SAML Specification

draft-sstc-ftf3-saml-spec-00.doc
20 June 2001

This document consists of:

draft-sstc-use-domain-05.doc
draft-sstc-core-09.doc
draft-orchard-maler-assertion-00.doc
draft-sstc-protocols-00.doc
draft-sstc-protocol-model-00.doc
samlconformance-clause-06.doc

NOTATION 5

DOMAIN MODEL: INTRODUCTION 5

STATIC MODEL 5
GLOSSARY (ABRIDGED): 6
PRODUCER CONSUMER MODEL 7
CHANGES FROM PRIOR VERSION OF DOMAIN MODEL 9
HAL LOCKHART 'S NOTES ON THIS VERSION OF DOMAIN MODEL 9

XML ASSERTION AND REQUEST SYNTAX 11

NAMESPACES 11
SAML ASSERTION 11
Element <SAMLAssertionPackage> 12
CLAIMS 13
Element <Claims> 13
Element <AssertionRef> 14
Element <Subject> 14
Element <Authenticator> 15
Element <DecisionClaim> 15
Element <AuthenticationClaim> 16
Element <AttributeClaim> 16
Element <ExtendedAttributeClaim> 16
Element <AuthorizationClaim> 17
CONDITIONS 17
Element <Conditions> 17
Element <AudienceRestrictionCondition> 18
ADVICE 19
Element <Advice> 19
SAML PROTOCOL 19
Element <SAMLQuery> 19
Element <Respond> 20
Element <SAMLQueryResponse> 21
SCHEMA EXTENSION 21

ALTERNATE ASSERTION STRUCTURE PROPOSAL 22

INTRODUCTION 22
DEFINITIONS 22
SECTION CONVENTIONS 22
XML DESIGN PRINCIPLES 22
SAML MESSAGE ARCHITECTURE 23
SAMLRequest Element 24
SAMLXQuery Element 25
SubjectAssertionsPackage Element 25
SAMLResponse Element 26
AssertionsPackage Element 26
INDIVIDUAL ASSERTION STRUCTURES 27
AttributeAssertion Element 27
AuthenticationAssertion Element 27
AuthorizationAssertion Element 27
AuthorizationDecisionAssertion Element 28
SUBJECT ELEMENT 28

Summary of Extensibility Features 28
Summary of Differences from core -07 29
Request Methods 29
W3C XML Schema Design principles 30
Schema and Example Documents 30
Complete Assertions Schema 30
Sample Authorization Decision Assertion 33
Sample Attribute Assertion 33
Sample Assertions Repository 34
Sample Extensions #1 – sampleExtensions1.xsd 35
Sample Extensions #2 – sampleExtensions2.xsd 35
Sample Request #1 35
Sample Result #1 36
Sample Request #2 36
Sample Request #7 36
Discussion of Xquery 38
Schema Extension Techniques 40
PHB/Core0.7 Class diagram 41

SAML REQUEST/RESPONSE PROTOCOLS 42

Protocol Model 45
Protocol exchanges 48
Principal-centered direct protocol 48
Principal-centered indirect protocol 49
Pull protocol 50
Push protocol 50
Primary domain session-close protocol 51
Secondary domain session-close protocol 52
Data structures 52
AuthnNotification 53
AuthnAcknowlegment 53
AuthnRequest 53
AuthnResponse 54
AuthnQuery 54
AuthnResult 55
AuthzNotification 55
AuthzAcknowlegment 55
AuthzRequest 56
AuthzResponse 56
AuthzQuery 56
AuthzResult 57
SessionNotification 57
SessionAcknowlegment 58
SessionRequest 58
SessionResponse 59
SessionQuery 59
SessionResult 59
Protocol Security considerations 60

CONFORMANCE 61

The SAML Conformance Clause 61
Conformance Nomenclature 61
Mandatory/Optional: 62
Extensions: 62
Alternate approaches 62
Authorities 62

Roles 63
Bindings 63
SAML Conformance Program 63
Things To Do (Conformance) 64

REFERENCES 65

Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in Key Words for Use in RFC's to Indicate Requirement Levels (RFC 2119).

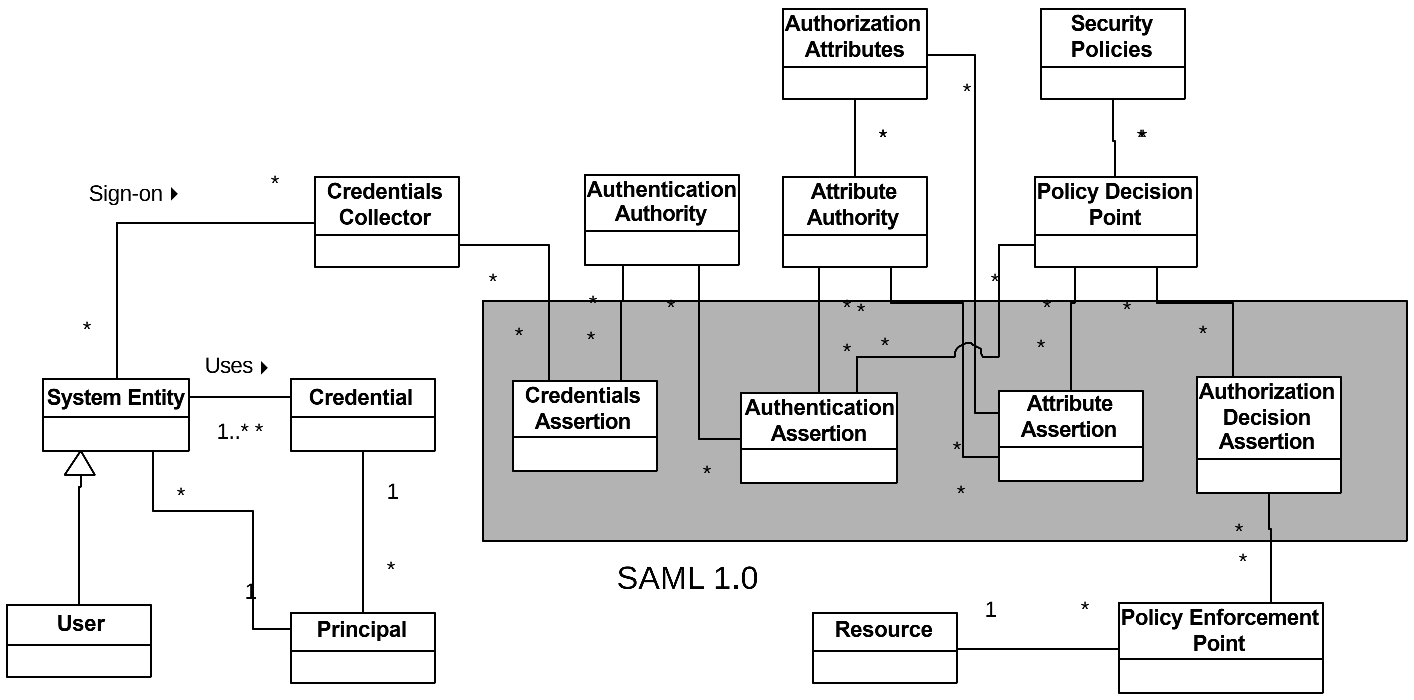
Domain Model: Introduction

This domain model provides a description and categorization of the domain that SAML solves problems in. People, software, data, interactions, and behavior are described in the abstract, without binding the specification to a particular implementation. It provides a standardized or normalized description of concepts for the purposes of further discussion in requirements, use-cases, etc. It covers material out-of-scope for the specification in order to show the context that the specification solves problems in. It does not describe implementation information such as API details, Schema definitions and data representations.

A typical use-case for this document is: "We all agree what we mean by term x and how entity y creates it and entity z consumes it. Is x in scope or out of scope for SAML?". Another use case "We have created an OASIS TC committee on functionality A. A is the standardization of term x that is out of scope for SAML".

In the rational unified process, an artifact we are working on is the logical view, http://www.rational.com/products/whitepapers/350.jsp#RTFToC2.

Static Model



ISSUES:

- Should there be a 1:1 relationship between credential and credential assertion, perhaps labeled represents?
- Should all the assertions relationships be 1:* to the authorities to represent that a given assertion can only be produced by 1 given authority, or left as *:* to represent that a given assertion can be produced by many authorities.
- Should there be explicit (perhaps *:*) relationships between the authorities?
- What names for relationships should be used?

Glossary (abridged):

Notation: Definitions that have been agreed upon by the use case subgroup are denoted(Conf)

Assertion: TBD

Attribute Authority: (Conf) A system entity that produces Attribute assertions, based upon TBD inputs.

Attribute Assertion: An assertion about attributes of a principal.

Authentication – (from glossary with principal added) (Conf) Authentication is the process of confirming an entity's asserted principal identity with a specified, or understood, level of confidence. [7]
The process of verifying a principal identity claimed by or for a system entity. [12]

Authentication Assertion: Data vouching for the occurrence of an authentication of a principal at a particular time using a particular method of authentication. Synonym(s): name assertion.

Authentication Authority: (Conf) A system entity that verifies credentials and produces authentication assertions

Authorization Attributes: (Conf) Attributes about a principal which may be useful in an authorization decision (group, role, title, contract code,...).

Authorization Decision Assertions: (from glossary) In concept an authorization assertion is a statement of policy about a resource, such as:
the user "noodles" is granted "execute" privileges on the resource "/usr/bin/guitar."

Credential: (Conf) Data that is transferred or presented to establish a claimed principal identity.

Policy Decision Point: (from glossary, access control decision) The place where a decision is arrived at as a result of evaluating the requester's identity, the requested operation, and the requested resource in light of applicable security policy. (surprisingly enough, not explicitly defined in [10])

Policy Enforcement Point: (from glossary, access enforcement function) The place that is part of the access path between an initiator and a target on each access control request and enforces the decision made by the Access Decision Function [10].

Principal, or Principle Identity: (Conf) An instantiation of a system entity within the security domain.

Resource: (from glossary) Data contained in an information system (e.g. in the form of files, info in memory, etc); or a service provided by a system; or a system capability, such as processing power or communication bandwidth; or an item of system equipment (i.e., a system component--hardware, firmware, software, or documentation); or a facility that houses system operations and equipment. (definition from [1])

Security Domain: TBD

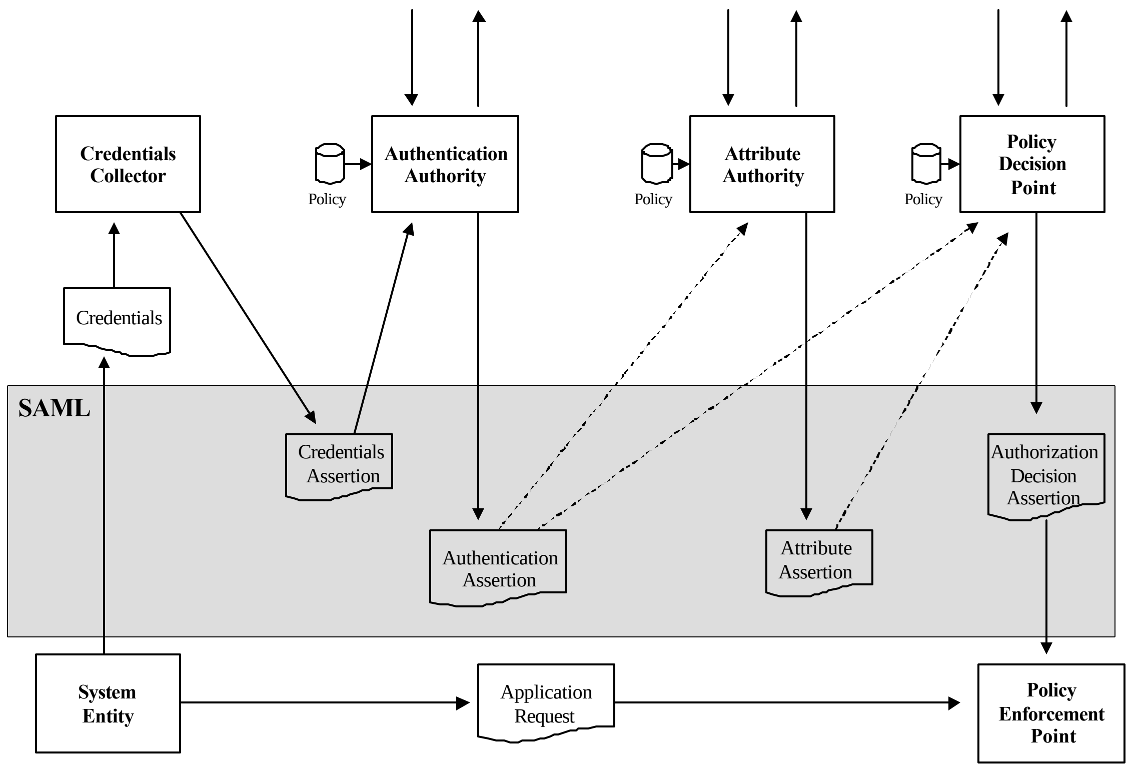
Security Policies: (from glossary) A set of rules and practices specifying the "who, what, when, why, where, and how" of access to system resources by entities (often, but not always, people).

Sign-on: The process of presenting credentials to an authentication authority for requesting access to a resource

System Entity: (from glossary) (Conf) An active element of a system--e.g., an automated process, a subsystem, a person or group of persons--that incorporates a specific set of capabilities. (definition from [1])

User: (Conf) A human individual that makes use of resources for application purposes. This may also be non-human such as parties and processes.

Producer Consumer model



This diagram provides a view of the elements of the SAML problem space that is focused on the architectural entities and their inputs and outputs. Its main purpose is to achieve a sufficient commonality of understanding the meanings of the various terms used to allow productive discussion. The names have been chosen either to be consistent with standard usage in the field or suggestive of their purpose or action, in many cases their exact nature or contents are not fully agreed upon. Although the diagram is intended to be neutral on the SAML design, the choice of which elements to include and which to leave out anticipates likely elements of the design.

This diagram should **not** be interpreted as describe message flows or a single processing flow. It merely attempts to describe which entities are capable of producing certain outputs and which entities may make use of certain inputs. For example, all of the following are consistent with this diagram:

- A PDP collects various assertions from their sources in order to make a policy decision
- An Attribute Assertion is returned to the System Entity that initiated the interaction (lower left) who presents it as required
- A PDP makes a decision without the use of any assertions

All of the entities shown may be a part of distinct security domains, or some of them may be in the same domain. Typically there will only be two or three security domains involved. Common groupings include:

- Combined Authentication Authority and Attribute Authority
- Combined PEP and PDP
- All combined except for PEP

Many of the components can have multiple instances. For example, there can be multiple Attribute Authorities or multiple PDPs. This may introduce relationships not shown in the diagram, for example, a PDP might provide assertions to another PDP.

There is an asymmetry between input and output. The outputs that are standardized have the names shown, by definition. The entities may or may not use the inputs identified for any particular action. This is represented by the use of solid and dashed lines respectively.

The entities that have an associated policy store, are assumed to use that policy to modulate the outputs they produce. This policy store is assumed to be non-volatile and capable of being administered in some way. The unlabeled arrows at the top represent other inputs and outputs, not specified by SAML. For inputs these fall into two categories: 1) inputs which have the same semantics as SAML defined Assertions, but are in unspecified format and 2) items which are not specified by SAML at all. An example of #1 is an X.509 Attribute Certificate. An example of #2 is the current date and time.

The diagram anticipates the design of SAML by identifying only the security assertions that could be output by these entities. SAML will also have protocol messages to send and receive these assertions and will make use of existing communications protocols to transmit these assertions.

The central gray box labeled SAML indicates which assertions **may be** specified by SAML. In particular, the inclusion of Credentials Assertions and Sessions Assertions has not been settled.
The definitions of these items can be found elsewhere.

The following comments cover points that may not be completely evident.

The System Entity in the diagram is the one requesting some action that will ultimately be permitted or denied. As a preliminary step it may provide credentials to authenticate itself.
The Credentials are not merely limited to a password, but might involve a sequence of messages exchanges, for example in a Public Key authentication protocol.

The Credentials Collector is an entity that can front-end the authentication process and pass to the Authentication Authority the information necessary for it to authenticate the System Entity. This is similar to the functionality provided by the RADIUS protocol.

The Authorization Decision Assertion might simply provide a yes/no response, or it might provide specific information about why access is denied, or it might provide statements of policy.

The Policy Enforcement Point is defined to have no policy, but to act directly on the contents of the Authorization Decision Assertion.

Changes from Prior Version of Domain Model

- Converted diagram from Together to Visio. This should make it more readable. I don't think powerpoint is an effective engineering diagram tool for the details that we want to represent, imho.
- Removed Sessions
- Changed authorization assertion to Attribute assertion
- Added indicator (grey area) to show SAML.
- removed reference to life cycle management
- made sure terminology between prod/cons model matches
- set principal/entity cardinalities to 1 to represent that a principal represents 1 entity
- set credential/principal cardinality to 1 to represent that a credential represents 1 principal
- set resource/PEP cardinality to 1 to represent that a given resource is policed by 1 PEP
- cardinalities all represented, most currently at *.   I need specific feedback on each of the links hence...
- I added a number of ISSUES on cardinality and relationships to the static model. Feedback would be great.
- Updated definition of User in static model glossary
- Removed Authorization Assertion from glossary
- Removed log-off from glossary
- Removed Session from the pubcon model.

Hal Lockhart's Notes on this Version of Domain Model

I did not understand the following or wasn't sure exactly what to do from the various minutes:
- what to do about authorization attributes. I noted some tendency to want to remove, but it seems to me that the association between attributes and the authorization authority seems relevent. Need resolution on keep or remove.
- The mention of a life-cycle model or diagram. I wasn't sure if this was meant to be a UML state transition diagram (assertion created, revoked, etc), a UML sequence diagram, a yourdon data flow diagram.
- The mention that the domain model has containment and "other" relationships. There are no containment/aggregation relationships listed, only a single inheritance (isa) relationship. So this confused me and I did nothing.

- I wasn't sure what to do about the domain glossary. I recall discussion about nuking it, but I didn't see any particular action to that regard.
- I didn't see a decision to change security policies.

XML Assertion and Request Syntax

Note: this section corresponds to draft-sstc-core-08.doc.

Namespaces

In this document, certain namespace prefixes represent certain namespaces.

All SAML protocol elements are defined using XML schema **[XML-Schema1][XML-Schema2]**. For clarity unqualified elements in schema definitions are in the XML schema namespace:

```
        xmlns="http://www.w3.org/2001/XMLSchema"
```

References to Security Assertion Markup Language schema defined herein use the prefix “s0” and are in the namespace:

```
    xmlns:s0="http://www.oasis.org/tbs/1066-12-25/"
```

This namespace is also used for unqualified elements in message protocol examples.

The SAML schema specification uses some elements already defined in the XML Signature namespace. The “XML Signature namespace” is represented by the prefix `ds` and is declared as:

```
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
```

The “XML Signature schema” is defined in **[XML-SIG-XSD]** and the `<ds:KeyInfo>` element (and all of its contents) are defined in **[XML-SIG]**§4.4.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
            targetNamespace="http://www.oasis.org/tbs/1066-12-25/"
            xmlns="http://www.w3.org/2001/XMLSchema"
            xmlns:s0="http://www.oasis.org/tbs/1066-12-25/"
            xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
            elementFormDefault="unqualified">
```

SAML Assertion

SAML specifies several different types of assertion for different purposes, these are:

Authentication Assertion
An authentication assertion asserts that the issuer has authenticated the specified subject.

Attribute Assertion
An attribute assertion asserts that the specified subject has the specified attribute(s). Attributes may be specified by means of a URI or through an extension schema that defines structured attributes.

Decision Assertion
A decision assertion reports the result of the specified authorization request.

Authorization Assertion
An authorization assertion asserts that a subject has been granted specific permissions to access one or more resources.

The different types of SAML assertion are encoded in a common XML package, which at a minimum consists of:

Basic Information.

Each assertion MUST specify a unique identifier that serves as a name for the assertion. In addition an assertion MAY specify the date and time of issue and the time interval for which the assertion is valid.

Claims.

The claims made by the assertion. This document describes the use of assertions to make claims for Authorization and Key Delegation applications.

In addition an assertion MAY contain the following additional elements. An SAML client is not required to support processing of any element contained in an additional element **with the sole exception that an SAML client MUST reject any assertion containing a Conditions element that is not supported.**

Conditions.

The assertion status MAY be subject to conditions. The status of the assertion might be dependent on additional information from a validation service. The assertion may be dependent on other assertions being valid. The assertion may only be valid if the relying party is a member of a particular audience.

Advice.

Assertions MAY contain additional information as advice. The advice element MAY be used to specify the assertions that were used to make a policy decision.

The SAML assertion package is designed to facilitate reuse in other specifications. For this reason XML elements specific to the management of authentication and authorization data are expressed as claims. Possible additional applications of the assertion package format include management of embedded trust roots **[XTASS]** and authorization policy information **[XACML]**.

Element `<SAMLAssertionPackage>`

The `<SAMLAssertionPackage>` element is specified by the following schema:

```
<element name="SAMLAssertionPackage" type="S0:SAMLAssertionPackageType">

<complexType name="SAMLAssertionPackageType">
      <!-- Basic Information -->
   <attribute name="Version"           type="string"/>
   <attribute name="AssertionID"       type="uriReference"/>
   <attribute name="Issuer"            type="string"/>
   <attribute name="IssueInstant"      type="timeInstant"/>
   <attribute name="NotBefore"         type="timeInstant"/>
   <attribute name="NotOnOrAfter"      type="timeInstant"/>

   <element name="Claims"     type="s0:Claims"      minOccurs="0"/>
   <element name="Conditions" type="s0:Conditions"  minOccurs="0"/>
   <element name="Advice"     type="s0:Advice"      minOccurs="0"/>
</complexType>
```

Six basic information attributes are defined; a protocol version identifier, a unique assertion identifier, an issuer identifier, the time instant of issue, the bounds of the validity interval.

Attribute `Version`

Each assertion MUST specify the SAML version identifier. The identifier for this version of SAML is the string "`1.0`".

Attribute `AssertionID`

Each assertion MUST specify exactly one unique assertion identifier. All identifiers are encoded as a Uniform Resource Identifier (URI) and are specified in full (use of relative identifiers is not permitted).

The URI is used as a *name* for the assertion and not as a *locator*. For the purposes of the SAML protocol it is only necessary to ensure that no two assertions share the same identifier. Provision of a service to resolve an identifier into an assertion is not a requirement but applications MAY specify a URL as the assertion identifier that MAY resolve to the assertion.

Attribute `Issuer`

The `Issuer` attribute specifies the issuer of the assertion by means of a URI.

Attribute `IssueInstant`

The `IssueInstant` attribute specifies the time instant of issue in Universal Coordinated Time (UTC).

Attribute `NotBefore` and `NotOnOrAfter`

The `NotBefore` and `NotOnOrAfter` attributes specify limits on the validity of the assertion.

The `NotBefore` attribute specifies the time instant at which the validity interval begins. The `NotOnOrAfter` attribute specifies the time instant at which the validity interval has ended

The `NotBefore` and `NotOnOrAfter` attributes are optional. If the value is either omitted or equal to the start of the epoch it is unspecified. If the `NotBefore` attribute is unspecified the assertion is valid at any time before the time instant specified by the `NotOnOrAfter` attribute. If the `NotOnOrAfter` attribute is unspecified the assertion is valid from the time instant specified by the `NotBefore` attribute with no expiry. If neither attribute is specified the assertion is valid at any time.

In accordance with the XML Schemas Specification, all time instances are interpreted in Universal Coordinated Time unless they explicitly indicate a time zone. Implementations MUST NOT generate time instances that specify leap seconds.

Claims

Element `<Claims>`

The `<Claims>` element contains one or more SAML assertion claims elements of type `<AbstractClaimType>`.

In each case if more than one assertion claim element is specified the validity of each claim is asserted jointly and severally, that is the semantics of a single assertion containing two claims are identical to the semantics of two separate assertions each of which contain one of the claims.

The following schema defines the `<Claims>` element:

```
<element name="Claims">
   <complexType>
      <sequence>
         <element name="AbstractClaim" type="s0:AbstractClaimType"
               minOccurs="0" maxOccurs="unbounded"/>
     </sequence>
  </complexType>
</element>

<complexType name="AbstractClaimType" abstract="true">
   <sequence>
      <element name="AssertionRef"     type="s0:AssertionRef"/>
      <!-- To add conditions on a per claim basis add :
      <element name="Conditions" type="s0:Conditions"  minOccurs="0"/>
          -->
   </sequence>
</complexType>
```

Element `<AssertionRef>`

The `<AssertionRef>` element specifies the assertion identifier of a prior assertion that has been used to generate the assertion in which the `<AssertionRef>` element occurs.

The primary purpose of `<AssertionRef>` elements is to permit auditing of SAML applications. As such an `<AssertionRef>` element is advisory only and does not mandate any specific action on the part of the application (such as tracking validity dependencies).

The following elements may include `<AssertionRef>` elements:

AbstractClaimType
Advises that the specified claim was derived from the specified assertion.

Subject
Advises that the Subject definition of the claim was derived from the specified assertion.

Advice
Advises that the referenced assertion was used to derive some unspecified portion of the assertion.

The following schema defines the `<AssertionRef>` element:

```
<complexType name="URIReferenceType">
   <attribute name="id" type="uriReference">
</complexType>

<element name="AssertionRef" type="s0:URIReferenceType">
```

Element `<Subject>`

The `<Subject>` element specifies the subject of the binding. In every case the subject of a SAML assertion binding is a principal. A principal MAY be identified by name and/or by reference to authentication credentials. The following forms of subject name are supported:

Element	Description
`<CommonName>`	An unstructured text string, for example “Alice Aardvark”.
`<NameID>`	A URI that specifies the principal by means of a machine-

	readable identifier.
`<Authenticator>`	Specifies credentials and an authentication protocol by which the subject may be identified.
`<AssertionRef>`	Specifies that the contents of the `<Subject>` element were derived from the specified assertion.
`<AbstractSubject>`	Extension schema…

In addition the principal MAY be specified by reference to authentication credentials by means of the `<Authenticator>` element.

The following schema defines the `<Subject>` element:

```
<element name="Subject">
   <complexType>
      <sequence>
         <element name="CommonName"       type="string"/>
         <element name="NameID"           type="s0:URIReferenceType"/>
         <element name="Authenticator"    type="s0:Authenticator"/>
         <element name="AssertionRef"     type="s0:AssertionRef"/>
         <element name="AbstractSubject"
                                         type="s0:AbstractSubjectType"/>
     </sequence>
  </complexType>
</element>

<complexType name="AbstractSubjectType" abstract="true"/>
```

Element `<Authenticator>`

The `<Authenticator>` element specifies a means of identifying the subject of the binding by means of their authentication credentials.

The authentication credentials MAY be specified either by means of the XML Digitial Signature `<ds:KeyInfo>` element or by means of the `<Authdata>` element. Applications SHOULD make use of the `<ds:KeyInfo>` element for credentials that it supports. Applications MAY use the `<Authdata>` element to specify other types of authentication credentials, including passwords.

The `<Authenticator>` element MAY specify one or more `<Protocol>` elements. If present the `<Protocol>` elements specify the authentication algorithms with which the authentication credentials MAY be used to obtain an acceptable authentication.

The following schema defines the `<Authenticator>` element:

```
<element name="Authenticator">
   <complexType>
      <sequence>
         <element name="Protocol" type="uriReference"
               minOccurs="0" maxOccurs="unbounded"/>
         <element name="Authdata" type="string"/>
         <element name="KeyInfo" type="ds:KeyInfo"/>
      </sequence>
   </complexType>
</element>
```

Element `<DecisionClaim>`

The `<DecisionClaim>` element asserts that the access permissions specified in the request identified by the corresponding RequestID were either permitted, denied or could not be determined.

The following schema defines the `<DecisionClaim>` element:

```
<complexType name="DecisionClaim">
   <complexContent>
      <extension base="s0:AbstractClaimType">
         <attribute name="Decision" type="s0:DecsionType"/>
      </extension>
   </complexContent>
</complexType>

<simpleType name=DecisionType>
   <restriction base="string">
      <enumeration value="Permit"/>
      <enumeration value="Deny"/>
      <enumeration value="Indeterminate"/>
   </restriction>
</simpleType>
```

Element `<AuthenticationClaim>`

The `<AuthenticationClaim>` element asserts that the specified subject has been authenticated.

The following schema defines the `<AuthenticationClaim>` element:

```
<complexType name="AuthenticationClaim">
   <complexContent>
      <extension base="s0:AbstractClaimType">
         <sequence>
            <element name="Subject"          type="s0:Subject"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>
```

Element `<AttributeClaim>`

The `<AttributeClaim>` element asserts that a specified subject has the specified attribute(s) specified by a URI.

The following schema defines the `<AttributeClaim>` element:

```
<complexType name="AttributeClaim">
   <complexContent>
      <extension base="s0:AbstractClaimType">
         <sequence>
            <element name="Subject" type="s0:Subject"/>
            <element name="AttributeID" type="s0:URIReferenceType"
                     minOccurs="0" maxOccurs="unbounded"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>
```

Element `<ExtendedAttributeClaim>`

The `<ExtendedAttributeClaim>` element asserts a relationship between the specified subject and a collection of attributes specified by means of an extension schema.

The following schema defines the `<ExtendedAttributeClaim>` element:

```
<complexType name="ExtendedAttributeClaim">
   <complexContent>
      <extension base="s0:AbstractClaimType">
         <sequence>
            <element name="Subject" type="s0:Subject"/>
            <element name="Attribute" type="s0:AbstractAttributeType"
                     minOccurs="0" maxOccurs="unbounded"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>

<complexType name="AbstractAttributeType" abstract="true"/>
```

Element `<AuthorizationClaim>`

The `<AuthorizationClaim>` element asserts that the specified subject is authorized to perfom the specified operation(s)on the specified resource(s).

Defined permissions are Read, Write, Execute, Delete and Control. Additional permissions may be specified by URI through an `<ExtendedPermissions>` element.

The following schema defines the `<AuthorizationClaim>` element:

```
<complexType name="AuthorizationClaim">
   <complexContent>
      <extension base="s0:AbstractClaimType">
         <sequence>
            <element name="Subject"          type="s0:Subject"/>
            <element name="Authorization"    type="s0:Authorization"
               minOccurs="0" maxOccurs="unbounded"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>

<element name="Authorization">
   <complexType>
      <sequence>
         <element name="Resource" type="uriReference"
               minOccurs="0" maxOccurs="unbounded"/>
         <element name="Permission" type="s0:PermissionType"
               minOccurs="0" maxOccurs="unbounded"/>
         <element name="ExtendedPermission" type="s0:URIReferenceType"
               minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
   </complexType>
</element>

<simpleType name=PermissionType>
   <restriction base="string">
      <enumeration value="Read"/>
      <enumeration value="Write"/>
      <enumeration value="Execute"/>
      <enumeration value="Delete"/>
      <enumeration value="Control"/>
   </restriction>
</simpleType>
```

Conditions

Element `<Conditions>`

Assertion Conditions are contained in the `<Conditions>` element. SAML applications MAY define additional elements using an extension schema. If an application encounters an element contained within a `<Conditions>` element that is not understood the status of the Condition MUST be considered Indeterminate.

The following schema defines the `<Conditions>` element:

```
<element name="Conditions">
   <complexType>
      <sequence>
         <element name="AbstractCondition"
               type="s0:AbstractConditionType"
               minOccurs="0" maxOccurs="unbounded"/>
     </sequence>
  </complexType>
</element>

<complexType name="AbstractConditionType" abstract="true"/>
```

Element <AudienceRestrictionCondition>

Assertions MAY be addressed to a specific audience. Although a party that is outside the audience specified is capable of drawing conclusions from an assertion, the issuer explicitly makes no representation as to accuracy or trustworthiness to such a party.

- Require users of an assertion to agree to specific terms (rule book, liability caps, relying party agreement)
- Prevent clients inadvertently relying on data that does not provide a sufficient warranty for a particular purpose
- Enable sale of per-transaction insurance services.

An audience is identified by a URI that identifies to a document that describes the terms and conditions of audience membership.

Each client is configured with a set of URIs that identify the audiences that the client is a member of, for example:

`http://cp.verisign.test/cps-2000`
Client accepts the VeriSign Certification Practices Statement

`http://rule.bizexchange.test/bizexchange_ruebook`
Client accepts the provisions of the *bizexchange* rule book.

An assertion MAY specify a set of audiences to which the assertion is addressed. If the set of audiences is the empty set there is no restriction and all audiences are addressed. Otherwise the client is not entitled to rely on the assertion unless it is addressed to one or more of the audiences that the client is a member of. For example:

`http://cp.verisign.test/cps-2000/part1`
Assertion is addressed to clients that accept the provisions of a specific part of the VeriSign CPS.

In this case the client accepts a superset of the audiences to which the assertion is addressed and may rely on the assertion.

The following schema defines the <AudienceRestrictionCondition> element:

```
<complexType name="AudienceRestrictionCondition">
```

```
   <complexContent>
      <extension base="s0:AbstractConditionType">
         <sequence>
            <element name="Audience" type="s0:URIReferenceType"
                          minOccurs="0" maxOccurs="unbounded"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>
```

Advice

The Advice element is a general container for any additional information that does not affect the semantics or validity of the assertion itself.

Element `<Advice>`

The `<Advice>` element permits evidence supporting the assertion claims to be cited, either directly (through incorporating the claims) or indirectly (by reference to the supporting assertions.

The following schema defines the `<Advice>` element:

```
<element name="Advice">
   <complexType>
      <sequence>
         <element name="Assertion" type="s0:Assertion"
               minOccurs="0" maxOccurs="unbounded"/>
         <element name="AssertionRef" type="s0:AssertionRef"
               minOccurs="0" maxOccurs="unbounded"/>
         <any namespace="##any" processContents="Skip">
     </sequence>
  </complexType>
</element>
```

SAML Protocol

SAML Assertions may be generated and exchanged using a variety of protocols. The bindings section of this document describes specific means of transporting SAML assertions using existing widely deployed protocols. SAML aware clients may in addition use the request protocol defined by the `<SAMLQuery>` and `<SAMLQueryResponse>` elements described in this section.

Element `<SAMLQuery>`

The query specifies the principal and the resources for which access is requested by use of the claim element syntax. The information requested in the response is specified by means of the `<Respond>` element described in section 0.

The `<SAMLQuery>` element is defined by the following schema:

```
<element name="SAMLQuery">
   <complexType>
      <sequence>
         <attribute name="RequestID" type="s0:AssertionID"/>
         <element name="QueryTemplate"
                     type="s0:SAMLAssertionPackageType"/>
         <element name="Respond"     type="s0:Respond"/>
     </sequence>
  </complexType>
</element>
```

Attribute `<RequestID>`

The `RequestID` attribute defines a unique identifier for the assertion request. If an assertion query specifies a `RequestID` value the same value MUST be returned in the response unless a Respond element of `Static` is specified.

Element `<Respond>`

The `<Respond>` element in the request specifies one or more strings included in the request that specify data elements to be provided in the response.

The Service SHOULD return a requested data element if it is available. The Service MAY return additional data elements that were not requested. In particular, the service MAY return data elements specified in the request with the response.

Defined identifiers include:

Identifier	Description
`Static`	Specifies that the response may return any data element thus allowing the responder to return a static pre-signed assertion.
`DecisionClaim`	Specifies that the response may return an assertion that contains a `<DecisionClaim>` element
`AttributeClaim`	Specifies that the response may return an assertion that contains a `<AttributeClaim>` element
`ExtendedAttributeClaim`	Specifies that the response may return an assertion that contains a `<ExtendedAttributeClaim>` element
`AuthorizationClaim`	Specifies that the response may return an assertion that contains a `<AuthorizationClaim>` element
`AuthenticationClaim`	Specifies that the response may return an assertion that contains a `<DecisionClaim>` element
`XML Schema URI`	If a URI is specified the response may contain Claims, Conditions and Advice elements specified by the corresponding XML schema.

The `<Respond>` element is defined by the following schema:

```
<element name="Respond" >
   <complexType>
      <sequence>
```

```
            <element name="Accept" type="string"
                  minOccurs="0" maxOccurs="unbounded"/>
         </sequence>
      </complexType>
</element>
```

Element `<SAMLQueryResponse>`

The response to a `<SAMLQuery>` is a `<SAMLQueryResponse>` element. This returns the `RequestID` specified in the response and a `<SAMLAssertionPackage>` element. The information returned in the response is controlled by the `<Respond>` element of the request.

The `<SAMLQueryResponse>` element is defined by the following schema:

```
<element name="SAMLQueryResponse">
   <complexType>
      <sequence>
                  <!-- Basic Information -->
         <attributename="RequestID"   type="s0:uriReference"/>
         <element name="SAMLAssertionPackage"
                  type="s0:SAMLAssertionPackageType"/>
     </sequence>
  </complexType>
</element>

</schema>
```

Schema Extension

The SAML schema is designed to support extensibility by means of XML abstract types. Extension schemas should specify the purpose of extension elements by defining them as extensions of the appropriate abstract types.

The following abstract types are defined in the schema:

Abstract Type	Purpose
`AbstractClaimType`	Specify a new claim element.
`AbstractSubjectType`	Specify a new element for identifying the subject of a claim.
`AbstractAttributeType`	Specify structured attribute data.
`AbstractConditionType`	Specify a new condition element.

In addition the `<Advice>` element permits arbitrary elements to be included without type restriction.

Alternate Assertion Structure Proposal

Note: this section contains draft-orchard-maler-assertion-00; the differences between this structure and the one in the previous section need to be reconciled.

Introduction

This section describes 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.

Section 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

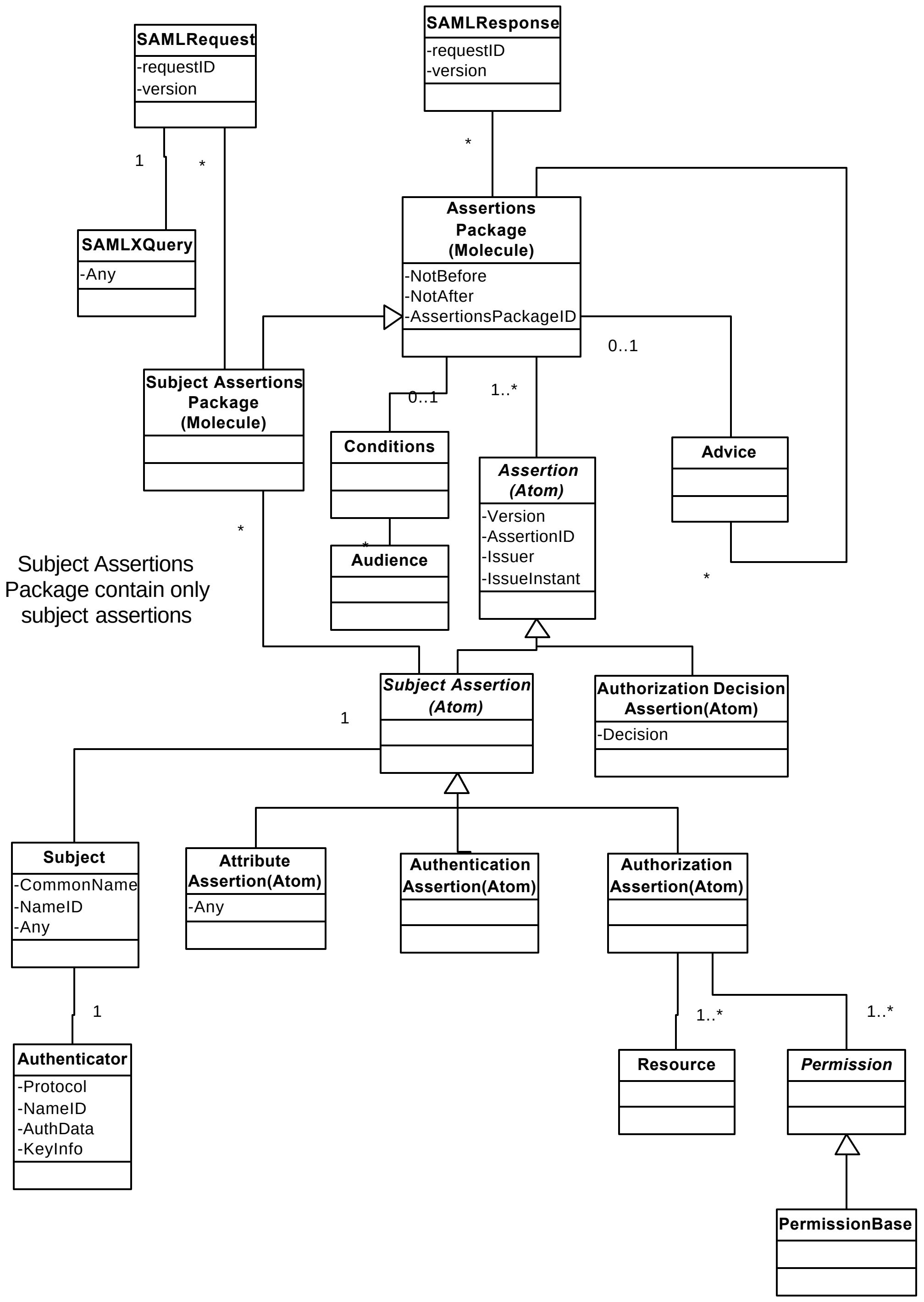
SAML-encoded information can be conveyed as a whole message in its own right ("standalone SAML"), without being embedded in another XML structure such as a purchase order. The form it takes for this purpose is either a request message or a response message. It is presumed that a SAML message is conveyed 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>
```

```xml

<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>
```

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:

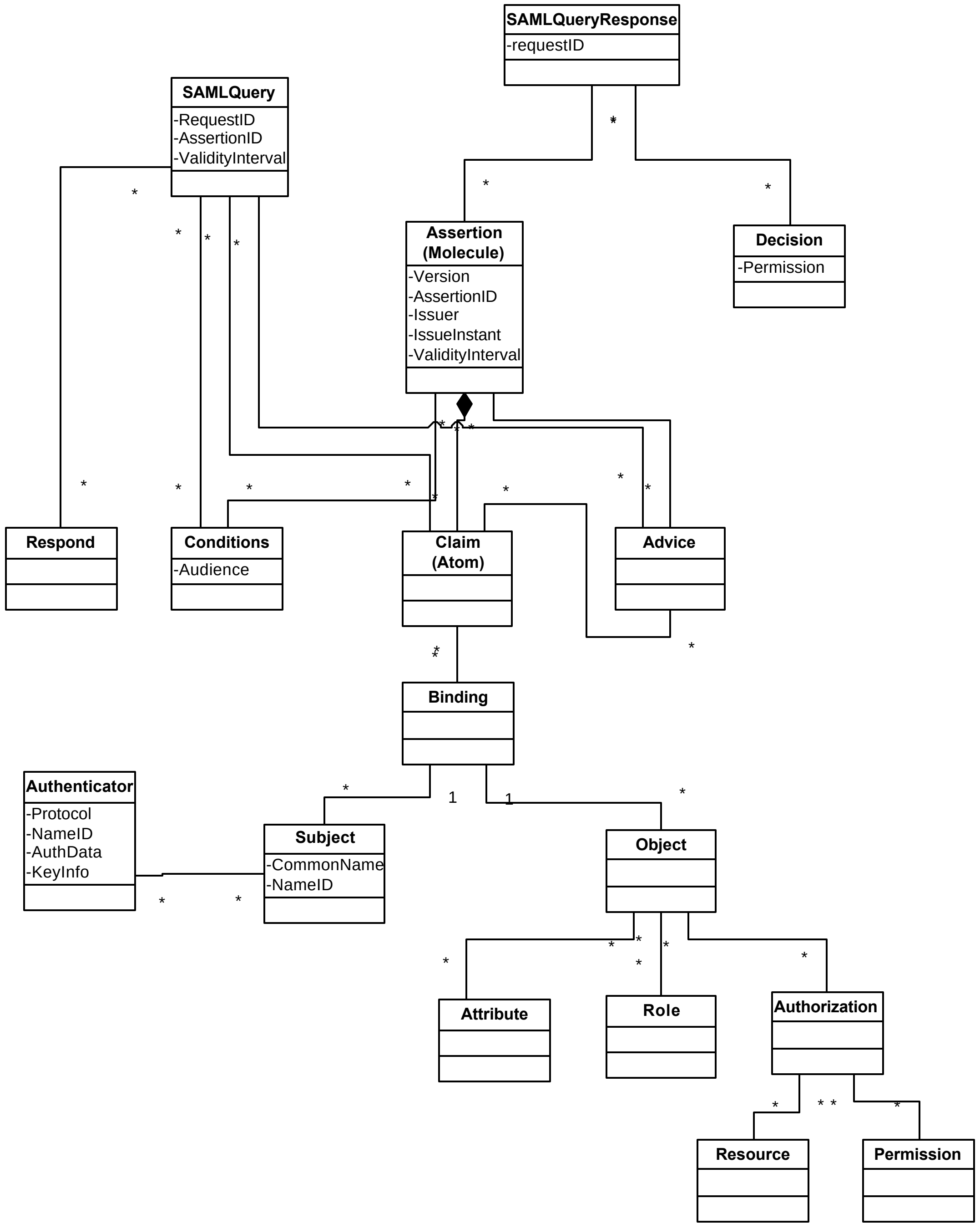
<PermissionList>
<BasePermission>R</BasePermission>
<ExtendedPermission>Provision</BasePermission>

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



SAML Request/Response Protocols

The basic data objects of the SAML protocol model are "Assertions" and "References" (to Assertions). Assertions are of two different types: "authentication" and "attribute". The resulting four data objects, in their current versions, are represented in the SAML namespace. Syntax definitions for the various types of assertion can be found elsewhere.

(Note: the decision assertion is eliminated, by allowing the PEP to request an attribute assertion (or reference thereto) that affirms the question to be decided (e.g. such-and-such a Principal occupies such-and-such a role, or such-and-such a Principal is permitted to perform such-and-such an action on such-and-such an object. If the PDP returns the requested assertion (or reference thereto), without modification, it has effectively answered "Yes" to the question).

The SAML protocol specification defines a Request/Response pair of messages by which the Requestor requests that the Responder issue an assertion of a specified type. If a suitable assertion already exists, then that assertion may be returned in response to the request, without the responder having to create a new one. Even for the case where the PEP requests that the PDP return a specified list of attributes for an identified Principal, the response is treated as an assertion whose authenticity is vouched for by the PDP.

This scope does not include the request by a Principal to a PEP for access to a resource. This aspect will be addressed directly by the "Bindings" working group.

The following entities in the protocol model may adopt the role of Requestor in the exchange: Principal, PEP, PDP and Authority. The following entities in the protocol model may adopt the role of Responder in the exchange: Authority and PDP. Table 1 shows typical applications of the messages.

Requestor	Responder	Typical application
Principal	Authority	The Authority returns an authentication or attribute assertion (or reference thereto) with the Principal as subject
Authority	PDP	The PDP returns an authentication or attribute assertion (or reference thereto) with a Principal designated by the Authority as subject
PEP	PDP	The PDP returns an attribute assertion (or reference thereto) with a Principal designated by the PEP as subject
PDP	Authority	The Authority returns an authentication or attribute assertion with a Principal designated by the PDP as subject

Table 1 - Typical applications of the request/response messages

The request is in the form of a "prototype" of the required assertion. Each attribute of the required assertion is represented in the prototype by a "type"/"value" pair. The requestor may omit the "value" field, if it does not know, or care, what value should be assigned to the corresponding element in the resulting assertion. The responder may modify the requested values. It may also omit requested elements and it may add additional elements. These actions are reflected in the "status" element of the response.

In addition to the prototype assertion, the Requestor may supply some or all of the information required by the Responder to prepare the requested assertion. The additional information may take the form of:

- Assertions of any type,
- References to assertions of any type, and
- Information about the Principal (such as its posited name and authenticator).

(Note: XML schemas are used here to define the contents of the request and response messages. However, it is not the intention that messages conformant with these schemas will actually form the messages exchanged between parties in the SAML model. The precise contents of messages will depend on the transport protocols to which they are bound, and it is the task of the "Bindings" working group to define the precise message contents for each transport protocol. The schemas defined here serve merely as guidance to the "Bindings" working group.)

There are two basic message types, the Request message and the corresponding Response message. The Request message contains the following fields.

```
<element name = "RequestIdentifier" type = "string"/>
<element name = "PrototypeAssertionsList">
        <element name = "PrototypeAssertion" minOccurs = "0" maxOccurs = "unbounded" >
                <complexType>
                        <sequence>
                                <element name = "FieldType" type = "string"/>
                                <element name = "FieldValue" type = " … " minOccurs = "0"/>
                        </sequence>
                </complexType>
        </element>
</element>
<element name = "SupportingInformation" type = "SupportingInformation"/>
</element>
```

The FieldType string is the name of the element requested to be present in the assertion returned by the responder.
The FieldValue value is the value requested for that element.

(Note: an alternative way to handle this is to include a conformant assertion whose field values are set to some special value that indicates they are to be completed.)

```
<element name = "SupportingInformation">
        <complexType>
                <sequence>
                        <element name = "Reference" type = "string" minOccurs = "0" maxOccurs = "1" />
                        <element name = "Assertion" type = "SamlAssertion" minOccurs = "0" maxOccurs =
                "unbounded"/>
                        <element name = "Principal" type = "Principal" minOccurs = "0" maxOccurs = "1"/>
                </sequence>
        </complexType>
</element>

<element name = "Principal">
        <complexType>
                <sequence>
                        <element name = "Name" type = "Name" minOccurs = "0" maxOccurs = "1" />
                        <element name = "Authenticator" type = "Authenticator" minOccurs = "0" maxOccurs =
                "unbounded"/>
                </sequence>
        </complexType>
</element>
```

The "Authenticator" element is yet to be defined. However, it must be capable of accommodating a salted password digest, a cryptographic challenge/response pair or a document/signature pair.

The Response message contains the following fields.

```
<element name = "RequestIdentifier" type = "string"/>
<element name = "AssertionsList">
	<element name = "Assertion" minOccurs = "0" maxOccurs = "unbounded">
		<complexType>
			<sequence>
				<element name = "Assertion" type = "SamlAssertion"/>
				<element name = "Status" type = "Status"/>
			</sequence>
		</complexType>
	</element>
</element>
</element>
```

Protocol Model

Editor's note*: some of the material in this section has been superseded by the material above. However some of the material below has not yet been incorporated into the Protocols text, so it has been included here for completeness*.

The model contains eight elements:

The Principal,

The Primary Domain,

The Secondary Domain,

The Authentication Authority,

The Authorization Authority,

The Session Authority,

The Policy Enforcement Point, and

The Policy Decision Point.

The **Principal** is an entity that requires controlled access to resources in a Secondary Domain.

The **Primary Domain** is an administrative domain in which the Principal can be authenticated without assistance from any other domain.

The **Secondary Domain** is an administrative domain in which the Principal cannot be authenticated except with assistance from a Primary Domain.

The Principal has at least one name in a namespace sub-tree administered by the **Authentication Authority** in the Primary Domain. The Authentication Authority binds the Principal's name to an authentication mechanism in a "name assertion".

The Principal may have one or more entitlements in an entitlement-space sub-tree administered by the **Authorization Authority** in the Primary Domain. The Authorization Authority binds the Principal's name to a name assertion in an "entitlement assertion".

The Principal may have a session state in a session state-space sub-tree administered by the **Session Authority**. The Session Authority binds the Principal's session state to a name assertion in a "session assertion".

The **Policy Enforcement Point** authenticates the Principal with the assistance of a Policy Decision Point and controls its access to resources in the Secondary Domain.

The **Policy Decision Point** authenticates the Principal and determines its eligibility to access resources in the Secondary Domain on the basis of the assertions.

Figure 1 indicates which elements of the model communicate with which other elements.

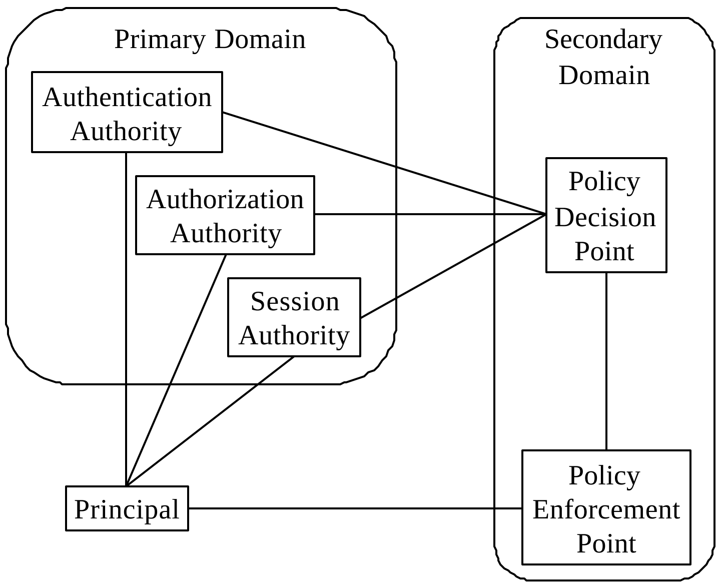


Figure 1 - Model

There are seven authentication data structures:

AuthnNotification,

AuthnAcknowlegment,

AuthnRequest,

AuthnResponse,

AuthnQuery,

AuthnResult and

Ref(AuthnNotification).

There are seven authorization data structures:

AuthzNotification,

AuthzAcknowlegment,

AuthzRequest,

AuthzResponse,

AuthzQuery,

AuthzResult and

Ref(AuthzNotification).

There are seven session data structures:

SessionNotification,

SessionAcknowlegment,

SessionRequest,

SessionResponse,

SessionQuery,

SessionResult and

Ref(SessionNotification).

For the purpose of explaining the model, only the authentication protocols will be described; the authorization and session data structures are used in an analogous fashion. In the authorization variants, the Policy Decision Point is responsible for obtaining the authorization policy definition appropriate to the specified action and the environmental variables appropriate to the policy. These two data structures are out of scope for the current version of the specification.

The Ref(AuthnNotification) data structure is defined in the Bindings section of the specification, not in this, the Protocols, section. The step in which the Principal authenticates itself to the Policy Enforcement Point is not defined in this specification. However, it is a requirement of this step that it provide a posited name for the

Principal and an authenticator. The posited name shall include a domain name, identifying the Authentication Authority in the Principal's Primary Domain, and a Principal name. The authenticator may be in any one of a number of forms, including a password, a symmetric-key challenge/response pair, an asymmetric-key challenge/response pair or a document/signature pair.

Discovery of services in a remote domain is outside the scope of this specification.

Protocol exchanges

Principal-centered direct protocol

This protocol may be used when the Principal is capable of relaying messages of unlimited length between the Primary Domain and the Secondary Domain, and when the Secondary Domain is not capable of communicating with the Primary Domain directly at the time at which the Principal communicates with the Secondary Domain.

Figure 2 shows the Principal-centered direct protocol.

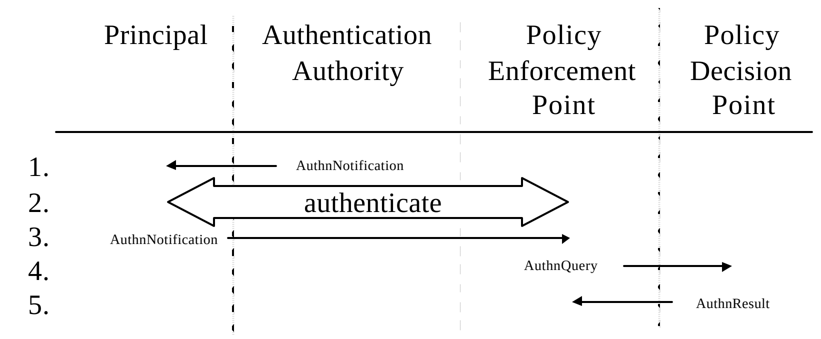


Figure 2 - Principal-centered direct protocol

It proceeds by the following steps.

1. The Principal obtains a name assertion from an Authentication Authority in the Primary Domain in an AuthnNotification message. The authentication of the Principal by the Authentication Authority is outside the scope of this specification.

2. The Principal conducts an authentication exchange with the Policy Enforcement Point. However, the Policy Enforcement Point is not capable of completing the authentication without the help of the Policy Decision Point.

3. The Principal provides the name assertion in an AuthnNotification message.

4. The Policy Enforcement Point sends the posited name, the authenticator and the name assertion to the Policy Decision Point in an AuthnQuery message.

5. The Policy Decision Point authenticates the Principal using the posited name, authenticator and name assertion provided in step 4 and returns the result to the Policy Enforcement Point in an AuthnResult message.

Principal-centered indirect protocol

This protocol may be used when the Principal is only capable of relaying messages of limited size from the Primary Domain to the Secondary Domain and the Secondary Domain is capable of communicating with the Primary Domain at the time at which the Principal communicates with the Secondary Domain.

Figure 3 shows the Principal-centered indirect protocol.

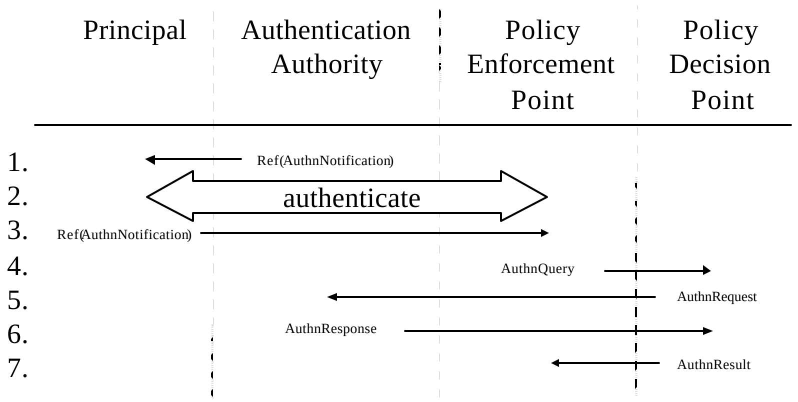


Figure 3 - Principal-centered indirect protocol

It proceeds by the following steps.

1. The Principal obtains a reference to a name assertion from an Authentication Authority in the Primary Domain in the Ref(AuthnNotification) message. As in the previous protocol, the authentication of the Principal by the Authentication Authority is out of scope.

2. The Principal conducts an authentication exchange with the Policy Enforcement Point. As before, the Policy Enforcement Point is not capable of completing the authentication without the help of the Policy Decision Point.

3. The Principal provides the reference to the name assertion in the Ref(AuthnNotification) message.

4. The Policy Enforcement Point sends the posited name, the authenticator and the reference to the name assertion to the Policy Decision Point in the AuthnQuery message.

5. The Policy Decision Point sends a request for the name assertion to the Authentication Authority in the Primary Domain in the AuthnRequest message.

6. The Authentication Authority sends the name assertion in an AuthnResponse message.

7. The Policy Decision Point authenticates the Principal and returns the result to the Policy Enforcement Point in an AuthnResult message.

Pull protocol

This protocol may be used when the Principal communicates with the Secondary Domain without being directed by the Primary Domain.

Figure 4 shows the pull protocol.

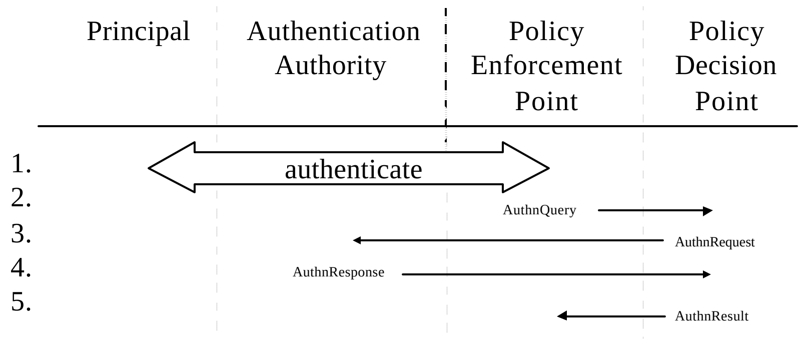


Figure 4 - Pull protocol

It proceeds by the following steps.

1. The Principal conducts an authentication exchange with the Policy Enforcement Point. As before, the Policy Enforcement Point is not capable of completing the authentication without the help of the Policy Decision Point.

2. The Policy Enforcement Point sends the posited name and the authenticator to the Policy Decision Point in the AuthnQuery message.

3. The Policy Decision Point sends a request for the name assertion to the Authentication Authority in the Primary Domain.

4. The Authentication Authority sends the name assertion in an AuthnResponse message.

5. The Policy Decision Point authenticates the Principal using the posited name and authenticator obtained from the Policy Enforcement Point in step 2 and the name assertion obtained from the Authentication Authority in step 4 and returns the result to the Policy Enforcement Point in the AuthnResult message.

Push protocol

This protocol may be used when the Principal communicates with the Secondary Domain under the direction of the Primary Domain. Because it requires the Policy Decision Point to maintain state between communication sessions with the Authentication Authority and the Principal, it is less favoured than the Principal-centered protocols.

Figure 5 shows the Push protocol.

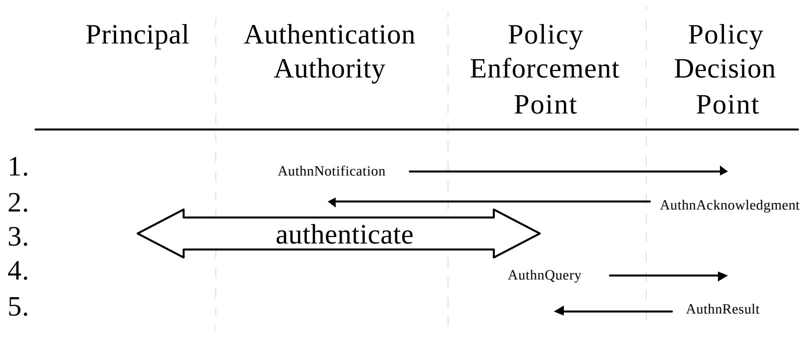


Figure 5 - Push Protocol

It proceeds by the following steps.

1. The Authentication Authority sends a name assertion in an AuthnNotification message to the Policy Decision Point in the Secondary Domain.

2. The Policy Decision Point sends an acknowledgment for the name assertion to the Authentication Authority in the Primary Domain in an AuthnAcknowledgment message.

3. The Principal conducts an authentication exchange with the Policy Enforcement Point. As before, the Policy Enforcement Point is not capable of completing the authentication without the help of the Policy Decision Point.

4. The Policy Enforcement Point sends the posited name and the authenticator to the Policy Decision Point in an AuthnQuery message.

5. The Policy Decision Point authenticates the Principal using the name assertion obtained in step 2 and the posited name and authenticator obtained in step 4 and returns the result to the Policy Enforcement Point in an AuthnResult message.

Primary domain session-close protocol

This protocol may be used to notify Secondary Domains when a Principal logs off in the Primary Domain.

Figure 6 shows the Primary Domain session-close protocol.

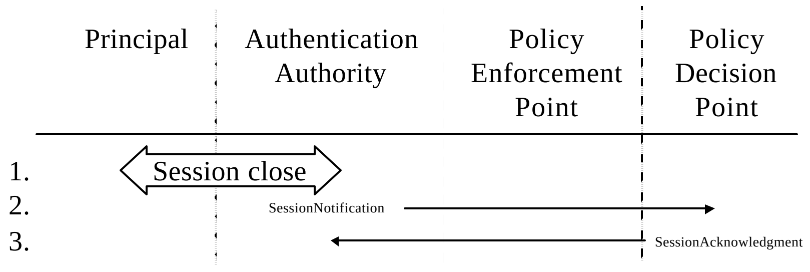


Figure 6 - Primary domain session close protocol

It proceeds by the following steps.

1. The Principal closes the existing session with the Authentication Authority.

2. The Authentication Authority sends a SessionnNotification message to the Policy Decision Point in the Secondary Domain indicating that the Principal has closed the session.

3. The Policy Decision Point sends an acknowledgment to the Authentication Authority in the Primary Domain using the SessionAcknowledgment message.

Note: the Policy Enforcement Point should confirm the session status of the Principal with the Policy Decision Point before processing each exchange between itself and the Principal. In this way, the session closure will be effective immediately.

Secondary domain session-close protocol

This protocol may be used when the Principal logs off in the Secondary Domain.

Figure 7 shows the Secondary Domain session-close protocol.

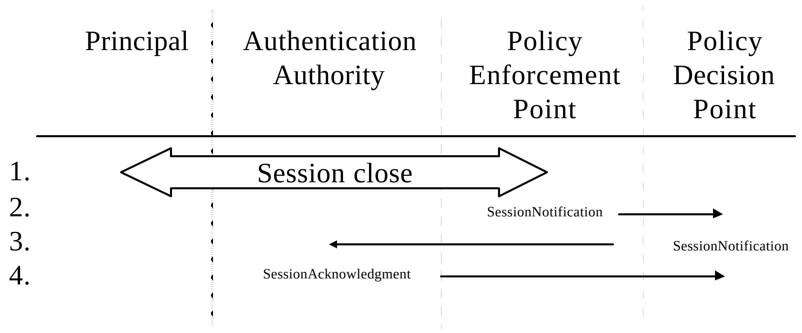


Figure 7 - Secondary domain session close protocol

It proceeds by the following steps.

1. The Principal closes the existing session with the Policy Enforcement Point.

2. The Policy Enforcement Point notifies the Policy Decision Point in a SessionNotification message.

3. The Policy Decision Point sends a SessionnNotification message to the Authentication Authority in the Primary Domain, indicating that the Principal has closed the session.

4. The Authentication Authority sends a SessionAcknowledgment message to the Policy Decision Point in the Secondary Domain.

Data structures

Note: there are separate data structures for authentication, authorization and session exchanges. If an entity needs information on any combination of name, entitlements and session status, it must conduct separate protocols for each. However, these separate protocols may proceed in parallel.

Schema for the data structures can be found in the Schema section of this specification.

AuthnNotification

The AuthnNotification message is used in the Principal-centered direct authentication protocol to send the name assertion from the Authentication Authority to the Principal and from the Principal to the Policy Enforcement Point. It is also used in the Push protocol to send the name assertion from the Authentication Authority to the Policy Decision Point. It contains the following information.

> version - this specification version number.
>
> notification-identifier - an identifier assigned by the message originator. It must be unique among all the outstanding AuthnNotification messages.
>
> name-assertion - the name assertion.
>
> sender - the name of the sender, as agreed between the sender and receiver during initialization. It must be unique among all the sender names recognized by the receiver.
>
> intended-receiver - the name of the receiver, as agreed between the sender and receiver during initialization. It must be unique among all the receiver names recognized by the sender.

Note: the name assertion contains identifiers for the Authentication Authority and the Principal. It also includes validity dates and authentication information (e.g. a public key).

AuthnAcknowlegment

The AuthnAcknowlegment message is used in the Push protocol for the Policy Decision Point to acknowledge receipt of the name assertion from the Authentication Authority. It contains the following information.

> version - this specification version number.
>
> notification-identifier - the notification identifier supplied in the corresponding AuthnNotification message.
>
> success-indicator - an indication of whether the receiver was able to process the AuthnNotification message.
>
> error-code - error code.

The following error codes shall be supported.

> Unsupported version
>
> Unsupported authentication method

AuthnRequest

The AuthnRequest message is used in the Principal-centered indirect protocol and the Pull protocol for the Policy Decision Point to request the name assertion from the Authentication Authority. It contains the following information.

version - this specification version number.

request-identifier - an identifier assigned by the message originator. It must be unique among all the outstanding AuthnRequest messages.

posited-name - the Primary Domain and Principal names claimed by the Principal. Optional.

reference to name assertion - a reference to the name assertion. Optional, if the posited name is not present, then this field must be present.

sender - the name of the sender, as agreed between the sender and receiver during initialization. It must be unique among all the sender names recognized by the receiver.

intended-receiver - the name of the receiver, as agreed between the sender and receiver during initialization. It must be unique among all the receiver names recognized by the sender.

Note: the Authentication Authority receives no evidence that the Principal has correctly authenticated to the Policy Enforcement Point.

AuthnResponse

The AuthnResponse message is used in the Principal-centered indirect protocol and the Pull protocol for the Authentication Authority to return the name assertion to the Policy Decision Point. It contains the following information.

version - this specification version number.

request-identifier - the request identifier supplied in the corresponding AuthnRequest message.

name-assertion - the name assertion.

success indicator

error code

AuthnQuery

This protocol is used in the Principal-centered direct and indirect protocols and the Pull and Push protocols for the Policy Enforcement Point to request the Policy Decision Point to perform the authentication of the Principal.

version - this specification version number.

request-identifier - an identifier assigned by the message originator. It must be unique among all the outstanding AuthnQuery messages.

posited name - the name claimed by the Principal.

authenticator - the data used in the authentication exchange between the Policy Enforcement Point and the Principal. This may be a user-name/password combination, a symmetric-key challenge/response combination, an asymmetric-key challenge response combination or a document/signature combination.

name-assertion - the name assertion. Optional.

reference to name assertion - a reference to a name assertion. Optional, at least one of "posited name", "name assertion" or "reference to name assertion" must be present.

AuthnResult

This protocol is used in the Principal-centered direct and indirect protocols and the Pull and Push protocols for the Policy Decision Point to return the result of the authentication of the Principal to the Policy Enforcement Point.

version - this specification version number.

request-identifier - the request identifier from the corresponding AuthnQuery message.

success indicator

error code

AuthzNotification

The AuthzNotification message is used in the Principal-centered direct authorization protocol to send the entitlement assertion from the Authorization Authority to the Principal and from the Principal to the Policy Enforcement Point. It is also used in the Push protocol to send the entitlement assertion from the Authorization Authority to the Policy Decision Point. It contains the following information.

version - this specification version number.

notification-identifier - an identifier assigned by the message originator. It must be unique among all the outstanding AuthzNotification messages.

entitlement-assertion - the entitlement assertion.

sender - the name of the sender, as agreed between the sender and receiver during initialization. It must be unique among all the sender names recognized by the receiver.

intended-receiver - the name of the receiver, as agreed between the sender and receiver during initialization. It must be unique among all the receiver names recognized by the sender.

Note: the entitlement assertion contains an identifier for the Authorization Authority and a reference to the associated Principal name-assertion. It also contains validity dates.

AuthzAcknowlegment

The AuthzAcknowlegment message is used in the Push protocol for the Policy Decision Point to acknowledge receipt of the entitlement assertion from the Authorization Authority. It contains the following information.

version - this specification version number.

notification-identifier - the notification identifier supplied in the corresponding AuthzNotification message.

success-indicator - an indication of whether the receiver was able to process the AuthzNotification message.

error-code - error code.

AuthzRequest

The AuthzRequest message is used in the Principal-centered indirect protocol and the Pull protocol for the Policy Decision Point to request the entitlement assertion from the Authentication Authority. It contains the following information.

version - this specification version number.

request-identifier - an identifier assigned by the message originator. It must be unique among all the outstanding AuthzRequest messages.

posited name - the posited name of the Principal. Optional.

reference to entitlement assertion - reference to an entitlement assertion. Optional. If the posited name is absent, then this field must be present.

sender - the name of the sender, as agreed between the sender and receiver during initialization. It must be unique among all the sender names recognized by the receiver.

intended-receiver - the name of the receiver, as agreed between the sender and receiver during initialization. It must be unique among all the receiver names recognized by the sender.

Note: the Authorization Authority receives no evidence that the Principal correctly authenticated to the Policy Enforcement Point. In the Pull protocol, all suitable entitlement assertions are requested.

AuthzResponse

The AuthzResponse message is used in the Principal-centered indirect protocol and the Pull protocol for the Authorization Authority to return the entitlement assertion to the Policy Decision Point. It contains the following information.

version - this specification version number.

request-identifier - the request identifier supplied in the corresponding AuthzRequest message.

entitlement assertion - the entitlement assertion.

success indicator

error code

AuthzQuery

This protocol is used in the Principal-centered direct and indirect protocols and the Pull and Push protocols for the Policy Enforcement Point to request the Policy Decision Point to confirm the authorization of the Principal.

version - this specification version number.

request-identifier - an identifier assigned by the message originator. It must be unique among all the outstanding AuthzQuery messages.

action - a compound variable comprising the name of the object method and a sensitivity value for the object that the Principal is attempting to access.

principal name - the authenticated or claimed name of the Principal. Optional. Must be identical to the posited name in any accompanying AuthnQuery message.

entitlement-assertion - the entitlement assertion. Optional.

reference to the entitlement assertion - a reference to the entitlement assertion. Optional, it should be present if the entitlement assertion is absent. Optional. At least one of "principal name", "ntitlement assertion" or "reference to entitlement assertion" must be present.

AuthzResult

This protocol is used in the Principal-centered direct and indirect protocols and the Pull and Push protocols for the Policy Decision Point to return the result of the authorization of the Principal to the Policy Enforcement Point.

version - this specification version number.

request-identifier - the request identifier supplied in the corresponding AuthzRequest message.

success indicator

error code

SessionNotification

The SessionNotification message is used in the Principal-centered direct session protocol to send the session assertion from the Session Authority to the Principal and from the Principal to the Policy Enforcement Point. It is also used in the Push protocol to send the session assertion from the Session Authority to the Policy Decision Point. It is also used in the Primary Domain session close and Secondary Domain session close protocols to indicate that the session with the Principal has been closed. It contains the following information.

version - this specification version number.

notification-identifier - an identifier assigned by the message originator. It must be unique among all the outstanding SessionNotification messages.

session-assertion - the session assertion.

sender - the name of the sender, as agreed between the sender and receiver during initialization. It must be unique among all the sender names recognized by the receiver.

intended-receiver - the name of the receiver, as agreed between the sender and receiver during initialization. It must be unique among all the receiver names recognized by the sender.

Note: the session assertion identifies the Principal either directly or by reference to a name assertion. It also contains an indication of the Principal's session state (e.g. "session closed").

SessionAcknowlegment

The SessionAcknowlegment message is used in the Push protocol for the Policy Decision Point to acknowledge receipt of the session assertion from the Session Authority. It is also used in the Primary Domain session close and Secondary Domain session close protocols to acknowledge that the session with the Principal has been closed. It contains the following information.

version - this specification version number.

notification-identifier - the notification identifier supplied in the corresponding SessionNotification message.

success-indicator - an indication of whether the receiver was able to process the SessionNotification message.

error-code - error code.

The following error codes shall be supported.

Unsupported version

SessionRequest

The SessionRequest message is used in the Principal-centered indirect protocol and the Pull protocol for the Policy Decision Point to request the session assertion from the Session Authority. It contains the following information.

version - this specification version number.

request-identifier - an identifier assigned by the message originator. It must be unique among all the outstanding SessionRequest messages.

principal name - the name of the Principal. Optional.

reference to session assertion - reference to the session assertion. Optional, is the principal name field is absent, then this field must be present.

sender - the name of the sender, as agreed between the sender and receiver during initialization. It must be unique among all the sender names recognized by the receiver.

intended-receiver - the name of the receiver, as agreed between the sender and receiver during initialization. It must be unique among all the receiver names recognized by the sender.

Note: the Session Authority receives no evidence that the Principal correctly authenticated to the Policy Enforcement Point.

SessionResponse

The SessionResponse message is used in the Principal-centered indirect protocol and the Pull protocol for the Session Authority to return the session assertion to the Policy Decision Point. It contains the following information.

version - this specification version number.

request-identifier - the notification identifier supplied in the corresponding SessionRequest message.

session-assertion - the session assertion.

success indication

error code

SessionQuery

This protocol is used in the Principal-centered direct and indirect protocols and the Pull and Push protocols for the Policy Enforcement Point to request the Policy Decision Point to confirm the session status of the Principal.

version - this specification version number.

request-identifier - an identifier assigned by the message originator. It must be unique among all the outstanding SessionQuery messages.

principal name - the authenticated or claimed name of the Principal. Optional. Must be identical to the posited name in any associated AuthnQuery message.

session assertion - a session assertion. Optional.

reference to session assertion - a reference to a session assertion. Optional, at least one of "principal name", "session assertion" or "reference to session assertion" must be present.

SessionResult

This protocol is used in the Principal-centered direct and indirect protocols and the Pull and Push protocols for the Policy Decision Point to return the result of the status evaluation of the Principal to the Policy Enforcement Point.

version - this specification version number.

request-identifier - the identifier from the corresponding SessionQuery message.

session assertion

success indicator

error code

Note: the session assertion returned in the SessionResult message may be integrity-protected by means other than XML Digital Signature. Alternatively, it may protected by the XML Digital Signature mechanism, signed by the Policy Decision Point.

Protocol Security considerations

With the exception of the session assertion in the SessionResult message, all assertions must be protected for integrity and authenticity using the XML Digital Signature mechanism. In addition, all protocol exchanges must be protected for integrity and authenticity. Mechanisms other than XML Digital Signature may be used for this latter purpose.

The exchange of Authority keys, certificates and certificate status information between domains is out of scope for this specification.

Conformance

The SAML Conformance Clause

The objectives of the SAML Conformance Clause are to:

a) Ensure a common understanding of conformance and what is required to claim conformance;
b) Promote interoperability for the exchange of authentication and authorization information
c) Promote uniformity in the development of conformance tests.

The conformance clause specifies explicitly all the requirements that have to be satisfied to claim conformance to the SAML Specification. These requirements can be applied at varying levels, so that a given implementation or application of the SAML Specification can achieve clearly-defined conformance with all or part of the entire set of requirements.

SAML conformance provides for both validation and certification. Validation may be done without certification, especially for such purposes as self-test. An implementer who has validated SAML conformance by means of self-test cannot legitimately use the term "certified for SAML conformance". However, validation may be all that is required for the particular purposes for which an implementer is using SAML.

Certification may require validation by a third-party or through self-test or by some automatic means e.g. by running thru a server in a lab, as determined by the certification authority.

The SAML conformance is expressed by three orthogonal dimensions.

- The first dimension is a partition, (a.k.a. profile) which is a subset of the overall specifications that includes all of the functionality necessary to satisfy the requirements of a particular community of users. The authorities for SAML are authentication authority, authorization authority, attribute authority, session authority, Policy decision authority and policy enforcement authority.
- The second dimension is the role of a system – consumer, producer or producer-consumer.
- The third dimension is the mapping of the assertions to a binding viz http, xmlp, soap, ebXML et al.

Conformance Nomenclature

The nomenclature for expressing SAML conformance would be two SAML conformance matrices as follows:

1. Partition-Role Table :

Partition	*Consumer*	*Producer*	*Producer/Consumer*
Authentication authority	y	Y	y
Authorization authority	y	Y	y
Attribute authority	y	Y	y
Session authority	y	Y	y
Policy decision authority	y	Y	y
Policy enforcement authority	y	Y	y

2. Partition-Bindings Table:

Partition	http	xmlp	SOA P	BEE P
Authentication authority	y	y	y	Y
Authorization authority	y	y	y	Y
Attribute authority	y	y	y	Y
Session authority	y	y	y	Y
Policy decision authority	y	y	y	Y
Policy enforcement authority	y	y	y	Y

Mandatory/Optional:

A system can choose to implement any or all of the partitions as per table 1, as a producer of SAML assertions, a consumer of SAML assertions or both. For each partition, role, binding combination (i.e., cell in the table) all functionality is mandatory. i.e. the system should support all SAML assertions related to that partition. It is optional as to which partition, role, binding combinations are supported (implemented). In short, as an example, if a system describes itself as conforming to a SAML Authorization authority, producer-consumer over http and SOAP, it has to consume and produce *all* SAML authentication assertions and be able to support the http and SOAP bindings described in the SAML specifications.

Extensions:

- Extensions shall not re-define semantics for existing functions
- Extensions shall not alter the specified behavior of interfaces defined in this standard
- Extensions may add additional behaviors
- Extensions shall not cause standard-conforming functions (i.e., functions that do not use the extensions) to execute incorrectly.

SAML assertions can be extended so long as the above conditions are met. It is requested that, if a system is extending the SAML assertions,

- The mechanism for determining application conformance and the extensions shall be clearly described in the documentation, and the extensions shall be marked as such;
- Extensions shall follow the spirit, principles and guidelines of the SAML specification, that is, the specifications must be extended in a standard manner as defined in the extension fields.
- In the case where an implementation has added additional behaviors, the implementation shall provide a mechanism whereby a conforming application shall be recognized as such, and be executed in an environment that supports the functional behavior defined in this standard

Note : Extensions are outside the scope of conformance. There are no mechanisms specified to validate and verify the extensions. This section contains the recommended guidelines for extensions.

Alternate approaches

The different transport mechanisms are covered under the bindings dimension.

Authorities

<Describe the authorities and relevant use case sections>

Roles

<Describe the roles and relevant use case sections>

Bindings

<Describe the bindings and relevant use cases sections>

SAML Conformance Program

The Conformance Program is described in detail in the separate SAML Conformance Program Specification V1.0. This document describes the tests required for validation and/or certification at a given profile and level, the procedure for running those tests, and the resources available to assist in validating or certifying implementations and applications **.**

Things To Do (Conformance)

1. There might be no bindings for an assertion, ie embedded assertions. Hoe can we specify and validate conformance?
2. Is partition right word ? subset ? profile ?
3. In each partition, should we define the core that is required and then the additional elements that a vendor can support for that partition? Now the granularity is a partition.

References

[Kerberos] *TBS*

[SAML-USE] *TBS*

[PKCS1] Kaliski, B., *PKCS #1: RSA Encryption Version 2.*0, RSA Laboratories, also IETF RFC 2437, October 1998.

[RFC-2104] Krawczyk, H., Bellare, M. and R. Canetti, *HMAC: Keyed Hashing for Message Authentication*, IETF RFC 2104, February 1997.

[SOAP] D. Box, D Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Frystyk Nielsen, S Thatte, D. Winer. *Simple Object Access Protocol (SOAP) 1.1*, W3C Note 08 May 2000, http://www.w3.org/TR/SOAP

[WSSL] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, *Web Services Description Language (WSDL) 1.0* September 25, 2000, http://msdn.microsoft.com/xml/general/wsdl.asp

[XACML] *TBS*

[XTASS] P. Hallam-Baker, *XML Trust Axiom Service Specification 1.0*, VeriSign Inc. January 2001. http://www.xmltrustcenter.org/

[XML-SIG] D. Eastlake, J. R., D. Solo, M. Bartel, J. Boyer , B. Fox , E. Simon. *XML-Signature Syntax and Processing*, World Wide Web Consortium. http://www.w3.org/TR/xmldsig-core/

[XML-SIG-XSD] XML Signature Schema available from http://www.w3.org/TR/2000/CR-xmldsig-core-20001031/xmldsig-core-schema.xsd.

[XML-Enc] *XML Encryption Specification*, In development.

[XML-Schema1] H. S. Thompson, D. Beech, M. Maloney, N. Mendelsohn. *XML Schema Part 1: Structures*, W3C Working Draft 22 September 2000, http://www.w3.org/TR/2000/WD-xmlschema-1-20000922/, latest draft at http://www.w3.org/TR/xmlschema-1/

[XML-Schema2] P. V. Biron, A. Malhotra, *XML Schema Part 2: Datatypes*; W3C Working Draft 22 September 2000, http://www.w3.org/TR/2000/WD-xmlschema-2-20000922/, latest draft at http://www.w3.org/TR/xmlschema-2/

http://www.itl.nist.gov/div897/ctg/conformProject.shtml

http://lists.oasis-open.org/archives/conformance/200104/msg00000.html

XML Protocol specification conformance issues