model, you have no control over the operators. Phil has been desperately wanting wildcards, and this gives it.

Take sample query #8, if SAML does not support this operation exactly, then a rev of the SERVER will have to happpen to add the query mechanism. With XQuery, you can simply change the query that you send. So it gives Clients much more flexibility

Another example is #7. Now this is easier to code up in XQuery than adding some new parameter (to say which extra specifications are to be used in the unsuccessful case) to the responds element.

Another reason why query is good is because there is no need to create a responds element. The whole point of the responds element is that it specifies what the requestor wants returned. But that means that the types of responses are rigidly defined. There is one out with the use of a schema URI, but that seems a strange way to do it. It also doesn't cover the if/then/else style of return decision. With XQuery you can return any part of the results that you've found, like just the Decision or the found Assertions or whatever.

Schema Extension Techniques

(Following are notes by Dave on how to do the extension of **Permission** values.)

Trying to get extension in the Permissions has been many hours, and ultimately I resorted to a technique I didn't really like.

The method that finally worked was Method 1(typeExtension) in the same namespace:

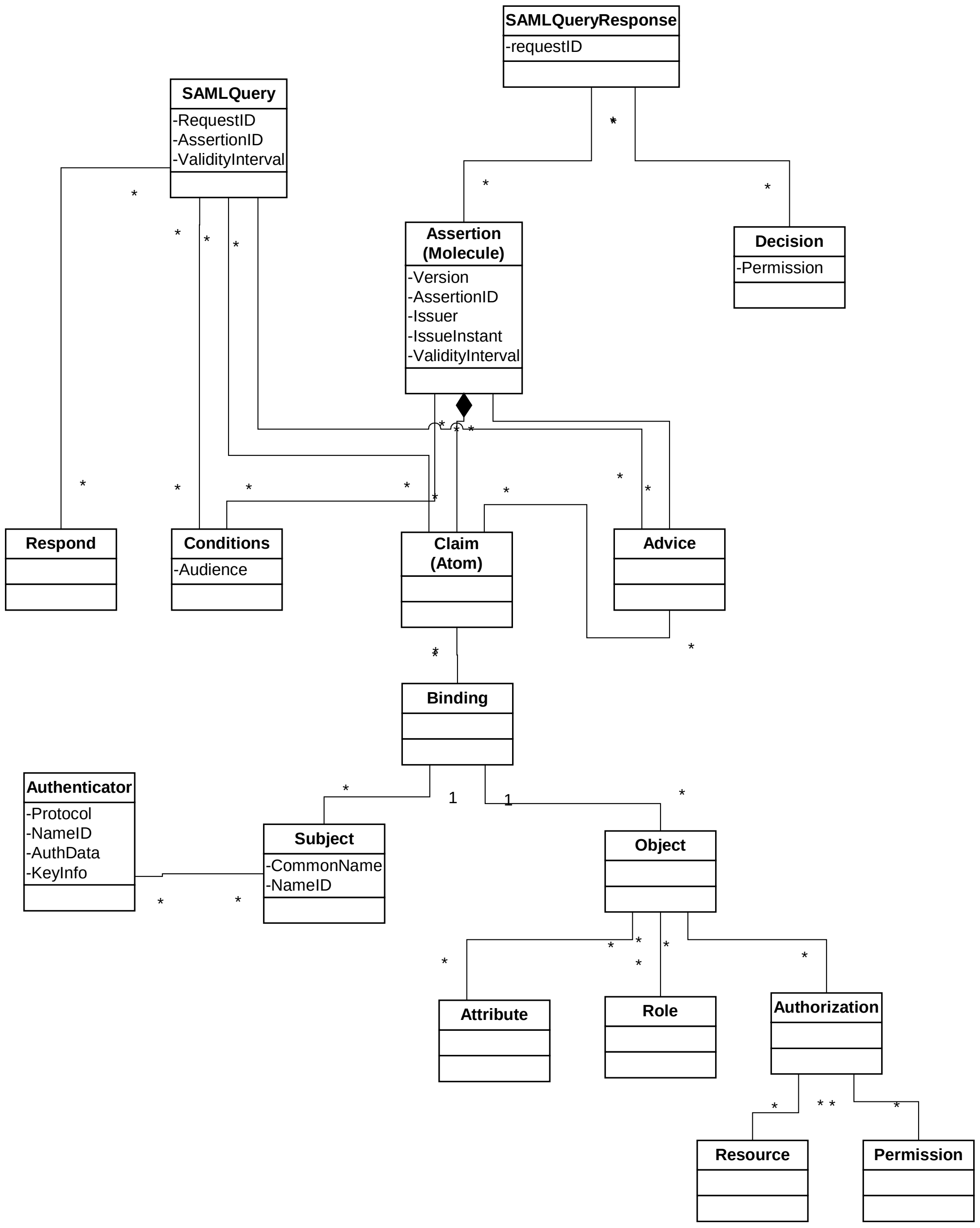
```
<PermissionList>
<BasePermission>R</BasePermission>
<ExtendedPermission>Provision</BasePermission>
```

The options for adding a Permission type, say Provision, to Assertion are:

- Extend the set of names allowed in an enumeration List - <Permissions>R W Provision</Permissions>. This doesn't work because the enumeration value space can't be extended.
- specification of different namespaced elements -
  <PermissionList>
  <s0:Permission>W</s0:Permission>
  <s1:Permission>Provision</s1:Permission>.
  I can't recall why this didn't work
- method 4 (dangling namespace) from xfront.
  <PermissionList>
  <Permission>W</Permission>
  <Permission>Provision</Permission>
  XMLSpy illegally follows the namespace declaration in the include.
- Method 3 (abstract base type with type substitution) from xfront
  <PermissionList>
  <Permission xsi:type="s0:PermissionBaseType">W</Permission>
  <Permission xsi:type="s1:PermissionExtensionType">Provision</Permission>
  XMLSpy gives the dreaded internal error on this case, I think because the Permission is a simpleContent.
- Method 1(typeExtension) in different namespaces
  <PermissionList>
  <s0:BasePermission>R</s0:BasePermission>
  <s1:ExtendedPermission>Provision</s1:BasePermission>

PHB/Core0.7 Class diagram

The following is a class diagram representing Core 0.7


SAMLQueryResponse
-requestID
SAMLQuery
-RequestID
-AssertionID
-ValidityInterval
Assertion (Molecule)
-Version
-AssertionID
-Issuer
-IssueInstant
-ValidityInterval
Decision
-Permission
Respond
Conditions
-Audience
Claim (Atom)
Advice
Binding
Authenticator
-Protocol
-NameID
-AuthData
-KeyInfo
Subject
-CommonName
-NameID
Object
Attribute
Role
Authorization
Resource
Permission
1
1