

1 **OASIS SSTC SAML Assertion Schema**

2 **Discussion**

3

4 draft-sstc-core-discussion-01.doc

5

6 28 July 2001

7 Authors:

8 Chris McLaren, Netegrity

9 Prateek Mishra, Netegrity

10 The Design Principles section is largely word-for-word from Dave Orchard and Eve Mahler's
11 draft (p.22-40, [draft-sstc-fff3-saml-spec-00]).

12

| | | |
|----|--|------------------|
| 12 | | |
| 13 | <i>OASIS SSTC SAML Assertion Schema</i> | <i>1</i> |
| 14 | <i>Discussion</i> | <i>1</i> |
| 15 | <i>1 Document Scope</i> | <i>3</i> |
| 16 | <i>2 Design Principles</i> | <i>3</i> |
| 17 | <i>3 Class Diagram</i> | <i>4</i> |
| 18 | <i>4 General Architecture</i> | <i>4</i> |
| 19 | <i>4.1 Discussion and Issues</i> | <i>4</i> |
| 20 | 4.1.1 Aggregating Assertions | 4 |
| 21 | 4.1.1.1 [ISSUE:CONS-01] Aggregation | 4 |
| 22 | 4.1.2 ID Types | 5 |
| 23 | 4.1.2.1 [ISSUE:CONS-02] IDType | 5 |
| 24 | 4.1.2.2 [ISSUE:CONS-03] Final Types vs Extensible types | 5 |
| 25 | <i>5 Assertion Specification</i> | <i>5</i> |
| 26 | <i>5.1 Discussion and Issues</i> | <i>5</i> |
| 27 | 5.1.1 Inheritance Structure | 5 |
| 28 | 5.1.1.1 [ISSUE:CONS-04] Extension Schema Structure | 6 |
| 29 | 5.1.2 Abstract Assertion type | 7 |
| 30 | 5.1.2.1 [ISSUE:CONS-05] Issuer | 7 |
| 31 | 5.1.2.2 [ISSUE:CONS-06] Version | 7 |
| 32 | 5.1.3 Conditions | 7 |
| 33 | 5.1.3.1 [ISSUE:CONS-07] Condition Types | 8 |
| 34 | 5.1.4 Advice | 8 |
| 35 | 5.1.5 Subject Assertion | 9 |
| 36 | 5.1.6 Subject | 9 |
| 37 | 5.1.7 NameIdentifier | 9 |
| 38 | 5.1.7.1 [ISSUE:CONS-08] NameIdentifier Strings | 10 |
| 39 | 5.1.8 Authenticator | 10 |
| 40 | 5.1.8.1 [ISSUE:CONS-09] Naming | 10 |
| 41 | 5.1.8.2 [ISSUE:CONS-10] Protocol Profile | 10 |
| 42 | 5.1.8.3 [ISSUE:CONS-11] “Bearer” Type | 10 |
| 43 | 5.1.9 AssertionSpecifier | 11 |
| 44 | 5.1.10 Authentication Assertion | 12 |
| 45 | 5.1.10.1 [ISSUE:CONS-12] AuthenticationCode Profile | 12 |
| 46 | 5.1.11 AuthLocale | 12 |
| 47 | 5.1.12 Attribute Assertion | 13 |
| 48 | 5.1.13 Attributes | 13 |
| 49 | 5.1.14 Authorization Decision Assertions | 14 |
| 50 | 5.1.14.1 [ISSUE:CONS-13] Authentication Decision Strings | 14 |
| 51 | 5.1.15 Object | 14 |
| 52 | 5.1.15.1 [ISSUE:CONS-14] <Action> Element Profile | 15 |
| 53 | 5.1.15.2 [ISSUE:CONS-15] Multiple Action Semantics | 15 |
| 54 | <i>5.2 Examples</i> | <i>15</i> |
| 55 | 5.2.1 Authentication Assertion Example | 15 |
| 56 | 5.2.2 Attribute Assertion Example | 17 |
| 57 | 5.2.3 Authorization Decision Example | 17 |
| 58 | <i>6 References</i> | <i>18</i> |

59

60 **1 Document Scope**

61 This document and a companion document [draft-sstc-protocol-discussion-01] provide
62 discussion and examples of schema elements and types given in [draft-sstc-schema-assertion-
63 12.xsd] and [draft-sstc-schema-protocol-12.xsd]. [draft-sstc-core-12] is the normative
64 specification document.

65 **2 Design Principles**

66 The proposed design adheres to the following principles for XML structure design:

- 67 1. Strong-typing of elements: Use XML Schema complex typing and inheritance to isolate
68 commonalities. This allows XML validators to function as “free error checkers” on
69 assertions and improves performance of streaming tools. Extension points can be created
70 by adding some abstract “base types” to the design.
- 71 2. Resist typing of data: The contents of leaf nodes have been set to either string or
72 uriReference. This does not reflect a rejection of the notion that some of these elements
73 need additional restrictions on their contents, but rather indicates a desire to avoid getting
74 drawn into the mire of “identifier religion”. Once the first-order questions of what the
75 structure of assertions and request/response pairs looks like are answered then the TC can
76 address what, if any, restrictions need to be placed on the contents of the leaf nodes.
- 77 3. Isolate extensions: Use XML Namespaces and XML Schema to isolate extensibility
78 features where possible, so that schema modules can be used to ensure compliance with
79 extensions and so that extensions can be uniquely referred to with XML namespace
80 names. This makes it easier to describe conformance to extensions.
- 81 4. Existing vocabularies: Existing XML vocabularies that are well supported, and that
82 directly address a SAML need should be used, where they exist, in preference to new
83 semantics. For example, if SAML needed a facility for marking up error messages, it
84 should prefer XHTML to a new SAML-specific vocabulary. This is illustrated in the used
85 of the XML-DSIG types for handling public key information.
- 86 5. Elements vs. attributes: Tend towards attributes for metadata and “single-field”
87 information, and elements for any content that has distinguishable subparts.
- 88 6. Distinguish clearly between required elements/attributes and optional elements/attributes.
89 Justify clearly rich cardinalities of the type “zero/one or more” instances of an element.

90

91

92

93

94

95

96

97 3 Class Diagram

98 This section should contain a complete class diagram for draft-sstc-assertion-12. For now we
 99 have the following overview of a few key types:

100

101

102

103

104

105

106

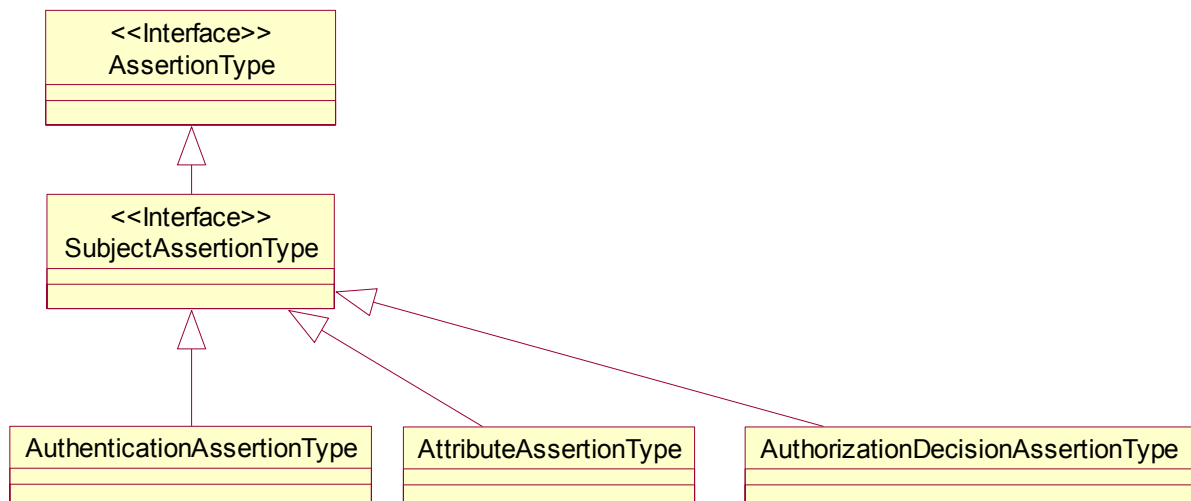
107

108

109

110

111



112 4 General Architecture

113 4.1 Discussion and Issues

114 4.1.1 Aggregating Assertions

115 Following the discussion at the third f2f no element has been provided for explicitly aggregating
 116 or collecting multiple assertions into a single object. Various SAML elements do provide
 117 context-dependent containers for assertions (e.g., `<Evidence>`) as needed in SAML messages.

118 4.1.1.1 [ISSUE:CONS-01] Aggregation

119 Do we need an explicit element for aggregating multiple assertions into a single object as part of
 120 the SAML specification? If so, what is the type of this element?

121 **4.1.2 ID Types**

122 There are a variety of places throughout the specification where objects are required to have an
123 identifier: assertions, requests, and responses all have (unique) identifiers, and the identifiers of
124 the initiating requests are also quoted back as part of responses.

125 These identifiers are all typed as instances of the “IDType”, which is in turn defined as an XML
126 Schema simple type. At present the only restriction on this type is that it must be a string.

127 Should additional constraints on the form of the identifier be deemed necessary this type’s
128 definition can be altered. Should it be deemed necessary that the form of assertion IDs needs to
129 differ from the form of, for example, request IDs then the IDType can be extended into the
130 relevant number of descendant IDTypes.

131 This issue corresponds to [ISSUE:F2F#3-8] from [f2f3-minutes] which should be consulted at
132 this point.

133 **4.1.2.1 [ISSUE:CONS-02] IDType**

134 Does the specification need additional specification for the types of assertion, request, and
135 response IDs? If so, what are these requirements?

136 **4.1.2.2 [ISSUE:CONS-03] Final Types vs Extensible types**

137 Does the TC plan to restrict certain types in the SAML schema to be final? If so, which types are
138 to be so restricted?

139 **5 Assertion Specification**

140 **5.1 Discussion and Issues**

141 **5.1.1 Inheritance Structure**

142 The specification defines three different types of assertion: authentication assertions, attribute
143 assertions, and authorization decision assertions. All of these assertion types are extensions of
144 the abstract base “subject assertion”, which is in turn an extension of the abstract base assertion
145 type.

146 This means that all three of the defined assertion types share the structure of a “subject
147 assertion”. Furthermore, since this common structure is contained within the abstract base class it
148 is available for extension, allowing new assertion types that share this structure to be defined in
149 the future.

150 The assertion base is also defined and exposed, allowing for possible future extension to create
151 assertions that do not refer to a subject.

152 **5.1.1.1 [ISSUE:CONS-04] Extension Schema Structure**

153 One of the goals of the f2f3 “whiteboard draft” was to use strong typing to differentiate between
154 the three assertion types and between the three different query forms. This has been achieved
155 through the use of “abstract” schema and schema inheritance. One implication is that any
156 concrete assertion instance MUST utilize the xsi:type attribute to specifically describe its type
157 even as all assertions will continue to use a single <Assertion> element as their container. XML
158 processors can key off this attribute during assertion processing.

159 Is this an acceptable approach? Other approaches, such as the use of substitution groups, are also
160 available. Using substitution groups, each concrete assertion type would receive its own
161 distinguished top-level element (e.g., <AuthenticationAssertion>) and there would be no need
162 for the use of xsi:type attribute in any assertion instance. At the same time the SAML schema
163 would be made somewhat more complex through the use of substitution groups.

164 Should the TC investigate these other approaches? Most important: what is the problem with the
165 current approach?

166

166 **5.1.2 Abstract Assertion type**

```

167 <element name="Assertion" type="saml:AssertionType"/>
168 <complexType name="AssertionType" abstract="true">
169   <sequence>
170     <element name="Conditions" type="saml:ConditionsType"
171       minOccurs="0"/>
172     <element name="Advice" type="saml:AdviceType" minOccurs="0"/>
173   </sequence>
174   <attribute name="Version" type="string" use="required"/>
175   <attribute name="AssertionID" type="saml:IDType" use="required"/>
176   <attribute name="Issuer" type="string" use="required"/>
177   <attribute name="IssueInstant" type="timeInstant" use="required"/>
178 </complexType>

```

179

180 The abstract assertion base type contains the common “header” information that is required in an
 181 assertion as well as optionally containing a collection of optional conditions and advice. Note
 182 that AssertionType is an **abstract** type; it can not be instantiated, it is only useful as a base for
 183 inheritance.

184 **Version:** This required attribute holds the string that uniquely identifies the version of the SAML
 185 specification within which this assertion was defined.

186 **AssertionID:** This required attribute is a string which identifies this assertion.

187 **Issuer:** This required attribute is the string the issuer provided at creation of the assertion. At
 188 present this is defined simply as a string. Additional requirements for this attribute’s form may
 189 be defined by the committee.

190 **IssueInstant:** This required attribute specifies the instant at which the assertion was issued.

191 **5.1.2.1 [ISSUE:CONS-05] Issuer**

192 Does the specification need to further specify the Issuer element? Is a string type adequate for its
 193 use in SAML? Discussion [F1] from [f2f3-minutes] points to the relevant thread on the list.

194 **5.1.2.2 [ISSUE:CONS-06] Version**

195 Does the specification need to define to further specify the version element? If so, what are these
 196 requirements? Should this be a string? Or is an unsignedint enough?

197 **5.1.3 Conditions**

```

198 <complexType name="ConditionsType">
199   <sequence>
200     <element name="Condition" type="saml:AbstractConditionType"
201       minOccurs="0" maxOccurs="unbounded"/>
202   </sequence>
203   <attribute name="NotBefore" type="timeInstant" use="optional"/>
204   <attribute name="NotOnOrAfter" type="timeInstant" use="optional"/>
205 </complexType>

```

```

206 <xsd:complexType name="AbstractConditionType" abstract="true"/>
207
208 <xsd:complexType name="AudienceRestrictionConditionType">
209   <xsd:complexContent>
210     <xsd:extension base="saml:AbstractConditionType">
211       <xsd:sequence>
212         <xsd:element name="Audience" type="xsd:anyURI"
213           minOccurs="0" maxOccurs="unbounded"/>
214       </xsd:sequence>
215     </xsd:extension>
216   </xsd:complexContent>
217 </xsd:complexType>

```

218

219 The <Conditions> element contains zero or more <Condition> elements, as well as optionally
 220 containing attributes that define the validity period over which the assertion is valid.

221 From the perspective of an RP the validity of a <Conditions> element is defined by:

222 (a) validity period as defined by the **NotBefore** and **NotOnOrAfter** attributes, AND

223 (b) the validity of the conjunction of the all of the <AbstractCondition> elements contained
 224 within it.

225 The only concrete condition type that is defined is the <AudienceRestrictionCondition>. This is
 226 a container for a sequence of <Audience> elements, each of which is a URI reference that
 227 specifies an audience to which this assertion is addressed. From the perspective of an RP which
 228 belongs to one or more audiences A_1, \dots, A_n , an assertion is addressed to the RP if at least one of
 229 the A_i occur within the <AudienceRestrictionElement>.

230 **NotBefore:** This optional attribute identifies the instant in time at which this assertion's validity
 231 begins.

232 **NotOnOrAfter:** This optional attribute identifies the instant in time at which this assertion's
 233 validity becomes false.

234 5.1.3.1 [ISSUE:CONS-07] Condition Types

235 The minutes of the F2F call for a reworking of the conditions structure to present a general
 236 conditions framework if it can be defended as “well-thought-out”. The structure presented here
 237 has a clear semantics and allows for future extensibility, via extension of the
 238 AbstractConditionType into new types of conditions. It also defines one condition type,
 239 audiences; which was the only type specifically required by the F2F minutes.

240 Does the ConditionsType meet the TC's requirements? If not, why not? Please read
 241 [ISSUE:F2F#3-17] and [ISSUE:F2F#3-18] at this point.

242 5.1.4 Advice

```

243 <xsd:complexType name="AdviceType">
244   <xsd:sequence>
245     <xsd:any namespace="##any" processContents="lax" minOccurs="0"
246       maxOccurs="unbounded"/>
247   </xsd:sequence>

```



```
248 </xsd:complexType>
```

249

250 The optional <Advice> element is an “any” container. Basically you can put any number of
251 arbitrary well-formed XML documents into this container.

252 **5.1.5 Subject Assertion**

```
253 <xsd:complexType name="SubjectAssertionType" abstract="true">
254   <xsd:complexContent>
255     <xsd:extension base="saml:AssertionType">
256       <xsd:sequence>
257         <xsd:element name="Subject" type="saml:SubjectType"
258           minOccurs="1" maxOccurs="1"/>
259       </xsd:sequence>
260     </xsd:extension>
261   </xsd:complexContent>
262 </xsd:complexType>
```

263

264 The SubjectAssertionType extends the AssertionType with the addition of a single required
265 element: the <Subject>. Note that SubjectAssertionType is an **abstract** type; it can not be
266 instantiated, it is only useful as a base for inheritance.

267 **5.1.6 Subject**

```
268 <xsd:complexType name="SubjectType">
269   <xsd:choice minOccurs="1" maxOccurs="unbounded">
270     <xsd:element ref="saml:NameIdentifier" minOccurs="0"
271       maxOccurs="unbounded"/>
272     <xsd:element ref="saml:Authenticator" minOccurs="0"
273       maxOccurs="unbounded"/>
274     <xsd:element ref="saml:AssertionSpecifier" minOccurs="0"
275       maxOccurs="unbounded"/>
276   </xsd:choice>
277 </xsd:complexType>
```

278

279 The <Subject> is a collection of one or more means of identifying the subject of an assertion.
280 The possible means are a <NameIdentifier> element, an <Authenticator> element or an
281 <AssertionSpecifier> element. Each element may occur one or more times and should be
282 understood as providing a “principal” or “description” for the subject.

283 **5.1.7 NameIdentifier**

```
284 <xsd:complexType name="NameIdentifierType">
285   <xsd:sequence>
286     <xsd:element name="SecurityDomain" type="string" minOccurs="1"
287       maxOccurs="1"/>
288     <xsd:element name="Name" type="string" minOccurs="1"
289       maxOccurs="1"/>
290   </xsd:sequence>
```

291 `</xsd:complexType>`

292

293 The NameIdentifier type represents the identification of a subject as a combination of a name
294 and a security domain.

295 **5.1.7.1 [ISSUE:CONS-08] NameIdentifier Strings**

296 Should the type of the <SecurityDomain> element of a <NameIdentifier> have additional or
297 different structure? This is also addressed in [ISSUE:F2F#3-11] of the [f2f3-minutes].

298 Should the type of the <Name> element have additional or different structure?

299 **5.1.8 Authenticator**

```
300 <complexType name="AuthenticatorType">
301   <sequence>
302     <element name="Protocol" type="uriReference"
303       maxOccurs="unbounded"/>
304     <element name="Authdata" type="string" minOccurs="0"/>
305     <element ref="ds:KeyInfo" minOccurs="0"/>
306   </sequence>
307 </complexType>
```

308

309 This element specifies one or more <Protocol> elements together an (optional) XML-DSIG
310 <KeyInfo> and/or an (optional) <AuthData> element. The intention here is that the <Protocol>
311 element would describe one or more acceptable authentication techniques such as
312 “urn:protocol:UNIX_PASSWORD_HASH”, “urn:protocol:SSL”, “urn:protocol:XML-DSIG”,
313 etc. The <KeyInfo> element would hold information about the public key (or certificate)—using
314 the structure specified by the XML-DSIG standard—and the <AuthData> element would hold
315 data such as the hash of a password.

316 **5.1.8.1 [ISSUE:CONS-09] Naming**

317 This element needs a better name than “Authenticator”. This is an open issue, being discussed on
318 the list.

319 **5.1.8.2 [ISSUE:CONS-10] Protocol Profile**

320 The TC will develop a namespace identifier (e.g., protocol above) and set of standard namespace
321 specific strings for the <Protocol> element above. If not, what approach should be taken here?

322 **5.1.8.3 [ISSUE:CONS-11] “Bearer” Type**

323 The following proposal has been made for identifying a “bearer” assertion: a distinguished URI
324 urn:protocol:bearer be used as the value of the <Protocol> element in <Authenticator> with no
325 other sub-elements. Is this an acceptable design?

326 **5.1.9 AssertionSpecifier**

```
327 <element name="AssertionSpecifier" type="saml:AssertionSpecifierType"/>
328 <xsd:complexType name="AssertionSpecifierType">
329   <xsd:choice>
330     <xsd:element name="AssertionID" type="saml:IDType" minOccurs="1"
331       maxOccurs="1"/>
332     <xsd:element name="Assertion" type="saml:AssertionType"
333       minOccurs="1" maxOccurs="1"/>
334   </xsd:choice>
335 </xsd:complexType>
```

336

337 This type is used when you want to identify the subject of an assertion by saying “The subject of
338 this assertion is whoever the subject of **the included** assertion is.” You specify the other
339 assertion either by its AssertionID, or by including the other assertion completely. Note that a
340 global element of this type has been declared, so this element can be referenced in other
341 definitions.

342

342 **5.1.10 Authentication Assertion**

```

343 <complexType name="AuthenticationAssertionType">
344   <complexContent>
345     <extension base="saml:SubjectAssertionType">
346       <sequence>
347         <element ref="saml:AuthenticationCode"/>
348         <element name="AuthenticationInstant"
349           type="timeInstant"/>
350         <element name="AuthLocale" type="saml:AuthLocaleType"
351           minOccurs="0"/>
352       </sequence>
353     </extension>
354   </complexContent>
355 </complexType>

```

356

357 The AuthenticationAssertionType extends the SubjectAssertionType with the addition of two
 358 required elements, and an optional one. Note that AuthenticationAssertionType is a **concrete**
 359 type and can be instantiated.

360 The extensions that make up this type are a string that identifies the type of authentication that
 361 was used to create the assertion (“AuthenticationCode”), an identifier of the time at which the
 362 authentication took place (“AuthenticationInstant”), and an optional advisory element that
 363 identifies the DNS domain name and IP address for system entity the authentication
 364 (“AuthLocale”).

365 **AuthenticationCode:** This is a string that identifies the type of Authentication used to generate
 366 the assertion.

367 **AuthenticationInstant:** This is the time at which the authentication took place.

368 **5.1.10.1 [ISSUE:CONS-12] AuthenticationCode Profile**

369 What restrictions, if any, should be placed on the format of the contents of the
 370 AuthenticationCode element? Should this be a closed list of possible values? Should the list be
 371 open, but with some “well-known” values? Should we refer to another list already in existence?

372 Are the set of values supported for the <Protocol> element ([ISSUE:CONS-08]) essentially the
 373 same as those required for the <AuthenticationCode> element?

374 **5.1.11 AuthLocale**

```

375 <xsd:complexType name="AuthLocaleType">
376   <xsd:sequence>
377     <xsd:element name="IP" type="string" minOccurs="0"
378       maxOccurs="1"/>
379     <xsd:element name="DNS_Domain" type="string" minOccurs="0"
380       maxOccurs="1"/>
381   </xsd:sequence>
382 </xsd:complexType>

```

383

384 This optional element contains two optional elements: an identifier of the IP address and DNS
 385 domain name of the authenticated system entity. This element is entirely advisory, since both
 386 these fields are quite easily “spoofed” but current practice appears to require its inclusion.

387 **5.1.12** *Attribute Assertion*

```
388 <complexType name="AttributeAssertionType">
389   <complexContent>
390     <extension base="saml:SubjectAssertionType">
391       <sequence>
392         <element ref="saml:Attribute" maxOccurs="unbounded"/>
393       </sequence>
394     </extension>
395   </complexContent>
396 </complexType>
```

397

398 The AttributeAssertionType extends the SubjectAssertionType with the addition of one or more
 399 attributes. Note that AttributeAssertionType is a **concrete** type and can be instantiated.

400 **5.1.13** *Attributes*

```
401 <complexType name="AttributeValueType">
402   <sequence>
403     <any namespace="##any" processContents="lax" minOccurs="0"
404       maxOccurs="unbounded"/>
405   </sequence>
406 </complexType>
407
408 <element name="Attribute" type="saml:AttributeType"/>
409
410 <complexType name="AttributeType">
411   <sequence>
412     <element name="AttributeName" type="string"/>
413     <element name="AttributeNamespace" type="uriReference"
414       minOccurs="0"/>
415     <element name="AttributeValue" type="saml:AttributeValueType"
416       minOccurs="0" maxOccurs="unbounded"/>
417   </sequence>
418 </complexType>
```

419

420 The attributes are combinations of an attribute name, and optionally a namespace and one or
 421 more values. The <AttributeNamespace> elements qualifies the <AttributeName>. The values
 422 are “any” aggregates so that an arbitrary number of well-formed XML documents (one or more)
 423 can make up a value.

424

424 **5.1.14 Authorization Decision Assertions**

```

425 <complexType name="AuthorizationDecisionAssertionType">
426   <complexContent>
427     <extension base="saml:SubjectAssertionType">
428       <sequence>
429         <element ref="saml:Object"/>
430         <element name="Answer" type="saml:DecisionType"/>
431         <element ref="saml:Evidence" minOccurs="0"
432           maxOccurs="unbounded"/>
433       </sequence>
434     </extension>
435   </complexContent>
436 </complexType>

```

437

438 The AuthorizationDecisionAssertionType extends the SubjectAssertionType with the addition of
 439 two required elements, and an optional one. Note that AuthorizationDecisionAssertionType is a
 440 **concrete** type and can be instantiated.

441 The required elements are the <Object> of the authorization decision, and the <Answer> (which
 442 represents the decision part of the authorization decision). The optional element, <**Evidence**>, is
 443 a container of zero or more AssertionSpecifiers (either AssertionIDs, or complete Assertions—
 444 see §4.1.3.1.3) that describe assertions provided as evidence for the decision. These evidence
 445 assertions can also be interpreted as “This decision is made subject to the assertions in the
 446 Evidence element”.

447 One of the required elements is the <**Answer**>, which is a string of the DecisionType. This type
 448 is an enumeration of valid answers to Authorization questions. At this time the set of possible
 449 answers is limited to “Permit”, “Deny”, and “Indeterminate” as defined below.

```

450 <xsd:simpleType name="DecisionType">
451   <xsd:restriction base="string">
452     <xsd:enumeration value="Permit"/>
453     <xsd:enumeration value="Deny"/>
454     <xsd:enumeration value="Indeterminate"/>
455   </xsd:restriction>
456 </xsd:simpleType>

```

457 **5.1.14.1 [ISSUE:CONS-13] Authentication Decision Strings**

458 Does {Permit, Deny, Indeterminate} cover the range of decision answers we need? See also
 459 discussion in [ISSUE:F2#3-33].

460 **5.1.15 Object**

```

461 <element name="Object" type="saml:ObjectType"/>
462 <complexType name="ObjectType">
463   <sequence>
464     <element name="Resource" type="xsd:uriReference"/>
465     <element name="Namespace" type="uriReference" minOccurs="0"/>
466     <element name="Action" type="string" maxOccurs="unbounded"/>
467   </sequence>

```

468 `</complexType>`

469
 470 The `<Object>` element is composed of a `uriReference` that identifies the resource (`<Resource>`),
 471 an optional namespace reference (`<Namespace>`), and a list of one or more actions that are
 472 relevant to the resource (`<Action>`). The `<Namespace>` element qualifies the `<Action>` element.

473

474 **Example:**

475 Namespace: `xmlns:http-action-namespace`

476 Actions: GET, POST, HEAD

477 **5.1.15.1 [ISSUE:CONS-14] `<Action>` Element Profile**

478 As part of f2f#3, there was a consensus that some kind of registry of actions and namespaces.
 479 This issue is also discussed in [ISSUE:F2F#3-32]. Where should this registry be maintained?
 480 There is a further question of whether the SAML specification should call components of this
 481 registry, either as part of this specification, or parallel to it (e.g., actions for HTTP, SMTP, J2EE
 482 etc.).

483 **5.1.15.2 [ISSUE:CONS-15] Multiple Action Semantics**

484 The f2f#3 left it somewhat unclear if multiple actions are supported within an `<Object>`. There is
 485 clear advantage to this type of extension (as defined in the schema above) as it provides a simple
 486 way to aggregate actions. Given that actions are strings (as opposed to pieces of XML) this does
 487 seem to provide additional flexibility within the SAML framework.

488 Does the TC support this type of flexibility?

489 **5.2 Examples**

490 **5.2.1 Authentication Assertion Example**

491 This example shows an assertion with a 5 minute lifespan that asserts that the subject (identified
 492 by both a `NameIdentifier` and a `KeyInfo` block) is in fact "SomeUser" of Example Company.

```

493 <Assertion xsi:type="saml:AuthenticationAssertionType"
494   version="http://www.oasis.org/tbs/1066-12-25/1.0"
495   AssertionID="{186CB370-5C81-4716-8F65-F0B4FC4B4A0B}"
496   Issuer="www.example.com"
497   IssueInstant="2001-05-31T13:20:00-05:00">
498     <Conditions
499       NotBefore="2001-05-31T13:20:00-05:00"
500       NotOnOrAfter="2001-05-31T13:25:00-05:00"/>
501     <Subject>
502       <NameIdentifier>
503         <SecurityDomain>www.example.com</SecurityDomain>
504         <Name>SomeUser</Name>
505       </NameIdentifier>

```

```
506     <Authenticator>
507         <ds:KeyInfo>
508             <KeyValue>
509                 <DSAKeyValue>
510                     <P>
511 /X9TgR11EilS30qcLuzk5/YRt1I870QAwX4/gLZRJmlFXUAIuFtZPY1Y+r/F9bow9s
512 ubVWzXgTuAHTRv8mZgt2uZUKWkn5/oBHSQIsJPu6nX/rfGG/g7V+fGqKYVDwT7g/bT
513 xR7DAjVUE1oWkTL2dfOuK2HXKu/yIgmZndFIACC=
514                     </P>
515                     <Q>l2BQjxUjC8yykrmCouuEC/BYHPU=</Q>
516                     <G>
517 9+GghdabPd7LvKtcNrhXuXmUr7v6OugC+VdMCz0HgmdRWVeOutRZT+ZxBxCBGLRJFn
518 Ej6EwoFhO3zwkyjMim4TwWeotUfI0o4KOuHiuzpnWRBqN/C/ohNLx+2J6ASQ7zKTX
519 vqhRkImog9/hWuWfBpKLZl6Ae1ULZAFMO/7PSSo=
520                     </G>
521                     <Y>
522 i5/D5JhXm/ZbA+ivdGtdqrrAu/HHkiMDit6J1/KFJLKkTidMzM5xJADzxw6Tj+mKji
523 +fJee5EHlQF90a7apwYTXpE6JZN8BMhOu8zw6FEhRg4xQBUerV0fRPkeN5PpyioN6
524 RvbHftp/ITULqN9N53lVTWdc9CHYat6PuOtfTWA=
525                     </Y>
526                 </DSAKeyValue>
527             </KeyValue>
528             <X509Data>
529                 <X509SubjectName>
530                     CN=SomeUser, OU=Some Group,
531                     O=Example, L=SomeCity, ST=SomeState,
532                     C=SomeCountry
533                 </X509SubjectName>
534                 <X509Certificate>
535 MIIIDMTCCAu8CBDqIR9gwCwYHkoZiZjgEAwUAMH4xCzAJBgNVBAYTA1VTMRYwFAYDVQOIeW1NYXNz
536 YWNodXNldHRzMRawDgYDVQQHEwNZZXRORWVudmRlYDQKEwloZXRLZ3JpdHkxGTAXBgNVBASt
537 EEIyQiBBZ2VudHMGR3JvdXAxZjAUBGNvbAMTDVJvYmVydCBUYX1sb3IwHhcNMDEwMjEzMDU2
538 WhcNMDEwNTEzMDU2WjB+MQswCQYDVQGEwJVUzEWMBQGA1UECBMTWFzc2FjaHVzZXR0czEQ
539 MA4GA1UEBxMHTWV0aHVlbnRzESMBAGA1UEChMJTmV0ZWdyaXR5MRkwFwYDVQQLExBCMkIgcWdlbnRz
540 IEddyb3VwMRYwFAYDVQQDEw1Sb2JlcnQgVGF5bG9yMIIIBuDCASwGByqGSM44BAEwggEfAoGBAP1/
541 U4EddRIpUt9Knc7s5Of2EbdSPO9EAMMeP4C2USZpRV1AI1h7WT2NWPq/xfW6MPbLm1Vs14E7gB00
542 b/JmYLdrmvclpJ+f6AR7ECLCT7up1/63xhv4O1fnxqimFQ8E+4P208UewwI1VBNAFpEy9nXzrith
543 1yrv8iIDGZ3RSAHHAhUAL2BQjxUjC8yykrmCouuEC/BYHPUcGYEA9+GghdabPd7LvKtcNrhXuXmU
544 r7v6OugC+VdMCz0HgmdRWVeOutRZT+ZxBxCBGLRJFnEj6EwoFhO3zwkyjMim4TwWeotUfI0o4KOu
545 HiuzpnWRBqN/C/ohNLx+2J6ASQ7zKTXvqhRkImog9/hWuWfBpKLZl6Ae1ULZAFMO/7PSSoDgYUA
546 AoGBAIufw+SYV5v2WwPor3Rk3aq6wLvx5IjA4reidfYhSSypE4nTMzOcSQA88cOk4/pio4vnyXn
547 uRB5UBfdGu2qcGE8aROiWTfATITrvM8OsBRIUYOMUAVHq1dH0T5HjeT6coqDekb2x37afyE1Jajf
548 Ted5VU1nXPqh2Grej7jrX01gMAsGByqGSM44BAMFAAMvADAsAhRy+2AJp8ZZ8OVSe02TsjZ21p0W
549 BQIUOvsjuK715yd715WvjEmP+MVzSjg=
550                 </X509Certificate>
551             </X509Data>
552         </ds:KeyInfo>
553     </Authenticator>
554 </Subject>
555 <AuthenticationType>X509v3</AuthenticationType>
556 <AuthenticationInstant>2001-05-31T13:20:00-05:00
557 </AuthenticationInstant>
558 </Assertion>
```


5.2.2 Attribute Assertion Example

This example illustrates the use of an attribute assertion to assign some attributes to a user. This example has a fictitious consortium assigning a credit summary to a given subject. Note that the value of the attribute is a block of arbitrary XML, presumably following the schema specified by the attribute namespace.

```

564 <Assertion xsi:type="saml:AttributeAssertionType"
565   version="0100"
566   AssertionID="{EE52CAF4-3452-4ebe-84D3-4D372C892A5D}"
567   Issuer="www.example.com"
568   IssueInstant="2001-05-31T13:20:00-05:00">
569   <Conditions
570     NotBefore="2001-05-31T13:20:00-05:00"
571     NotOnOrAfter="2001-05-31T13:25:00-05:00">
572   </Conditions>
573   <Subject>
574     <NameIdentifier>
575       <SecurityDomain>www.example.com</SecurityDomain>
576       <Name> cn=SomeUser,ou=finance,co=example </Name>
577     </NameIdentifier>
578   </Subject>
579   <Attribute>
580     <AttributeName>NetWorthSummary</AttributeName>
581     <AttributeNamespace>
582       http://ns.finance-vocab.org/finance
583     </AttributeNamespace>
584     <AttributeValue>
585       <CreditSummary>
586         <HistoryScore>Excellent</HistoryScore>
587         <CurrentAssets>Loaded</CurrentAsserts>
588       </CreditSummary>
589     </AttributeValue>
590   </Attribute>
591 </Assertion>

```

5.2.3 Authorization Decision Example

This example shows the result of a credit check, for a given subject. Note that the above attribute assertion is given as evidence.

```

595 <Assertion xsi:type="saml:AuthorizationDecisionAssertionType"
596   version="0100"
597   AssertionID="{5CFCA396-C2AC-497c-975F-233CDC69CFE4}"
598   Issuer="www.example.com"
599   IssueInstant="2001-05-31T13:20:00-05:00">
600   <Conditions
601     NotBefore="2001-05-31T13:20:00-05:00"
602     NotOnOrAfter="2001-05-31T13:25:00-05:00">
603     <Condition xsi:type="saml:AudienceRestrictionConditionType">
604       <Audience>
605         http://www.example.com/agreements/credit.html
606       </Audience>
607     </Condition>
608   </Conditions>
609   <Subject>

```

```
610         <NameIdentifier>
611             <SecurityDomain>us-staff</SecurityDomain>
612             <Name>cn=SomeUser,ou=finance,co=example</Name>
613         </NameIdentifier>
614     </Subject>
615     <Object>
616         <Resource>
617             http://www.example.com/confidential/agree.html
618         </Resource>
619         <Action>GET</Action>
620         <Action>POST</Action>
621         <Namespace>urn:samlaction:HTTP</Namespace>
622     </Object>
623     <Answer>Permit</Answer>
624     <Evidence>
625         <AssertionID>{EE52CAF4-3452-4ebe-84D3-4D372C892A5D}</AssertionID>
626     </Evidence>
627 </Assertion>
628
629
630
```

631

6 References

- 632
- 633 [draft-sstc-ftp3-saml-spec-00] [http://lists.oasis-open.org/archives/security-](http://lists.oasis-open.org/archives/security-services/200106/pdf00002.pdf)
634 [services/200106/pdf00002.pdf](http://lists.oasis-open.org/archives/security-services/200106/pdf00002.pdf)
- 635 [draft-sstc-protocol-discussion-01] [http://www.oasis-open.org/committees/security/docs/draft-](http://www.oasis-open.org/committees/security/docs/draft-sstc-protocol-discussion-01.pdf)
636 [sstc-protocol-discussion-01.pdf](http://www.oasis-open.org/committees/security/docs/draft-sstc-protocol-discussion-01.pdf)
- 637 [draft-sstc-schema-assertion-12.xsd] [http://www.oasis-open.org/committees/security/docs/draft-](http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-assertion-12.xsd)
638 [sstc-schema-assertion-12.xsd](http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-assertion-12.xsd)
- 639 [draft-sstc-schema-protocol-12.xsd] [http://www.oasis-open.org/committees/security/docs/draft-](http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-protocol-12.xsd)
640 [sstc-schema-protocol-12.xsd](http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-protocol-12.xsd)
- 641 [draft-sstc-core-12] <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-12.pdf>