Proposal for Extensibility Features in the SAML Schemas

Proposal Draft 02~~1~~, 1~~19~~ ~~January~~February 2004

Document identifier:

sstc-maler-schema-extension-02~~1~~

Location:

http://www.oasis-open.org/committees/documents.php?wg_abbrev=security

Authors:

Scott Cantor, individual (cantor.2@osu.edu)
Eve Maler, Sun Microsystems (eve.maler@sun.com)

Abstract:

This document proposes changes to the various extensibility mechanisms used in SAML's XML schemas. It is hoped that some of the proposal's content will contribute to a technical paper on the SAML TC's extensibility design decisions and/or to the SAML core specification's explanation of extensibility.

Status:

This is the ~~first~~second draft of this proposal, incorporating some feedback and decisions from the 20 January 2004 Security Services TC telecon and additional ideas from the XML community. Comments should be sent to the authors or to the security-services@lists.oasis-open.org mailing list.

Table of Contents

1 Introduction

The SAML TC has long had a goal of letting customizers extend SAML in a controlled way in order to maximize interoperability. SAML provides extensibility in three areas:

- **Extensibility of structure:** This includes ways to modify (add to or subtract from) SAML's native XML content models.
- **Extensibility of content:** This includes ways to customize the format and interpretation of the content of SAML's XML elements and attributes. (SAML artifacts could also be considered to come under this category.)
- **Extensibility of "protocol":** This includes ways to define new flows, called profiles, of SAML assertion creation, usage, and exchange. Sometimes these profiles also involve extended XML structures and content, as described above.

SAML was initially developed at a time when the art of W3C XML Schema (XSD) [Schema1] "schemography" was extremely new. Since SAML V1.0 first became stable, a number of users have gained experience with customizing SAML. This experience, along with lessons learned about the abilities and limitations of XSD and its conforming processors, are contributing to a new understanding of how SAML can best achieve its extensibility goal.

This proposal attempts to harness this experience to recommend improvements to SAML's extensibility in the areas of structure and content. (Note that schema features that do not affect extensibility are not discussed here, though this could easily become the subject of another position paper.)

1.1 SAML Customizations ~~to Date~~

The Liberty Alliance schemas [LibProtSchema] are the best-known and most comprehensive customization of SAML. These schemas were developed somewhat in parallel with the original SAML standard and were extended from stable but incomplete drafts of the original specification. This created some divergence as well as some duplication of work as SAML evolved and addressed limitations that Liberty also addressed in the form of extensions; these conflicts have been addressed in part through subsequent revisions of the Liberty schemas, demonstrating that extensions tend to be "evolving" specifications that must adapt as the underlying foundation changes. Understanding these points of contact and minimizing their impact is the essential goal of extensibility.

~~[@@Briefly iscuss the nature of Liberty customizations, and add mention of other known customizations]~~

1.2 SAML's Extensibility Requirements

SAML's general requirements for extensibility can be stated as follows.

> **Note:** The SAML Conformance Program Specification [SAMLConform] §2.4 outlines how extensions and their processors must behave in order to conform with SAML. For example, extensions are never allowed to redefine existing semantics nor alter specified behavior.

1. **Allow customizers to modify SAML's XML structures in sufficiently powerful, maintainable, and validatable ways**

   SAML is generic technology that is designed to be reused, customized, and profiled for a variety of purposes. In general, any customization is likely to be promoted by its creators as a second-order standard, which suggests that customizers' need to create formal schemas is relatively high. This will be particularly true wherever a customizer wishes to make third-order customizations possible.

   More controversially, in the past we have had an implicit goal of forcing customizers to define schema customization layers on top of SAML so that we could ensure their correct usage of the native SAML portions. With experience, it seems that this requirement was too nosy and, at times, too impractical.

2. **To the extent possible, gracefully tolerate extensions**

   This is something of an inverse of requirement #1: SAML processors should be able to parse (and validate) a document that has an extension in it even if the schema for the extension is not available. This probably can't be achieved in all situations, but we can use it to color our choices of mechanisms. Note that this requirement might have consequences for the Conformance document. David Orchard's recent article on versioning [Orchard] discusses some ways to achieve this goal more sucessfully.

3. **Make the extensibility mechanisms practical**

   They should not be so cumbersome or costly that customizers avoid them and use cut-and-paste instead. Support and interoperability of XSD processors and features are key variables here.

4. **Allow for unambiguous reuse of foreign taxonomies where SAML is just a carrier for them**

In general, SAML tries to avoid doing much more than codifying current usage unless current technologies are wholly inadequate. Thus, if a system of ~~information~~codes, keywords, or values ~~(such as for resource actions)~~ is already available, SAML needs to allow for a way to indicate interpretation of this information~~ provided this way~~. An example is SAML's action namespace framework, which allows usage of existing sets of actions, such as HTTP verbs and UNIX file permissions. Another example is SAML's attribute namespace framework, which has been less successful because it apparently allows for ambiguous semantics.

2 Extensibility Mechanism Choices

The following sections describe the major contenders for extensibility mechanisms, SAML V1.1's usage of each, and brief analyses of their strong and weak points. This is not an exhaustive list of the mechanisms made available by XSD. The analysis focuses on these characteristics:

- **Tools support:** This addresses #1, the practicality goal.
- **Interactions with other mechanisms:** This also addresses #1, the practicality goal.
- **Validation power:** This addresses the goal of allowing customizations to be validated in #2. Some checking can be done at “compile time” (with the schema alone as input), and some is done at “run time” (with the schema and the XML instance as input).
- **Flexibility:** This addresses the goal of allowing sufficiently powerful customizations in #2.
- **Reuse:** This addresses the goal of allowing sufficiently powerful customizations in #2, and also #3, which deals with SAML's ability to reuse in the other direction.

2.1 Derivable Types

These are types, abstract and otherwise, that are designed for derivation. XSD allows a type to serve as a base type (a parent) of a specialized type, à la object-oriented systems, unless the original type is set to “final.” This setting can be controlled globally for a whole schema.

For the complex types that get bound to XML element content models, you can do type extension, which allows adding to the end of a content model, and type restriction, which allows subtracting any optional element from a content model. For the simple types that get bound to simple element and attribute string content, only restriction through “facets,” extension by allowing a list of multiple same-typed values, and extension by unioning two types are allowed.

XSD allows the redefinition (without renaming) of imported types in an importing schema.

All of SAML's defined types are non-final and are explicitly documented as being derivable. In several cases, SAML defines “deep” complex type hierarchies (and matching elements) especially for derivation purposes. In the assertion schema, this includes:

- **ConditionAbstractType** > **AudienceRestrictionConditionType**
- **StatementAbstractType** > **SubjectStatementAbstractType** > (**AuthenticationStatementType**, **AttributeStatementType**, **AuthorizationDecisionType**)
- **AttributeDesignatorType** > **AttributeType**

In the protocol schema, this includes:

- **RequestAbstractType** > **RequestType**
- **QueryAbstractType** > **SubjectQueryAbstractType** > (**AuthenticationQueryType**, **AttributeQueryType**, **AuthorizationDecisionQueryType**)
- **ResponseAbstractType** > **ResponseType**

The types with “Abstract” in their name are, in fact, set to be abstract, which means they must be derived from in order to be used at all. The SAML core specification [SAMLCore] §6 mentions that derivations of SAML's types may be used either with the XSD `xsi:type` attribute on native elements or through substitution of a foreign element bound to the new type.

Tools support

Extension tends to be supported. No problems have been reported with SAML customizations of this sort so far. Support for restriction of complex types may be a concern, though trivial restriction from **anyType** or **anySimpleType** is likely not to pose a problem; XSD processors tend to be sloppy about checking compatibility of the restricted type at compile time. At least one expert is suspicious of the overall interoperability of derivation from complex types [Kawaguchi]. Element substitution of restricted types appears to be a problem for some XSD processors.

We don't know of any cases of restriction from the SAML schemas.

Interactions with other mechanisms

Derivation gains significant new abilities in combination with substitution and redefinition, but need not be used with them (and in this way avoids their negatives).

Use of the `<xs:any>` wildcard at the end of a sequence causes problems when extending the original sequence because of ambiguiities between optional elements and wildcard matching.

Validation power

Derivations of SAML types are validatable in XSD processors, though it should be noted that at least one expert is concerned about run-time misinterpretations [Obasanjo1]. Any other mechanism that allows for or requires derivation will gain its validation power. When abstract types are used (as in some cases in SAML), derivation (and validation of it) is forced.

Flexibility

Based on experience to date, the ability to extend types by adding to the end of the content model is sufficient for most customizations, except when the original model terminates in the `<xs:any>` wildcard.

Reuse

Customizers' reuse of native constructs: Non-final types are all available to be used in the construction of an entirely new schema that does not use the native elements. It is not possible to forbid this.

Native reuse of others' constructs: Our schemas of interest (SAML) could reuse the types defined by others' schemas. (It currently does not; the only “foreign” types it are ones built into the XSD specification.)

2.2 Model Groups and Attribute Groups

Model groups are like programming macros that allow for usage of a previously defined XML content model particle or list of attributes. They can be used to build new content models without the usual extension/restriction constraints, and more importantly for the purposes this paper, an importing schema can redefine a model group (without changing its name) to have different content.

SAML does not currently use model groups, so schema customizations can't redefine them.

Tools support

We don't know if tools support is an issue here. At a guess, proper support for redefinition is less universal than support for basic usage of model and attribute groups.

Interactions with other mechanisms

Model and attribute groups don't depend on other mechanisms, though they can be used in combination with type derivation.

Validation power

Model groups form content models, against which XML instances are validated by conforming XSD processors. Attribute groups contribute to the list of attributes on an element, against which instances are likewise validated. Any redefinitions of groups also get validated. There is no way to turn this validation off.

Flexibility

Model and attribute groups allow for a greater degree of flexibility in content model/attribute list modification than type derivation does. For example, you can use a model group for the middle of a content model, which then allows for redefinition of only that portion of the content model; type extension can only add to the end of a content model.

Reuse

Customizers' reuse of native constructs: Model and attribute groups ~~[@@?test this]~~ can, we believe, be reused by any schemas that import the schema in which the groups are originally defined.

Native reuse of others' constructs: Our schemas of interest (SAML) could reuse model and attribute groups defined by others' schemas. (It currently contains no model or attribute groups, native or otherwise.)

2.3 Element Substitution

In XSD, unless an element has been declared as "blocked," other elements with a compatible type (the same or a derived type) can appear in place of it. This setting can be controlled globally throughout a schema. Substitution can be thought of as a type-controlled, late-binding version of an `<xs:choice>` group, which allows a mutually exclusive choice among several elements.

SAML currently allows all elements to be heads of substitution groups.

Tools support

Some tools do not seem to fully support this mechanism, especially when combined with extension by restriction.

Interactions with other mechanisms

Substitution is typically used with derivable types, in order to substitute derived-type elements for base-type elements. It's possible to use substitution without derivation by substituting an element with an identical type, rather than a derived one, for the original element.

Validation power

The type compability of the substituted element for the head element is validated at compile time by XSD processors. This can't be turned off.

Flexibility

Some have commented that substitution is more limited than `<xs:choice>` groups [Kawaguchi] because it can only occur with compatible types rather than arbitrary types. However, substitution groups allow for customizers to extend "choices" after the original SAML schema creation, which `<xs:choice>` can't do. Note that type derivation used alone also allows for this possibility with the use of the `xsi:type` attribute.

Reuse

Does not apply.

2.4 Content Models Containing "Any" Wildcards

These are content models that contain the XSD `<xs:any>` and `<xs:anyAttribute>` particles. They create partially or fully "open" portions of a content model, where a variety of specific elements not foreseen by the original schema may appear.

The `<xs:anyAttribute>` wildcard is currently not used anywhere in SAML. The `<xs:any>` wildcard is used in content models as follows:

- `<Advice>` in the assertion schema contains `<Assertion>`, `<AssertionIDReference>`, and `##other` laxly validated elements
- `<StatusDetail>` in the protocol schema contains `##any` laxly validated elements

Tools support

There seems to be reasonable support for these features in tools.

Interactions with other mechanisms

When `<xs:any>` is used with the `##any` namespace setting in a content model mixed with native elements bound to derivable types, and a customizer derives a new type from one of these types and creates XML instances where the element bound to this derived type is supplied in an `<xs:any>` spot and uses `xsi:type`, the content model becomes "ambiguous." The workaround used in the Liberty Alliance customization was to invent an `<Extension>` element to hold the `<xs:any>` particle exclusively, avoiding mixing it together with native elements in a content model.

`<xs:any>` also causes ambiguity if a new type extends a sequence containing `<xs:any>` and adds new elements to the sequence. The new elements are not distinguishable from the original wildcard. This suggests that `<xs:any>` is not effective in conjunction with type derivation and may be best used in types that block (or lack the need for) derivation.

The effect of the `<xs:anyAttribute>` wildcard can be achieved implicitly when the **xs:anyType** ~~**[@@or xs:anySimpleType**? it appears so, but test]~~ is bound to an element.

An element must be global to satisfy an `<xs:any>` particle.

Validation power

The `<xs:any>` wildcard allows the schemographer to set the desired level of validation when XML content fills the "any" slot: `strict` (there must be a customized schema present that provides a definition for this subtree), `lax` (validate if a customized schema is present covering this subtree), or `none` (no validation will be done on this subtree). Note that use of `xsi:type` supersedes this optionality and forces validation.

Flexibility

The universe of allowed elements in an "any" slot is controllable by namespace (a list of namespaces to allow or bar, or arbitrarily namespaced elements with `##any`, or just elements from foreign namespaces with `##other`). The `##any` setting gives maximum choice to the customizer (but causes problems, as noted above under Interactions).

Also, as noted just above under Validation, the flexibility around validation power is greatest in this mechanism.

Reuse

Does not apply.

2.5 Elements Bound to "Any" Types

These are elements that are assigned the **xs:anyType** and **xs:anySimpleType** types. These are types that allow for "open" content but that also may be extended or restricted in order to close down some options.

The **xs:anySimpleType** type is not currently used anywhere in SAML. Two elements in the assertion schema are bound to **xs:anyType**:

- `<SubjectConfirmationData>`
- `<AttributeValue>`

Tools support

SAML has used this mechanism without any significant tools problems surfacing, though some older parsers do occasionally exhibit problems validating mixed and attribute content~~.~~

Interactions with other mechanisms

Using **xs:anyType** ~~[@@and also **xs:anySimpleType**?]~~ gives the effect of `<xs:anyAttribute>` for free on that element.

More tightly specified content models can be freely defined by extension of such an element, allowing second order extensions to take an open content model and precisely define its use in that context.

Validation power

Since essentially any content is compatible with such an element, an XSD processor will permit anything to appear without complaining. Thus, validation is essentially disabled for the element, unless `xsi:type` or element substitution is used.

Flexibility

An element bound to **xs:anyType** essentially stands in the original schema as a placeholder for whatever the same schema or an extension might define as a specific instance of that element. It thus provides a "conceptual" placeholder in a content model in which any XML can be used to represent the concept.

It also enables a combination of validating and non-validating behavior, which is useful when interpretation of the non-validated content is strictly optional or governed by other agreement mechanisms. SAML attribute values are an example of this.

Reuse

Does not apply.

2.6 Global Elements and Attributes

Elements and attributes that are global are allowed to be reused independently in the creation of a new foreign schema. Global elements can appear in `<xs:any>` wildcard content, and can serve as the root element of an XML document.

All of SAML's elements are global (and none of its attributes are). SAML has not consciously considered this to be an extensibility mechanism, but rather a general schema style that we hoped would be well supported by tools. However, it would be wise to examine the potential reuse of SAML elements in this light.

Tools support

There seems to be very good tools support for both native global elements and reuse of them in a foreign schema, as this was the first use case XSD needed to solve.

Interactions with other mechanisms

See [Maler] for a comparative analysis of global elements, local elements, named types, and anonymous types.

Validation power

Global elements (as well as local ones) have declarations and are bound to types, and are thus validated against these types. Unless an element is bound to something like **xs:anyType**, the validation can't effectively be turned off.

Flexibility

Global elements are more flexible than local ones, but are also more complex because of the usage (and reuse) choices they offer.

Reuse

The main benefit of global elements is their reuse – appearing in more than one content model of other elements – in the same or importing schemas.

2.7 URI-Based Identifiers and Namespaces~~SAML Namespaces~~

SAML ~~defines what it calls "namespaces," after the fashion of XML namespaces,~~ uses URI-based identifiers for interpreting selected SAML element and attribute content correctly. They are indicated through an attribute that contains a URI reference, which is designed to disambiguate which of the many possible meanings for a particular ~~subtree or keyword~~SAML construct is meant. It is also possible in some cases for customizers to assign a validatable type to the XML construct ~~containing the subtree or keyword~~, so that it can be syntactically validated.

In some of the cases where URI-based identifiers are used, SAML terms the mechanism "SAML namespaces." This name is typically used when the structure to be interpreted is inside the element on which the URI appears and when the content is keyword-like.

This mechanism differs from the other mechanisms discussed above~~,~~ in that it is a ~~SAML-specific~~ technique specific to the SAML vocabulary and not global to XSD. The SAML assertion schema uses ~~the namespace~~ this mechanism in ~~two~~ the following cases:

- ~~Names of~~ Interpreting the name of an **action** to be performed on a resource by referring to a URI-based identifier representing a set of names of actions ~~to be performed on resources~~(“action namespace”)
- Interpreting the name of an **attribute** by referring to a URI-based identifier representing a set of attribute names ~~Names of attributes~~(“attribute namespace”)
- Conveying the **authentication method** that was used for a subject in an authentication statement, or the authentication method being provided to confirm the binding of a subject to the bearer in any kind of assertion, by referring to a URI-based identifier representing an authentication method
- Interpreting the format of a **name identifier** by referring to a URI-based identifier representing the format

Tools support

This mechanism seems well supported, though it should be noted that the “semantics” gained from it is outside the knowledge of a generic XSD processor; it is specific to the SAML schemas.

Interactions with other mechanisms

The way SAML has defined this mechanism, it typically works hand in hand with assignment of a type for syntactic checking of the content, though this validation is not required.

Validation power

If the namespaced content is assigned a type as discussed above, it can be validated through normal XSD means. Otherwise, any validation must be application-specific. (See [Maler] for a description of an XML standard that uses a different mechanism to encourage syntactic validation much more strongly.)

Flexibility

Because this mechanism can be entirely divorced from XSD validation, and because anyone can invent a new namespace, it is very flexible. However, it may be that our underspecification of the attribute namespace case may have allowed too much flexibility, causing some divergence in usage.

Reuse

Namespaces defined by the SAML specifications and by customizers are available to be reused by others.

3 Recommendations for SAML Extensibility

The following sections contain our recommendations for modifications to the SAML schemas and specifications in order to support our extensibility goals more closely.

3.1 Recommendations Related to Native Types

Finality ✓

Keep all types non-final.

Rationale: The derivation mechanism is the most well-supported for allowing SAML constructs to be extended, and is known to interact successfully with other mechanisms SAML uses (or should use). The extreme power of model groups for extension has not proven to be needed, may not be fully supported in tools, and is an unfamiliar design pattern for most schemographers.

Action: None.

Abstraction ✓

Continue to make selected base types abstract and provide matching elements for them in an `<xs:choice>` group with non-abstract versions. The types that should be subjected to this treatment are the ones that have no useful semantics in SAML all by themselves, but have partial semantics that we feel may be useful to customizers are building blocks for their own SAML-derived work.

Rationale: This seems to be the only way to make a “deep” type hierarchy available, and experience with existing customizations seems to suggest it's working successfully.

Issues: Do we have a problem with the fact that this forces customizers to define derived types off of the abstract ones? We don't necessarily care if they do this, but the mechanism gives no choice.

Action: None.

Substitution ✓

Option 1: Block all substitution because its support, particularly in combination with other mechanisms, is iffy. The action in this case would be to set `blockDefault` attribute at the top level of the schema.

Option 2: Allow it, recognizing its limitations, since customizers have the right to choose it if they wish. No action would be required in this case.

Decision: The TC decided on 20 January 2004 to accept Option 1.

3.2 Recommendations Related to Open Content

xs:anyType vs. <xs:any>

Option 1: Migrate to using **xs:anyType** exclusively in order to stay away from known and potential problems with <xs:any>, allow for schema-less processing, and become consistent across the SAML schemas. The action in this case would be to work on the `<Advice>` declaration and **AdviceType** definition to add a new subelement that would be bound to the **xs:anyType** type, and to work on the `<StatusDetail>` element declaration to bind it to **xs:anyType** instead of **StatusDetailType**, which could be removed.

Option 2: Migrate to using `<xs:any>` exclusively in order to give additional validation-level flexibility to customizers, allow for schema-less processing, and become consistent across the SAML schemas. Note that the substitution-blocking decision made earlier mitigates the major problems with `<xs:any>`; however, it should be used as the lone content of whatever content model it appears in. The action in this case would be to change `<SubjectConfirmationData>` and `<AttributeValue>` to contain `<xs:any namespace=”##any” processContents=”lax”>` and to change `<Advice>` and **AdviceType** to add a new subelement that would contain only `<xs:any>`.

Option 3: Develop a set of criteria that provide guidance on when to use each mechanism, measure each current usage against the criteria, and synchronize the usage as necessary. NB: This set of criteria does not exist yet and would need to be developed! ~~I think o~~One of the likely criteria amounts to a guess as to whether the element containing the `<xs:any>` would be likely to need derivation. Metadata is an example. ~~I think a~~An element that amounts to an "element bag" seems like a poor candidate for extension and using `<xs:any>` seems like a way of allowing new optional "bag contents" from extension schemas to be defined while being ignored by older implementations.

Option 4: Do nothing and keep the current ad hoc split between the two mechanisms. This requires no action, but also provides no guidance if we add future elements that need flexible/foreign content.

~~**Wherever open content is desired, bind the element to xs:anyType rather than using the `<xs:any>` content model particle.**~~

~~**Rationale: The use of xs:anyType has proven to be more robust, flexible, and well-supported in tools than `<xs:any>`, and it doesn't have as many weird interactions with other mechanisms. It doesn't require customizers to build a formal schema, but this is of no concern to SAML. And it forces the open content to be sequestered in its own element, but this seems like good practice anyway for reasons of clear specification.**~~

~~**Action: This would require work on the `<Advice>` declaration and AdviceType definition to add a new subelement that would be bound to the xs:anyType type. It would also require work on the `<StatusDetail>` element declaration to bind it to xs:anyType instead of StatusDetailType, which could be removed.**~~

<xs:anyAttribute> ✓

Option 1: Consider adding `<xs:anyAttribute>` to all elements that aren't bound to **xs:anyType** (which implies `<xs:anyAttribute>`), because allowing for open attribute content is a harmless but powerful kind of extensibility that will allow SAML users to pick up and use interesting new global attributes produced in other venues. The action in this case would be to add the `<xs:anyAttribute>` particle to all complex type definitions except for those bound to **xs:anyType**.

Option 2: Don't do this because new attributes can always be added through type extension. No action would be required in this case.

Decision: The TC decided on 20 January 2004 to take no action now, but to consider adding `<xs:anyAttribute>` on a case-by-case basis as needed. (Note: David Orchard's article [Orchard] recommending wide application of `<xs:anyAttribute>` has come to the authors' attention since the decision was made.)

3.3 Recommendations Related to Global Elements

Restrained Globalness

Keep all elements global, but add prose saying which elements are allowed to be root elements, roots of SOAP payloads, and so on. Keep all attributes local, unless we see fit in the future to define any “common attributes” for use in SAML that we think have utility beyond SAML.

Rationale: SAML has succeeded with its global elements so far, and its vocabularies are small enough that there seems to be no pressure to have multiple elements with the same name but different content (the main value of local elements). However, without further qualification, global elements are more powerful than we intend most of our elements to be. Adding specification prose to prohibit usage of elements that are “subelements by nature” will avoid odd conformance scenarios. If we were to switch to using local qualified elements to enforce the “root element” constraint in a machine-readable way, we would probably cause any XSLT-based SAML processing (at the least) to have to change, since local element names can be globally non-unique and thus need to be additionally anchored (e.g., simple child ladders would have to be anchored all the way up to a global ancestor).

Action: This would require adding prose to mention that only certain elements – we propose to start with assertions, requests, responses, name identifiers, and attributes – MAY be reused in importing schemas and appear as root elements in an XML document. ~~I think this is potentially a bit strict in a few cases. I think <NameIdentifier> will be self-describing in most contexts, for one.Possibly <Attribute> would be another example?~~

3.4 Recommendations Related to SAML Namespaces

Attribute Clarity

> **Note:** Be prepared to recognize the "XML attribute" vs. "attribute of a SAML subject" distinction when reading this recommendation!

Add a new XML attribute on the `<AttributeDesignator>` and `<Attribute>` elements to capture the source of a SAML attribute.

Rationale: Rather than continuing to allow users to overload `AttributeNamespace` with multiple meanings, it's better to settle on exactly what "metadata" is needed to fully disambiguate an attribute name and allow SAML users to provide that metadata.

Action: This would require adding an attribute to **AttributeDesignatorType**. Note that any action here may have dependencies on the resolutions of the W-28* (attribute-related) work items.

Semantic vs. Syntactic Clarity

For SAML attributes and actions (and future any uses of the namespace mechanism), document more thoroughly the relationship between a namespace (identifying the "semantic" basis for the value) and the ability to assign a type to the element that holds the value of the attribute or action (identifying the syntactic constraints to which the value should be held during validation).

Rationale: The spec says little on this point today, and the mechanism is likely to be better understood and more widely and correctly used in future if we explain it better.

Action: This would require some writing only: no schema changes. Note that any action here may have dependencies on the resolutions of the W-28* (attribute-related) work items.

Namespace Terminology

Currently there's the namespace camp and the (unnamed) camp, with no solid connection between them.

Option 1: Unify the descriptions of all the parts of SAML that use the URI-based identitifer mechanism, focusing on the "identifier" language rather than the "namespace" language because it's more broad. This would require changing the names of `AttributeNamespace` and `ActionNamespace` and modifying any associated prose.

Option 2: Unify the descriptions of all the parts of SAML that use the URI-based identitifer mechanism, focusing on the "namespace" language rather than the "identifier" language because it's more evocative. This would require modifying prose that discusses URI-based identifiers and calling them "SAML namespaces."

Option 3: Do nothing because the distinction hasn't caused any problems to date.

3.5 Recommendations Related to Extension Understandability

Must Ignore

David Orchard's article [Orchard] recommends that vocabularies adopt an explicit model stating how to process extensions, with the default recommendation being the "Must Ignore" rule ("Document receivers MUST ignore any XML attributes or elements in a valid XML document that they do not recognize") in combination with extensibility techniques of the sort described in this document. Another choice might be a fallback model. Do we want to adopt this posture in some fashion? It might require some expression in the metadata, so that entities exchanging SAML can agree on which extensions they produce and expect.

4 References

[Kawaguchi] K. Kawaguchi. "W3C XML Schema Made Simple", XML.com, 6 June 2001. **http://www.xml.com/pub/a/2001/06/06/schemasimple.html**.

[LibProtSchema] S. Cantor et al., *Liberty ID-FF Protocols and Schema Specification*, Version 1.2, Liberty Alliance Project, 12 November 2003. http://www.projectliberty.org/specs/liberty-idff-protocols-schema-v1.2.pdf.

[Maler] E. Maler. "Schema Design Rules for UBL...and Maybe for You", XML 2002 paper, 8-13 December 2002. http://www.idealliance.org/papers/xml02/dx_xml02/papers/05-01-02/05-01-02.html. Note particularly Section 4. Also note that the schema examples in this section share a common error: `<xs:sequence>` is missing from around the content of `<xs:complexType>`.

[Obasanjo1] D. Obasanjo. "XML Schema Design Patterns: Is Complex Type Derivation Unnecessary?", XML.com, 29 October 2003. **http://www.xml.com/pub/a/2003/10/29/derivation.html**.

[Orchard] D. Orchard. "Versioning XML Vocabularies", XML.com, 3 December 2003. http://www.xml.com/pub/a/2003/12/03/versioning.html.

[SAMLConform] E. Maler et al. *Conformance Program Specification for the OASIS Security Assertion Markup Language (SAML).* OASIS, September 2003. Document ID oasis-sstc-saml-conform-1.1. http://www.oasis-open.org/committees/security/.

[SAMLCore] E. Maler et al. *Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML)*. OASIS, September 2003. Document ID oasis-sstc-saml-core-1.1. http://www.oasis-open.org/committees/security/.

[Schema1] H. S. Thompson et al. *XML Schema Part 1: Structures.* World Wide Web Consortium Recommendation, May 2001. http://www.w3.org/TR/xmlschema-1/.

A. Revision History

Rev	Date	By Whom	What
01	19 Jan 2003	Eve Maler, Scott Cantor	Initial draft.
02	28 Jan 2003	Eve Maler, Scott Cantor	Takes into account the feedback and decisions from the 20 January 2004 SSTC telecon, additional discussions between the authors, and an interesting article from David Orchard.

B. Notices

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification, can be obtained from the OASIS Executive Director.

OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to implement this specification. Please address the information to the OASIS Executive Director.