Service Provisioning Markup Language (SPML) Version 1.0

OASIS Committee Specification, 3 June 2003

Document identifier: cs-pstc-spml-core-1.0.doc

Location: http://www.oasis-open.org/committees/provision/docs/

Send comments to: pstc-comment@lists.oasis-open.org

Editor:

Darran Rolls, Waveset Technologies (Darran.Rolls@waveset.com)

Contributors:

Archie Reed, Critical Path
Doron Cohen, BMC
Gavenraj Sodhi, Business Layers
Gerry Woods, IBM
Hal Lockhart, BEA
Jeff Bohren, OpenNetwork Technologies
Jeff Larson, Waveset Technologies
Jesus Fernandez, Computer Associates
Matthias Leibmann, Microsoft
Mike Polan, IBM
Paul Madsen, Entrust
Rami Elron, BMC
Tony Gallotta, Access, IBM
Yoav Kirsh, Business Layers

Abstract:

This specification defines the concepts, operations deployment and XML schema for an XML based provisioning request and response protocol.

Status:

This is a candidate Committee Specification that is undergoing a vote of the OASIS membership in pursuit of OASIS Standard status.

If you are on the provision list for committee members, send comments there. If you are not on that list, subscribe to the provision-comment@lists.oasis-open.org list and send comments there. To subscribe, send an email message to provision-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.

Table of contents

1. Introduction (non-normative)

1.1. Glossary

1.1.1 Preferred terms

1.1.1.1	Account	The set of attributes that together define a user's access to a given service. Each service may define a unique set of attributes to define an account. An account defines user or systems access to a resource or service.
1.1.1.2	Access Rights	A description of the type of authorized interactions a subject can have with a resource. Examples include read, write, execute, add, modify, and delete.
1.1.1.3	Administrator	A person who installs or maintains a system (e.g. a SPML-based provisioning system) or who uses it to manage system entities, users, and/or content (as opposed to application purposes. See also End User). An administrator is typically affiliated with a particular administrative domain and *may* be affiliated with more than one administrative domain.
1.1.1.4	Attribute	A distinct characteristic of an object. An object's attributes are said to describe the object. Objects' attributes are often specified in terms of their physical traits, such as size, shape, weight, and color, etc., for real-world objects. Objects in cyberspace might have attributes describing size, type of encoding, network address, etc. Which attributes of an object are salient is decided by the beholder.
1.1.1.5	Authentication	To confirm a system entity's asserted principal identity with a specified, or understood, level of confidence.
1.1.1.6	Authorization	The process of determining which types of activities are permitted. Usually, authorization is in the context of authentication. Once you have authenticated an entity, the entity may be authorized different types of access or activity. The (act of) granting of access rights to a subject (for example, a user, or program).
1.1.1.7	Credential	Data that is transferred to establish a claimed principal identity.
1.1.1.8	End User	A natural person who makes use of resources for application purposes (as opposed to system

		management purposes. See Administrator, User).
1.1.1.9	External Enterprise	Environment which may contain many or all of the following: Managed Services, contractors, temporary employees, multiple organizations, private to public registry systems.
1.1.1.10	Identity	The unique identifier for a person, resource or service.
1.1.1.11	Login Logon Signon	The process of presenting credentials to an authentication authority, establishing a simple session, and optionally establishing a rich session.
1.1.1.12	Principal	A system entity whose identity can be authenticated
1.1.1.13	Provisioning	The process of managing attributes and accounts within the scope of a defined business process or interaction. Provisioning an account or service may involve the Creation, modification, deletion, suspension, restoration of a defined set of accounts or attributes. The process of provisioning an account or service may involve the execution of a defined business or system process.
1.1.1.14	Provisioning service (PS)	Any system entity that supports the receipt and processing of SPML artifacts
1.1.1.15	Provisioning Service Point (PSP)	Reference to a given Provisioning Service
1.1.1.16	Provisioning Service Target (PST)	A resource managed by a PSP. Example PST's are directories, NIS instances, NT domains, individual machines, applications or groups of application and settings that together denote a service offering, appliances or any provisioning target.
1.1.1.17	Requesting Authority (RA)	Party or system that is authorized to request a resource for the party.
1.1.1.18	Resource	An application or service supporting the provisioning or account or attribute data.
1.1.1.19	Service Object	An object of a defined SPML service. Example: an email account jdoe@myservice.com from the published service schema “interopUser”
1.1.1.20	Session	A lasting interaction between system entities, often involving a user, typified by the maintenance of some state of the interaction for the duration of the interaction.
1.1.1.21	Subject	A principal, in the context of a security domain, about which a given provisioning request is made or requested.

1.1.1.22	System	An active element of a computer/network system--e.g., an automated process or set of processes, a subsystem, a person or group of persons—that incorporates a distinct set of functionality.
1.1.1.23	Service	A specific type of resource that is not physically obtained by a user, but is accessed periodically by the user
1.1.1.24	Security	Security refers to a collection of safeguards that ensure the confidentiality of information, protect the system(s) or network(s) used to process it, and control access to it (them). Security typically encompasses the concepts/topics/themes of *secrecy*, *confidentiality*, *integrity*, and *availability*. It is intended to ensure that a system resists potentially correlated attacks.
1.1.1.25	SPML	Service Provisioning Markup Language. The name for the XML framework proposed by the OASIS PSTC
1.1.1.26	User	A natural person that makes use of a system and its resources for any purpose See also Administrator, End User.

1.2. Notation

This specification contains schema conforming to W3C XML Schema and normative text to describe the syntax and semantics of XML-encoded policy statements.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in IETF RFC 2119 **[RFC2119]**

"they MUST only be used where it is actually required for interoperation or to limit behavior which has potential for causing harm (e.g., limiting retransmissions)"

These keywords are thus capitalized when used to unambiguously specify requirements over protocol and application features and behavior that affect the interoperability and security of implementations. When these words are not capitalized, they are meant in their natural-language sense.

```
Listings of SPML schemas appear like this.
```

```
Example code listings appear like this.
```

Conventional XML namespace prefixes are used throughout the listings in this specification to stand for their respective namespaces as follows, whether or not a namespace declaration is present in the example:

- The prefix dsml: stands for the Directory Services Markup Lanuage namespace **[DSML].**
- The prefix saml: stands for the SAML assertion namespace [**SAML**].
- The prefix ds: stands for the W3C XML Signature namespace [**DS**].

- The prefix xs: stands for the W3C XML Schema namespace [**XS**].

This specification uses the following typographical conventions in text: <SPMLElement>, <ns:ForeignElement>, **Attribute**, **Datatype**, OtherCode. Terms in ***italic bold-face*** are intended to have the meaning defined in the Glossary.

1.3. Schema organization and namespaces

The SPML core operations schema syntax is defined in a schema associated with the following XML namespace:

```
urn:oasis:names:tc:SPML:1:0
```

2. Background (non-normative)

In late 2001, the OASIS Provisioning Services Technical Committee (PSTC) was formed to define an XML-based framework for exchanging user, resource, and service provisioning information. This section is intended to provide a high level definition of provisioning within the context of the PSTC and an overview of the scope of the SPML specification.

2.2. What does service provisioning mean?

Service provisioning means many different things to many different people. In the context of this specification it refers to the "preparation beforehand" of IT systems' "materials or supplies" required to carry out some defined activity. It goes further than the initial "contingency" of providing resources, to the onward management lifecycle of these resources as managed items. This could include the provisioning of purely digital services like user accounts and access privileges on systems, networks and applications. It could also include the provisioning of non-digital or "physical" resources like the requesting of office space, cell phones and credit cards.

The following short definition has been adopted by the Provisioning Services Technical Committee as its formal definition of the general term "provisioning":

"Provisioning is the automation of all the steps required to manage (setup, amend & revoke) user or system access entitlements or data relative to electronically published services".

2.3. What is a provisioning system?

At this stage it is not necessary to define the implementation or physical makeup of a service provisioning system. Simply assume the existence of a network service whose sole purpose is the execution and management of provisioning requests. A given Requesting Authority (client) sends the provisioning service a set of requests in the form of a well formed SPML document. Based on a pre-defined service execution model, the provisioning service takes the operations specified within the SPML document and executes provisioning actions against pre-defined service targets or resources.

Figure 1 shows a high-level schematic of the operational components of an SPML model system. In SPML request flow A, the Requesting Authority (client) constructs an SPML document subscribing to a pre-defined service offered by Provisioning System One (PS One). PS One takes the data passed in this SPML document, constructs its own SPML document and sends it to PST one (SPML request flow B). PST One represents an independent resource that provides an SPML-compliant service interface. In order to fully service the initial Requesting Authority's request, PS One then forwards a provisioning request (SPML request flow C) to a second network service called Provisioning System Two (PS Two). PS Two is autonomously offering a provisioning service it refers to as Resource E. In this case, Resource E is a relational database within which PS Two creates some data set. Having successfully received PS One's request, PS Two carries out the implementation of its service by opening a JDBC connection to Resource E and adding the relevant data (data flow D).

In this example, the SPML document flow follows a simple request/response protocol that supports both synchronous and asynchronous operations. Importantly, these SPML flows are initiated unidirectionally. When PS One made a request of PS Two, it assumed the role of a Requesting Authority and initiated its own request/response flow with its chosen service point. When PS Two implemented its service at Resource E, it DID NOT use an SPML protocol message as Resource E did not support an SPML interface.

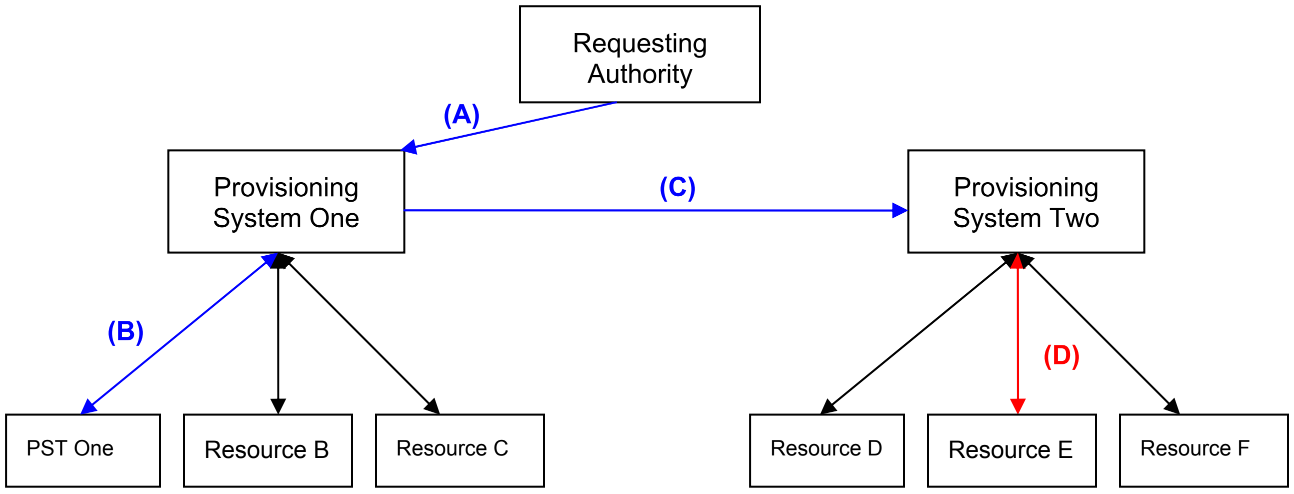


Figure 1. Provisioning Systems.

2.4. Why do we need service provisioning standards?

There are several areas of provisioning systems that would benefit from standardization. XRPM [XRPM] and ADPr [ADPR] both addressed the business needs and possible benefits for establishing standardization in this space. Each initiative identified this need at opposite ends of the provisioning scenario depicted in Figure 1. XRPM set out to define a standard for interoperability and functioning between Provisioning Systems. ADPr set out to define a standard for interoperability and functioning between the Provisioning System and the managed resource.

The PSTC was formed to address the specification of a single XML-based framework for the exchange of information at all levels. This is achieved at the protocol level by allowing a Provisioning Service Target (resource) to adopt the role of a Provisioning Service Point (a server), respond to client requests and operate as a full service point responsible for a single service or resource, itself.

2.5. Requirements

The following requirements contributed to the generation of this specification. Their source can be found in the committee mail list archive [ARCHIVE-1] and in the official SPML requirements document [SPML-REQ].

2.5.1 To define an extensive set of use cases that model the functional requirements of the proposed protocol and to see these use cases operable by implementing the resulting specification.

2.5.2 To define an XML Schema based protocol for exchanging provisioning requests between a Requesting Authority (RA) and a Provisioning Service Point (PSP).

2.5.3 To define an XML Schema based protocol for exchanging requests provisioning requests between a PSP and a Provisioning Service Target (PST) AND if possible to implement this and requirement 1 (above) in a single protocol.

2.5.4 To provide a query model that MAY allow a RA to discover details about those provisioning elements it is authorized to see and act upon at a given PSP. Implicitly, the “decision" on what services to display to what RA’s lies with the implementation and authorization model of the PSP provider.

2.5.5 To provide a model that allows a RA and a PSP to dynamically discover the required data values for a given provisioning action.

2.5.6 To provide consideration for the security and general operational concerns of such an exchange system.

2.5.7 To provide guidelines on binding SPML to the SOAP and HTTP protocols.

2.5.8 To provide an open extensible solution that is independent of any one vendors implementation or solutions model.

2.5.9 To provide a transactional element to the request/response model that allows for the exchange of ordered batches of requests.

2.5.10 To deliver a solution in a timely manor.

2.5.11 To where possible and reasonable to re-use and extend existing standards efforts for the benefit of the SPML solution.

2.5.12 To provide a standard suitable for use both inside a single organization or enterprise and between multiple separate organizations/enterprises operating on separate network infrastructures.

2.5.13 To provide an open protocol that does not dictate the underlying infrastructure or technology used by the implementer of RA, PSP or PST entities to support that protocol.

2.6. Use Cases

The PSTC has produced a number of use cases that define the operational requirements of the SPML V1.0 specification. The SPML v.10 use cases [PSTC-UC] can be found on the PSTC web site. Section eight of this document provides a two working examples taken from several of these use cases.

3. Models (non-normative)

The following sections describe the general object model and operational model for an SPML system are described in the following sub-sections.

3.1 Protocol Overview

The general model adopted by this protocol is one of clients performing protocol operations against servers. In this model, a client issues an SPML request describing the operation to be performed at a given service point. The service point is then responsible for performing the necessary operation(s) to constitute the implementation of the requested service. Upon completion of the operation(s), the service point returns to the client an SPML response detailing any results or errors pertinent to that request.

In order to promote standardization of the service subscription and provisioning interface, it is an active goal of this protocol to minimize the complexity of the client interface in order to promote widespread deployment of applications capable of issuing standardized service provisioning requests. With this goal in mind SPML builds on a simplistic core operations model in which the semantics of an individual provisioning action lay in the definition of the underlying service schema. The core operations schema provides a small number of generic operations (Add, Modify, Delete, Search) and an open model for the definition and discovery of that schema as a set of simple name=(multi)value pairs. To complement this, SPML V1.0 also provides an operations extension model based on an <*ExtendedRequest*> operation that allows individual providers to define new operations that do not overlap with V1.0 core operations.

SPML V1.0 provides both a synchronous and asynchronous batch request model. However, there is no requirement for a blocking synchronous behavior on the part of either clients or servers in either operating model. Requests and responses for multiple operations may be freely exchanged between a client and server in any order, provided the client eventually receives a response for every request that requires one.

The SPML V1.0 XML Schema follows an “Open Content Model”. In this model, any SPML element in an XML document can have additional child elements and attributes that are not declared in the core SPML V1.0 schema. In contrast, a “Closed Content Model”, would prevent the inclusion of any elements not defined in the SPML core XSD.

3.2 Domain Model

The following section introduces the main conceptual elements of the SPML domain model. The Entity Relationship Diagram (ERD) in Figure 2 shows the basic relationships between these elements.

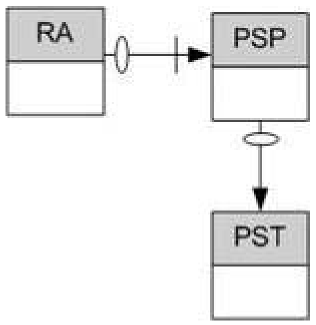


Figure 2. Conceptual system elements

3.2.1 Introduction to RA

SPML introduces the concept of a Requesting Authority (RA). A Requesting Authority is a software component that issues well formed SPML requests to a known SPML service point. In an end-end integrated provisioning scenario, any component that issues SPML requests is said to be operating as a Requesting Authority. Implicit in this description is an established trust relationship between the RA and its corresponding service point. The details of establishing and maintaining this trust relationship is out of scope for this specification. Example RAs are portal applications that broker the subscription of client requests to system resources AND service subscription interfaces within an Applications Service Provider.

3.2.2 Introduction to PSP

SPML introduces the concept of a Provisioning Service Point (PSP). A Provisioning Service Point is a software component that listens for, processes and returns the results for well formed SPML requests, from a known SPML Requesting Authority. In an end-end integrated provisioning scenario, any component that receives and processes SPML request is said to be operating as a PSP. Implicit in this description is an established trust relationship between the PSP and its corresponding Requesting Authority. The details of establishing and maintaining this trust relationship is out of scope for this specification. Example PSPs are Enterprise Provisioning systems.

3.2.3 Introduction to PST

SPML introduces the concept of a Provisioning Service Target (PST). A Provisioning Service Target is a software component that is seen as the end point for a given provisioning action. Within the SPML object model PST's are abstract entities that implement some part of the physical service provisioning action. Frequently a PST will be a traditional user account source like an NT domain or directory. The SPML model does not restrict a PST outside of its ability to be uniquely identifiable to a PSP and the ability to (through some arbitrary method) guarantee the unique identification of data entries (or records) within its implementation.

Note that a PST is an abstract element in the SPML object model. No specific protocol flows are unique to a request/response to a modeled PST. In fact, when a PSP makes a down-stream request to a modeled PST, in protocol terms the PSP is functioning as an RA (or RA) issuing operational requests and the PST is functioning as an PSP responding to those requests. As such, a PST is simply a useful model abstract that helps to explain the requirements and operation of an end-to-end provisioning scenario.

3.2.4 Introduction to PSO-ID

SPML introduces the concept of a Provisioning Service Object Identifier (PSO-ID). A PSO-ID represents a unique identifier for a collection of individual provisioning requests. An example explains this best.

Consider the provisioning of IT resource accounts for a new corporate user. The new user requires an account on a Windows NT domain, a Lotus Notes server, a corporate directory server and a UNIX file server. In this example the RA would present the PSP with its own unique identifier for the “corporate user”, say a full name, a list of the PSTD-ID’s it would like to create on the target systems (see below) and the set of attributes required to complete the provisioning request. In this example, the PSO-ID would be the full name specified by the RA. The PSO-ID would be used to relate the created PSTD-ID’s together. This relationship could be maintained by both the RA and the PSP, the details of which is deliberately left un-defined in the SPML protocol.

PSO-ID’s can be defined by the RA. It is therefore the responsibility of the PSP implementation to guarantee this. PSO-ID’s can also be generated by the PSP at provisioning time and reported back to the RA as part of the response element.

The PSO-ID therefore represents the unique identification for a set of provisioned data throughout the life cycle of that PSP.

Figure 3 shows some of the possible relationships between a RA, PSP, PSO-ID, PST and PSTD-ID.

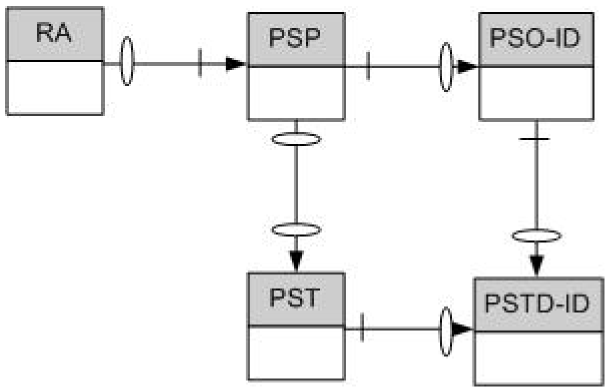


Figure 3. High-level system element relationships

The PSO-ID relationship with PSTD-ID’s is one on many possible relationships that could be modeled behind an SPML compliant service interface. In the context of enterprise IT resource provisioning it is an important one and hence is explicitly called out in the SPML domain model. Note that although important, this relationship is not implicitly bound into the SPML protocol. In order to accurately model these relationships, the definition of concepts like the PSO-ID are dropped to the data schema layer and are hence modeled by service providers in the definition of service data schema.

3.2.5 An Introduction to PSTD-ID

SPML introduces the concept of a Provisioning Service Target Data Identifier or PSTD-ID. A PSTD-ID is a unique identifier for a data set (account or managed data) on a PST. An example of a PSTD-ID on a UNIX/Linux server would be the UID; an example of a PSTD-ID for a directory entry would be a Distinguished Name (DN). In some cases PSTD-ID's are specified by the RA issuing the SPML request. In others the PSTD-ID is set by the PSP/PST.

It is assumed that a PSTD-ID is unique to a PST (if not implemented by the native resource then implemented by the functioning PST/PSP implementation through some custom namespace mechanism).

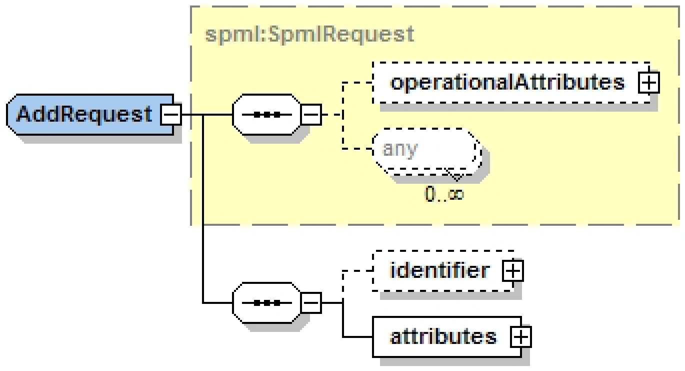
The simple ERD shown in Figure 3 shows some of the possible relationships between a RA, PSP, PSO-ID, PST and PSTD-ID. Remember that these relations are not always directly reflected in the SPML 1.0 protocol; often they explain the model behavior of the entire system.

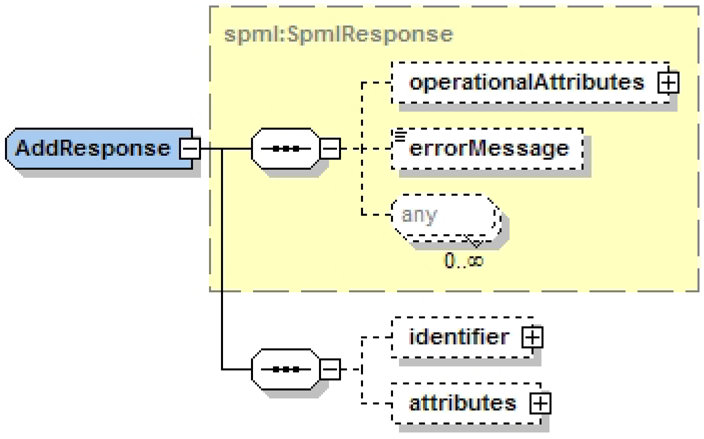
3.3 Operations Overview

This section provides a non-normative discussion of the specific operations defined in SPML version 1.0 and describes them in relative terms to the overall design and purpose of SPML.

The following sections provide an introduction to the core operations, request-response protocol and service schema definition language.

3.3.1 SPML Add Operation





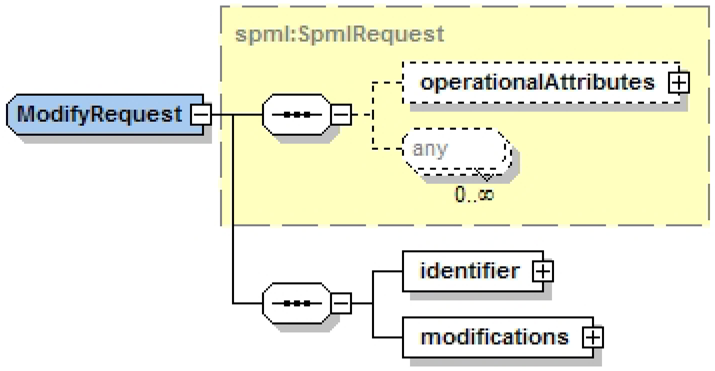
The <*AddRequest*> element allows an RA to request the creation of a new object of an object as defined by its object classes. The <*attributes*> sub element is used to envelope the set of name=(multi)value pairs required to subscribe to a given published service. Note the use of the special attribute "**objectclass**". This attribute is used to define the target service schema the RA wishes to create an object of. The example below shows a request to "**create**" an object of the "**emailUser**" service. The RA supplies the "**cn**" "**gn**" and "**sn**" attributes required to subscribe to this service. By issuing this request, the RA hopes to create a new email account for the "**cn**" Jane Doe.

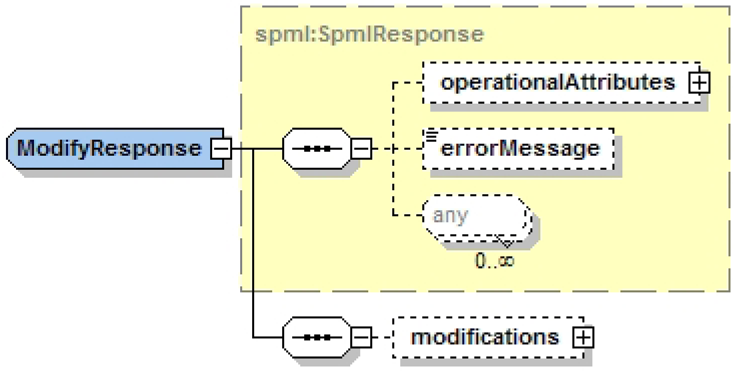
```
<addRequest>
    <attributes>
        <attr name="objectclass">
          <value>emailUser</value>
        </attr>
        <attr name="cn">
            <value>Jane Doe</value>
        </attr>
        <attr name="gn">
            <value>Jane</value>
        </attr>
        <attr name="sn">
            <value>Doe</value>
        </attr>
    </attributes>
</addRequest>
```

The resulting <*AddResponse*> element returned by the PSP supplies the RA with a result code attribute to indicate that the add request was successfully processed. In this example, the PSP returns an <*identifier*> element containing the primary record identifier for the newly created service object (PSTD). This value will be required for subsequent operations on this newly created PSTD-ID. The response element also includes an optional <*attributes*> element to envelope a set of data generated by the functioning PST. In this example, the PSP uses these returned attributes to notify the RA that the "mailBoxLimit" attribute was automatically set to "50MB".

```
<addResponse result = "urn:oasis:names:tc:SPML:1:0#success">
    <identifier type = "urn:oasis:names:tc:SPML:1:0#EMailAddress">
        <spml:id>Jane.Doe@acme.com</id>
    </identifier>
    <attributes>
        <attr name="mailBoxLimit">
            <value>50MB</value>
        </attr>
    </attributes>
</addResponse>
```

3.3.2 SPML Modify Operation





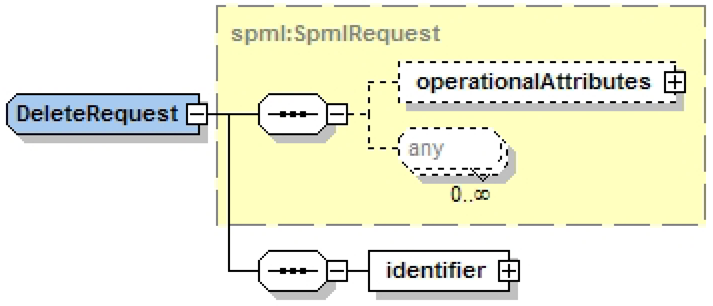
The <*ModifyRequest*> element is used by a RA to specify a change request to a PSP or PST. Implicit in the modification of a service object is the identification of the exact PSTD-ID to be modified. In this example, the RA specifies an <*IdentifierType*> of EmailAddress and uniquely identifies the PSTD via the <*id*> of Jane.Doe@acme.com. The <*modifications*> element is used to envelope the set of modified attributes for this PSTD. Here the RA requests the modification of the "**mailBoxLimit**" attribute to the value of "**100MB**".

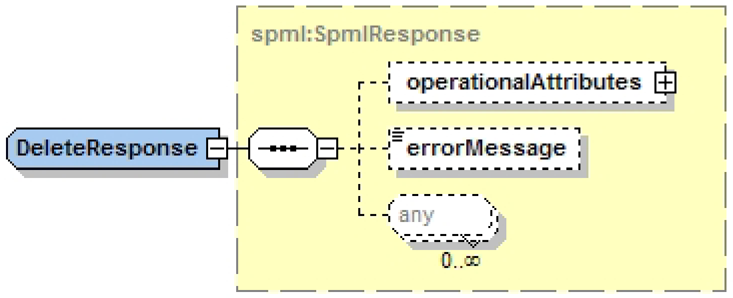
```
<modifyRequest>
    <identifier type = "urn:oasis:names:tc:SPML:1:0#EMailAddress">
        <id>Jane.Doe@acme.com</id>
    </identifier>
    <modifications>
        <modification name="mailBoxLimit">
            <value>100MB</value>
        </modification>
    </modifications>
</modifyRequest>
```

Using the <*ModifyResponse*> element, the PSP returns the status of the <*ModifyRequest*> as shown below.

```
<modifyResponse result = "urn:oasis:names:tc:SPML:1:0#success" />
```

3.3.3 SPML Delete Operation





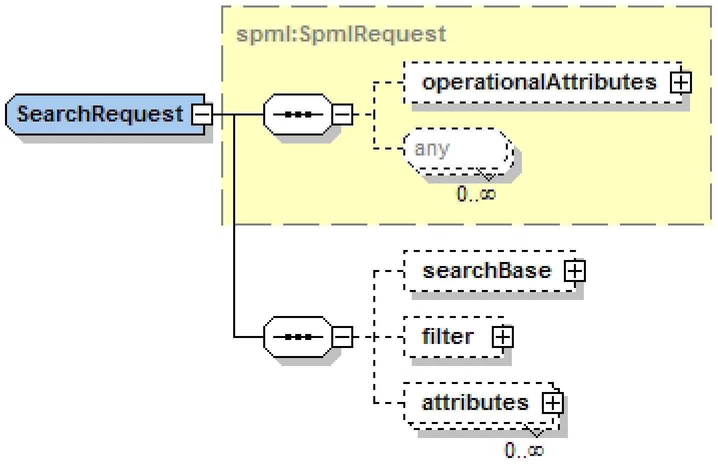
The <*DeleteRequest*> element is used by a RA to request the deletion of a specific PSTD-ID. Implicit in the deletion of a service object is the identification of the exact PSTD-ID to be deleted. In this example, the RA specifies an <*IdentifierType*> of EmailAddress and uniquely identifies the PSTD via the <*id*> of Jane.Doe@acme.com.

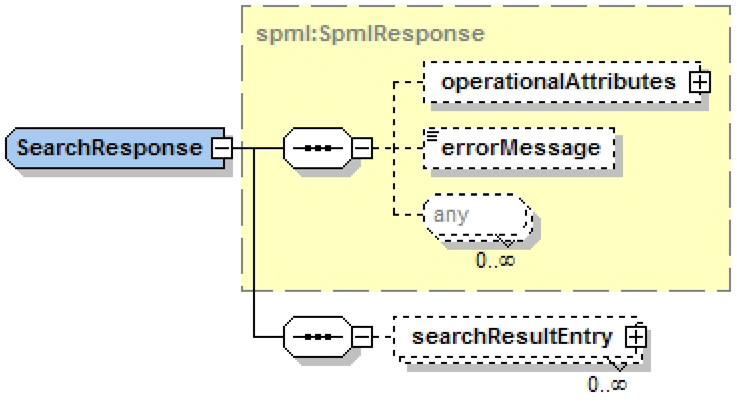
```
<deleteRequest>
    <identifier type = "urn:oasis:names:tc:SPML:1:0#EMailAddress">
        <id>Jane.Doe@acme.com</id>
    </identifier>
</deleteRequest>
```

Using the <*DeleteResponse*> element, the functioning PSP returns the status of the <*DeleteRequest*>.

```
<deleteResponse result = "urn:oasis:names:tc:SPML:1:0#success" />
```

3.3.4 SPML Search Operations





The <*SearchRequest*> element is used to allow a SPML V1.0 client to request a search be performed on its behalf by a PSP. A search is used to read attributes from service objects managed by the PSP. A search request specifies a <*searchBase*> as the identifiable start point for a search operation, search criteria <*filter*> and the attributes to be returned <*attributes*>. Note you can not specify what values can be returned. If no <*attributes*> are specified on the search request, all attributes are returned in the corresponding <*SearchResponse*>. In the following example, the RA requests a search for all accounts in the acme.com domain that have a "**cn**" attribute that ends in "**Doe**". The <*filter*> element defines the search criteria. This sample request also defines the list of data values it wants returned in the <*SearchResponse*>. Using the <*attributes*> sub element the client constrains the target schema attributes returned in the results to "**cn**" and "**email**":

```
<searchRequest>
    <searchBase type = "urn:oasis:names:tc:SPML:1:0#URN">
        <spml:identifier>urn:acme.com</id>
    </searchBase>
    <filter>
        <substrings name="cn">
            <final>Doe</final>
        </substrings>
    </filter>
    <attributes>
```

```
        <attribute name="cn"/ >
        <attribute name="email/>
    </attributes >
</searchRequest>
```

The results of the search request are returned to the RA using the <SearchResponse> element. Search responses include an "**result**" attribute and a series of <*SearchResultEntry*> elements. Each entry returned in a <*SearchResultEntry*> will contain all requested attributes, complete with associated values, as specified in the attributes field of the Search Request. The specific return of attributes is subject to access control and administrative policy defined by the PSP.

The following example shows the result of the search requested above. Note that the "**result**" defines the successful completion of the request and that each <*SearchResultEntry*> includes an <*identifier*> with its corresponding <*IdentifierType*> to uniquely identify each record. Following this identifier is the list of <*attributes*> requested in the search request. The ordering of the returned attributes corresponds to the order in which they are requested in the <*attributes*> element in the search request.

```
<searchResponse result = "urn:oasis:names:tc:SPML:1:0#success">
    <searchResultEntry>
        <identifier type = "urn:oasis:names:tc:SPML:1:0#EMailAddress">
            <identifier >Jane.Doe@acme.com</id>
        </identifier>
        <attributes>
            <attr name="cn"><value>Jane Doe</value></attr>
            <attr name="email"><value>Jane.Doe@acme.com</value></attr>
        </attributes>
    </searchResultEntry>
    <searchResultEntry>
        <identifier type = "urn:oasis:names:tc:SPML:1:0#EMailAddress">
            <identifier >John.Doe@acme.com</id>
        </identifier>
        <attributes>
            <attr name="cn"><value>John Doe</value></attr>
            <attr name="email"><value>John.Doe@acme.com</value></attr>
        </attributes>
    </searchResultEntry>
</searchResponse>
```

3.3.5 Operational Attributes

SPML V1.0 provides a convenient way for requestors and responders to provide additional operational attributes when issuing and responding to SPML requests. The <*SpmlRequest*> and <*SpmlResponse*> elements both provide an <*operationalAttributes*> sequence element that allows either functioning party to specify additional information pertinent to the execution of a given operation. In the example below, the functioning RA issues the delete request as defined in section 7.3.3 or this document. Notice that in this example, the RA adds <*operationalAttributes*> to the delete request specifying an archival policy for the deleted mail box and a time at which the request should be executed.

```
<deleteRequest>
        <identifier type= "urn:oasis:names:tc:SPML:1:0#EMailAddress">
                <id>Jane.Doe@acme.com</id>
        </identifier>
        <operationalAttributes>
```

```
                    <attr name="retainMailbox">
                              <value>true</value>
                    </attr>
                    <attr name="executeJulianDate">
                              <value>2452842</value>
                    </attr>
          </operationalAttributes>
</deleteRequest>
```

3.3.6 Error Conditions

A PSP reports operational request errors back to the RA using specific attributes on the <*SpmlResponse*> definition. When requests are collected into an <*BatchRequest*>, errors relating to each individual request in the batch are handled as described here. Additional error handling attributes that apply to the entire batch are set on attributes of the <*BatchRequest*>. See section 7.5 of this document for full details of batch handling and error profiling.

Each <*SpmlResponse*> element can contain three areas for error reporting. The first is a mandatory ″**result**″ attribute included with every response element. This details a value from an enumerated list of specific status codes for the requested operation (**success**, **failure** or **pending**). The intention of the result code is to provide a clear (almost) binary statement on the status of the request. This basic request status is optionally accompanied by the second attribute "**error**″ that details an <*ErrorCode*> from a list of error codes. The error code is used to expand on the result code to detail the reason for the failure. Lastly, the PSP can optionally provide a sequence of "***errorMessage***″ attributes in the <*SpmlResponse*>. These attributes are then available for the SPML service to report back application and action specific messages for the information of the requesting client.

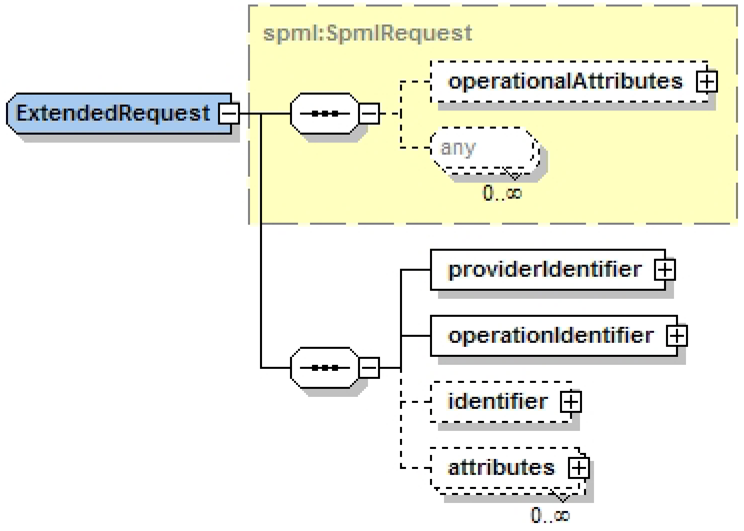
```
    <xsd:simpleType name="ResultCode">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#success"/>
            <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#failure"/>
            <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#pending"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="ErrorCode">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#malformedRequest"/>
            <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#unsupportedOperation"/>
            <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#unsupportedIdentifierType"/>
            <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#noSuchIdentifier"/>
            <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#noSuchRequest "/>
            <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#customError"/>
        </xsd:restriction>
    </xsd:simpleType>
```

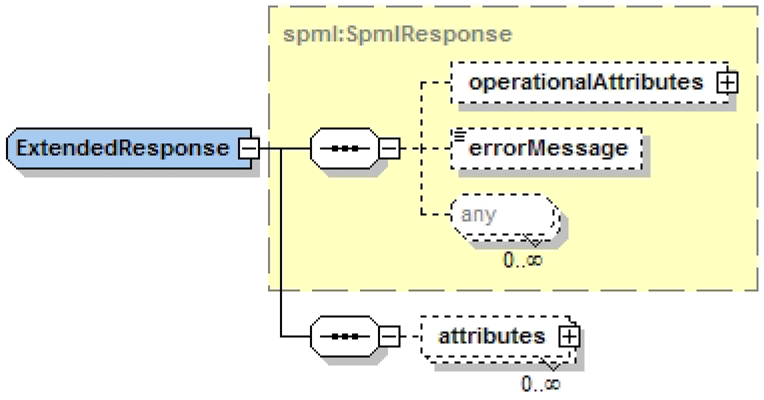
Syntax errors occur when an RA issues what is fundamentally a malformed SPML request. In these cases, rather than simply disregarding the request, the PSP sends a response back to the client with a specific "**error**" attribute value of "**malformedRequest**" to aid debugging and re-issuance at the client. Below is a sample deletion response that shows a syntax error in the request.

```
<deleteResponse result = "urn:oasis:names:tc:SPML:1:0#malformedRequest" />
```

To further enable implementation specific error handling SPML V1.0 provides the "**ErrorCode**" value of "**urn:oasis:names:tc:SPML:1:0#customError**". A PSP may use this error code in conjunction with <*operationalAttributes*> from the <*SpmlResponse*> element to report implementation specific error codes.

3.3.7 SPML Extended Operations





An extension mechanism has been added to the SPML v1.0 specification to allow additional operations to be defined for services not available elsewhere in this protocol. The extended operation allows clients to make requests and receive responses with predefined syntaxes and semantics. Extended request operations may be defined by other standards bodies or may be private to particular vendor implementations. A RA uses the <*ExtendedRequest*> element to request an extended SPML operation.

Each new extended request operation defines a unique <*operationIdentifier*> that names and identified the new operation. In the example below, the PSP has provided an extended operation "**urn:acme.com.mailservice.ops:purge**" that allows its clients to request the purging of a defined mail box. In this implementation no service schema attributes are required when requesting this operation. The RA simply provides the unique <*identifier*> for the PSTD-ID it wished to operate against.

```
<extendedRequest>
      <providerIdentifier providerIDType = "urn:oasis:names:tc:SPML:1:0#OID">
          <providerID>1.3.6.1.4.868.2.4.1.2.1.1.1.3.3562</providerID>
      </providerIdentifier>
      <operationIdType = "urn:oasis:names:tc:SPML:1:0#URN">
          <operatIdentifier>urn:acme.com.mailservice.ops:purge</operationID>
     </identifier>
     <identifier type= "urn:oasis:names:tc:SPML:1:0#EMailAddress">
          <id>Jane.Doe@acme.com</id>
     </identifier>
</extendedRequest>
```

Using the <*ExtendedResponse*> element, the functioning PSP returns the status of the <*ExtendedRequest*> as shown below.

```
<extendedResponse  result = "urn:oasis:names:tc:SPML:1:0#success" />
```

SPML V1.0 also defines a mandatory <*providerIdentifier*> sub-element that allows an extended request provider to annotate their new operations and provide an aggregation mechanism such that a set of extended requests can be correlated back to a specific provider. In the examples below, the <*poviderIdentifier*> "1.3.6.1.4.868.2.4.1.2.1.1.1.3.3562" is used to collect together the two new operations ***purge*** and ***compress***.

For a full explanation of <*providerIdentifier*> and <*operationIdenfifier*> elements see section 3.4.2 of this document.

```
<extendedRequest>
     <providerIdentifier providerIDType = "urn:oasis:names:tc:SPML:1:0#OID">
          <providerID>1.3.6.1.4.868.2.4.1.2.1.1.1.3.3562</providerID>
     </providerIdentifier>
     <operationIdentifier operationIDType = "urn:oasis:names:tc:SPML:1:0#URN">
          <operationID>urn:acme.com.mailservice.ops:purge</operationID>
     </identifier>
     <identifier type= "urn:oasis:names:tc:SPML:1:0#EMailAddress">
          <id>Jane.Doe@acme.com</spml:id>
     </identifier>
</extendedRequest>
```

3.4 SPML Identifiers

3.4.1 Target Identifiers

In the SPML V1.0 each of operations Add, Modify, Delete and Extended requests require the RA to specify a unique <*Identifier*> element which defines the PSTD-ID the requested operation applies. This is used to directly identify the PSTD to be operated against. The <*IdentifierType*> type defines the allowable types for an <Identifier>. The following types are supported:

```
     <xsd:simpleType name="IdentifierType">
          <xsd:restriction base="xsd:string">
```

```
            <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#EMailAddress"/>
            <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#DN"/>
            <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#UserIDAndOrDomainName"/>
            <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#LibertyUniqueID"/>
            <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#PassportUniqueID"/>
            <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#GUID"/>
            <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#URN"/>
            <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#GenericString"/>
            <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#SAMLSubject"/>
            <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#OID"/>
        </xsd:restriction>
    </xsd:simpleType>
```

Using one of these identifiers the RA issues an SPML operational request directed at a specific PSTD-ID. In the example below, the RA is requesting the deletion of a mail box identified by the <*Identifier*> of Jane.Doe@acme.com. Here the ***urn:oasis:names:tc:SPML:1:0#EMailAddress*** type is used as the target identifier type:

```
<deleteRequest>
    <identifier type= "urn:oasis:names:tc:SPML:1:0#EMailAddress">
        <id>Jane.Doe@acme.com</id>
    </identifier>
</deleteRequest>
```

More complex identifiers can be specified using the other <*IdentifierTypes*>. For example, if an RA wishes to subscribe to a published SPML service and wishes to use a SAML subject statement as it's identification for the service object.

3.4.2 Provider & Operation Identifiers

SPML <*ExtendedRequest*> operations and the service schema definition model both implement a <*providerIdentifier*> element. The <*providerIdentifier*> element allows a PSP to annotate new operations and published services such that a set of extended requests or service schema elements can be correlated back to a specific provider. The <*providerIdentifier*> element is used in conjunction with <*operationIdentifier*> element to control the globally-unique identification of extended requests. The <*providerIdentifier*> element is used in conjunction with the <*schemaIdentifier*> element to control the globally-unique identification of service schema definitions.

For information about the <*schemaIdentifier*> elements see section 3.6.2 in this document.

3.5 Request / Response Model Overview

The general model adopted by this protocol is one of clients (RA's) performing protocol operations against servers (PSPs). In this model, a client issues an SPML request describing the operation to be performed at a given service point. SPML requests can be issued as singleton requests or as multi-request batches, both of which may be executed either synchronously or asynchronously based on the client and/or servers requirements and capabilities. The following sections describe the use of batch and non-batch operation and the use and semantics of the differing operational models supported by this protocol.

3.5.1 Execution Types

SPML requests support two distinct execution types **synchronous** and **asynchronous**. When an RA constructs a request it uses the "**execution**" attribute to instruct the PSP of the execution model for the singleton or batch operations. Synchronously processed requests are executed by the server in a blocking synchronous fashion in which the RA holds open its "execution" call and awaits its completion. In the asynchronous model, the RA issues its request with a globally unique "**requestID**" attribute and does not exclusively wait for the return of the corresponding batch result document. The <*requestID*> is maintained by the PSP throughout the execution of the batch and is returned to the client in the batch response document. The <*requestID*> is then exclusively used to query, control and manage pending and executing SPML batches.

3.5.2 Request Status

When SPML operations are being executed asynchronously, the RA needs a way to query the status of requests. SPML V1.0 provides the <*StatusRequest*> operation to allow clients to request processing status from the corresponding service. The example below shows a RA request status and corresponding server response. Note the status response includes an explicit typed "**result**" attribute.

```
<statusRequest requestID="A4DF567HGD"/>
```

```
<statusResponse requestID="A4DF567HGD"
        Result="urn:oasis:names:tc:SPML:1:0#pending"/>
```

SPML V1.0 provides a fixed set of result codes for <*StatusResponse*> elements; these are defined by the <*StatusReturnsType*> element shown below:

```
    <xsd:simpleType name="StatusReturnsType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#status "/>
            <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#result"/>
        </xsd:restriction>
    </xsd:simpleType>
```

In certain circumstances it is desirable for an RA to be able to query not only the status of an asynchronous request but to also get the current request results set. SPML V1.0 provides this capability via the "**statusReturns"** attribute on the <*StatusRequest*> element. In the following example, a RA has issued an asynchronous <*SearchRequest*>. The same RA then issues an <*StatusRequest*> with "**statusReturns=urn:oasis:names:tc:SPML:1:0#result**"*.* The resulting <*StatusResponse*> element is then returned with the currently processed search results.

```
<statusRequest requestID="B98Y76T" statusReturns="result">
```

```
<statusResponse requestID=" B98Y76T " Result="urn:oasis:names:tc:SPML:1:0#pending">
    < searchResultEntry>
        < identifier type= "urn:oasis:names:tc:SPML:1:0#EMailAddress">
```

```
            <identifier >Jane.Doe@acme.com</id>
        </identifier>
        <attributes>
            <attr name="email"><value>Jane.Doe@acme.com</value></attr>
        <attr name="cn"><value>Jane Doe</value></attr>
    </attributes>
    </searchResultEntry>
        <searchResultEntry>
        <identifier type= "urn:oasis:names:tc:SPML:1:0#EMailAddress">
            <id >John.Doe@acme.com</id>
        </identifier>
        <attributes>
            <attr name="email"><value>John.Doe@acme.com</value></attr>
            <attr name="cn"><value>John Doe</value></attr>
        </attributes>
    </searchResultEntry>
</statusResponse>
```

3.5.3 Cancel Request

When SPML operations are being executed asynchronously, the RA also needs a way to cancel a pending request. SPML V1.0 provides the <*CancelRequest*> operation to allow clients to request the cancellation of a requested batch from the corresponding service. The example below shows an RA requesting the cancellation of a previously requested operation, followed by the resulting response from the PSP. Note the cancellation response includes an explicit typed "**CancelResults**" attribute.

```
<cancelRequest requestID="A4DF567HGD"/>
```

```
<cancelResponse requestID="A4DF567HGD"
          cancelResults="urn:oasis:names:tc:SPML:1:0#canceled "/>
```

SPML V1.0 provides a fixed set of result codes for <*CancelResponse*> elements; these are defined by the <*CancelResultsType*> element shown below:

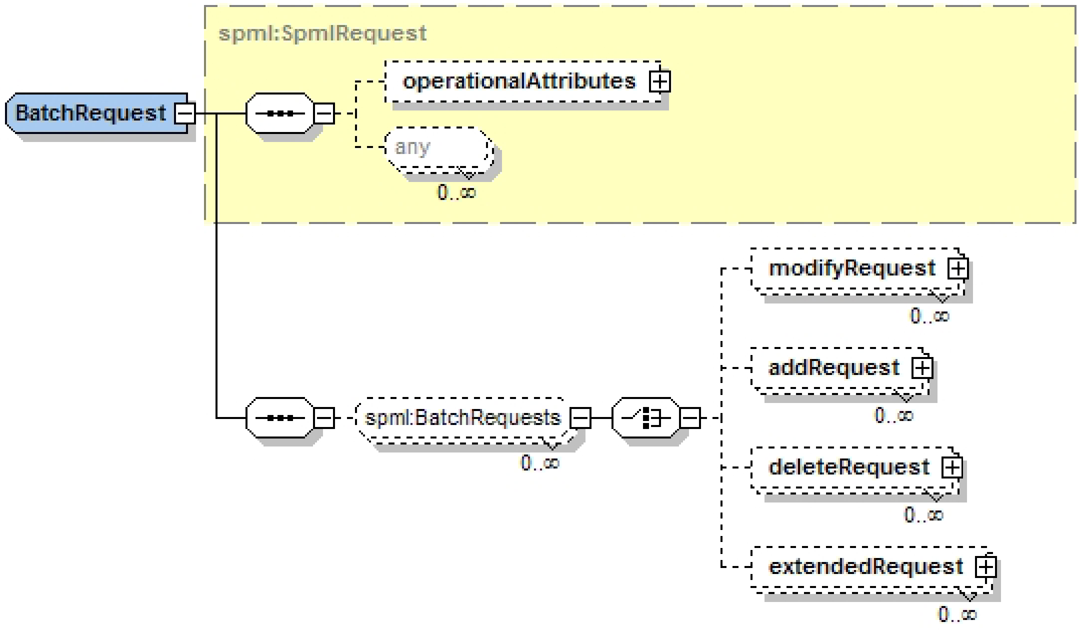
```
<xsd:simpleType name="CancelResultsType">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#noSuchRequest "/>
      <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#canceled"/>
      <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#couldNotCancel"/>
   </xsd:restriction>
</xsd:simpleType>
```

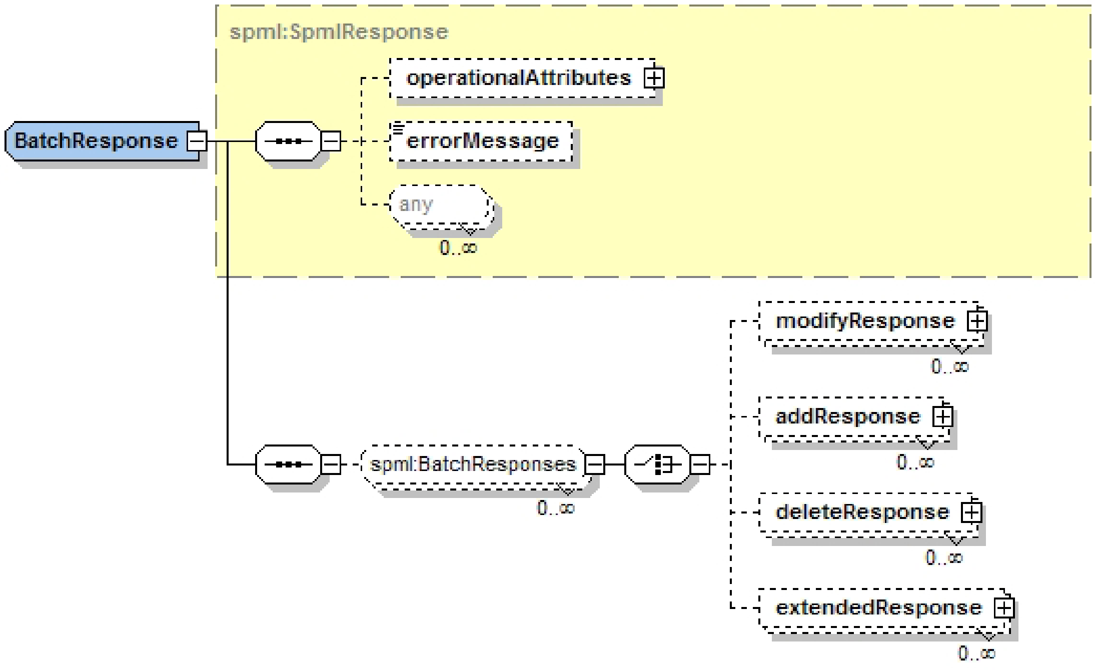
3.5.4 Singleton Requests

Each of the SPML core operations implicitly ties together a request and a response. Individual requests can be constructed by an RA and bound to the chosen protocol as defined in the SPML bindings document [**SPML-Bind**] and dispatched to a given PSP. A singleton request follows the basic "**execution"** model defined in section 7.5.1 of this document. The default execution mode for all singleton requests is synchronous. The following example shows a basic singleton <*AddRequest*>:

```
<addRequest requestID="Request-92.168.1.1-992A1">
    <attributes>
        <attr name="objectclass"> <value>emailUser</value></attr>
        <attr name="cn"> <value>Jane Doe</value></attr>
        <attr name="gn"><value>Jane</value></attr>
        <attr name="sn"><value>Doe</value></attr>
    </attributes>
</addRequest>
```

3.5.5 Batch Requests





SPML V1.0 provides a comprehensive batch request/response model in which multiple SPML operations can be collected together and issued as a single <*BatchRequest*>. The RA and PSP associate individual responses in an <*BatchResponse*> with the corresponding individual request in a <*BatchRequest*> based on a system of positional correspondence.

Positional correspondence refers to the ordering of response elements from the PSP such that the *nth* response element corresponds to the *nth* request element. For example, if the third SPML operation in a batch is a <*DeleteRequest*>*,* the third request element in the batch response is guaranteed to be the corresponding <*DeleteResponse*>*.*

Using positional correspondence, a valid batch request-response pair can never have fewer responses in the response document than requests in the request document. If an error occurs and the batch *"***onError***"* is *"**resume**"*, positional correspondence ensures that the failing nth request element would correspond to the nth response element with a relevant error code, and the rest of the batch elements would be executed and returned in the same order.

The <*BatchRequest*> and <*BatchResponse*> elements are extensions of <*SpmlRequest*> and <*SpmlResponse*> elements. They therefore both provide access to <*operationalAttributes*>. RAs may use <*operationalAttributes*> as defined in section 3.3.4 of this document to provide additional batch control parameters.

3.5.5.1 Processing Types

Batch requests support two distinct processing types ***sequential*** and ***parallel***. When an RA constructs a batch request it uses the "**processing**" attribute to instruct the PSP of the processing order of the individual operation in that batch. Sequentially processed batch operations are executed by the PSP in the exact order they are defined in the <*B*atchRequest>. Parallel batch operations may be executed by the PSP in any order. In both cases, the <*BatchResponse*> maintains positional correspondence in the return of individual operation response elements. In the below example the RA request the asynchronous execution of parallel batch of two add requests. Note the mandatory use of the **requestID** attribute on asynchronous batches.

```
<batchRequest xmlns:spml="urn:oasis:names:tc:SPML:1:0" requestID="A4DF567HGD"
                          processing ="parallel" execution="asynchronous">
    <addRequest>
        <attributes>
            <attr name="objectclass"> <value>emailUser</value></attr>
            <attr name="cn"> <value>Jane Doe</value></attr>
            <attr name="gn"><value>Jane</value></attr>
            <attr name="sn"><value>Doe</value></attr>
        </attributes>
    </addRequest>
    <addRequest>
        <attributes>
            <attr name="objectclass"> <value>emailUser</value></attr>
            <attr name="cn"> <value>John Doe</value></attr>
            <attr name="gn"><value>John</value></attr>
            <attr name="sn"><value>Doe</value></attr>
        </attributes>
    </addRequest>
</batchRequest>
```

3.5.5.2 Batch Errors

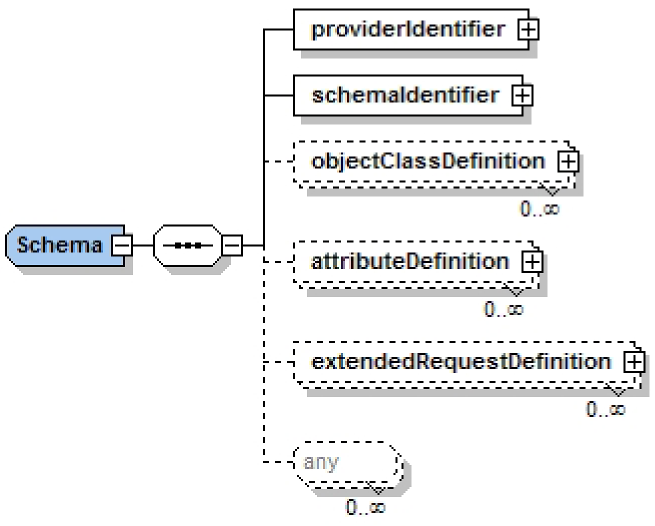
SPML V1.0 batches support two basic error handling modes "***resume***" and "***exit***". The <BatchRequest> base element provides an ***onError*** attribute to control this behavior. When an RA issues a batch request with "***onError*=resume**" errors encountered processing individual operations within that batch are handles by the PSP and do not effect the attempted execution of the remaining operations in the batch. It is the responsibility of the PSP to maintain the positional correspondence of the individual operations and provide appropriate error reporting as described in section 3.3.6.

When an RA issues a batch request with ***onError="exit",*** an error encountered processing an individual operation within that batch results in the termination of processing for the entire batch and all of the requests that did not get processes are marked as failed. When used with the **processing** attribute, **onError** provides the RA with the ability to guarantee execution order and pre-requisite processing in batch operations.

When an RA issues a <*BatchRequest*> with **onError="continue"** and some of the request in that batch succeed and some fail, the PSP will return a <*BatchResponse*> element with the **result='failure'** even when some of the requests in that batch may have completed successfully. It is the responsibility of the RA to parse all response elements in the batch to assess exactly which requests succeeded and what failed.

The SPML V1.0 batch request and response elements extend the base <*SpmlRequest*> and <*SpmlResponse*> elements and as such reuse the same <*Result*>, **error** and <*errorMessage*> models described for singleton request/responses in section 3.3.4.

3.6 SPML Schema Overview



At the center of the SPML V1.0 protocol model is the definition of provisioning schema. Provisioning schema represents a standardized way of defining the attributes that constitute the definition and description of a given service. SPML uses the XML Schema **[XS]** based attribute typing model and adds to this an object class definition and attribute sharing model similar to those parts of the X.500 object model.

Each of the SPML V1.0 core operational requests represent an action against a given service schema. In each operation the service schema is defined by the special attribute "**objectclass**" as shown below. The **objectclass** attribute is a direct reference to a known SPML V1.0 <*objectclassDefinition*> element. In the example below, an <*AddRequest*> operation is requesting the creation of an object of the pre defined object class "**urn:oasis:names:tc:SPML:standard**".

```
< addRequest>
    <attributes>
        <attr name="objectclass">
          <value>urn:oasis:names:tc:SPML:standard:person</value>
        </attr>
        .........
    </attributes>
</addRequest>
```

The SPML service schema model provides the <*schema*> element to contain the definition of the service schema and a simple request/response protocol for exchanging schema definitions between SPML actors.

3.6.1 The Schema Element

The SPML V1.0 service schema is represented by the <*schema*> element. This element is used to define the reusable definition of an SPML service as a uniquely identifiable set of XML Schema typed data attributes and object class definitions with corresponding attributes and properties. The following example shows an SPML schema definition that defines a simple service schema object class called "**standard**".

```
<schema majorVersion="1" minorVersion="0">

    <providerIdentifier providerIDType="urn:oasis:names:tc:SPML:1:0#URN">
        <providerID>urn:oasis:names:tc:SPML</providerID>
    </providerIdentifier>

    <schemaIdentifier schemaIDType="urn:oasis:names:tc:SPML:1:0#GenericString">
        <schemaID>standard</schemaID>
    </schemaIdentifier>

    <attributeDefinition name="cn" description="Full name, or common name."/>
    <attributeDefinition name="email" description="E-mail address."/>
    <attributeDefinition name="description" description="Description."/>

    <objectclassDefinition name="person" description="Sample standard person.">
        <memberAttributes>
            <attributeDefinitionReference name="cn" required="true"/>
            <attributeDefinitionReference name="email" required="true"/>
            <attributeDefinitionReference name="description"/>
        </memberAttributes>
    </objectclassDefinition>

</schema>
```

The “**standard**” schema defined above is uniquely identified by the combination of the globally unique “**urn:oasis:names:tc:SPML:1:0#URN**” <*providerIdentifier*> element and a locally unique “**urn:oasis:names:tc:SPML:1:0#GenericString**” <*schemaIdentifier*> element. This schema then defines three attributes that may be used in locally defined object class definitions or reference from external schema. The <*objectclassDefinition*> brings all this together in a single definition of the services schema named “**person**”. The person service object class includes a text description attribute and in this simple example, references the attributes defined in the local schema element itself.

3.6.2 Schema Qualification

One of the requirements of SPML is to provide a model for the definition and exchange of service schema and in doing so to enable the definition and re-use of both entire schema definitions AND individual schema attributes. Vital to this goal is the judicious and complicit use of scoped and qualified names for defined object class definitions and contained attributes. In the SPML V1.0 model, attribute namespace is controlled by the use of a URN based schema identification model that reuses the core SPML <*providerIdentifier*> and an additional <*schemaIdentifier*> element.

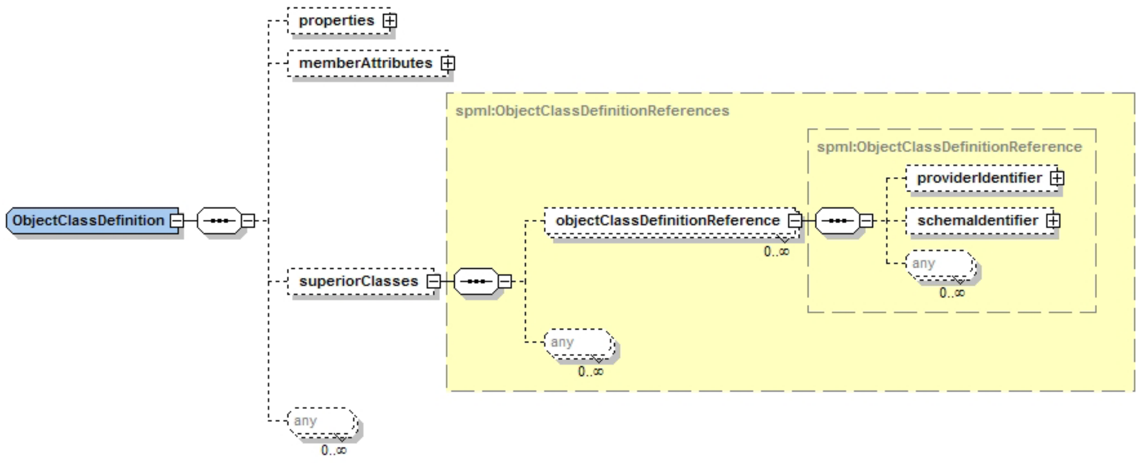
The <*providerIdentifier*> element is described in section 3.4.2 of this document. The <*schemaIdentifier*> is used in conjunction with the <*providerIdentifier*> element to provide a unique name for a given service schema. The following <*schemaIdentifier*> types are supported by SPML V1.0.

```
    <xsd:complexType name="SchemaIdentifier">
        <xsd:sequence>
            <xsd:element name="schemaID" type="xsd:anyType"/>
        </xsd:sequence>
        <xsd:attribute name="schemaIDType" use="required">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#URN"/>
                    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#OID"/>
                    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#GenericString"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
    </xsd:complexType>
```

The following example the ***GenericString*** <*schemaIdentifier*> of "**GoldEmailService**" and the unique **OID** of "**ID-1.3.6.1.4.868.2.4.1.2.1.1.1.3.3562**" are used to create a globally unique identifier for a service schema.

```
<schema …>
    ……..
    <providerIdentifier providerIDType="urn:oasis:names:tc:SPML:1:0#OID">
        <providerID>ID-1.3.6.1.4.868.2.4.1.2.1.1.1.3.3562</providerID>
    </providerIdentifier>

    <schemaIdentifier schemaIDType="urn:oasis:names:tc:SPML:1:0#GenericString">
        <schemaID>GoldEmailService</schemaID>
    </schemaIdentifier>
    ……….
</schema>
```

3.6.3 Service Object Class Definition



The <*objectclassDefinition*> element is used to define, name and describe a specific service object. Each object class optionally contains a <*memberAttributes*> container element that defines the specific attributes of that class definition. NOTE all attribute definitions in the <*memberAttributes*> element are <*AttributeDefinitionReferences*>*.* All attributes are included in a given <*objectclassDefinition*> by one of three means. One, they are referenced <*attributeDefinition*> elements from the enclosing <*schema*> definition in which case they have a simple string name. Two, they are referenced as <*attributeDefinition*> elements from another SPML V1.0 schema in which case they have qualified URN names. Three, they are "inherited" as member attributes of the <*superiorClasses*> definition for the new class definition.

The <*objectclassDefinition*> <*superiorClasses*> sub element implements an attribute inheritance model for SPML V1.0 service schema. By defining a <*superiorClasses*> element a given schema can automatically inherit the set of <*objectclassDefinition*> elements from another SPML schema. In the first example below the SPML service schema defines a "**standard**" service schema which is then referenced in the second example to create the new SPML service schema “**interopUser**”.

```
<schema majorVersion="1" minorVersion="0">
    <providerIdentifier providerIDType="urn:oasis:names:tc:SPML:1:0#URN">
        <providerID>urn:oasis:names:tc:SPML</providerID>
    </providerIdentifier>
    <schemaIdentifier schemaIDType="urn:oasis:names:tc:SPML:1:0#GenericString">
        <schemaID>standard</schemaID>
    </schemaIdentifier>
    <attributeDefinition name="cn" description="Full name, or common name."/>
    <attributeDefinition name="email" description="E-mail address."/>
    <attributeDefinition name="description" description="Description."/>
    <objectclassDefinition name="person" description="Standard person.">
        <memberAttributes>
            <attributeDefinitionReference name="cn" required="true"/>
            <attributeDefinitionReference name="email" required="true"/>
            <attributeDefinitionReference name="description"/>
        </memberAttributes>
    </objectclassDefinition>
</schema>
```

```
<schema majorVersion="1" minorVersion="0">
    <providerIdentifier providerIDType="urn:oasis:names:tc:SPML:1:0#URN">
        <providerID>urn:oasis:names:tc:SPML</providerID>
    </providerIdentifier>
    <schemaIdentifier schemaIDType="urn:oasis:names:tc:SPML:1:0#GenericString">
        <schemaID>interop</schemaID>
    </schemaIdentifier>
    <attributeDefinition name="memberLevel"/>
    <attributeDefinition name="company"/>
    <attributeDefinition name="registrationTime"/>
    <objectclassDefinition name="interopUser" description="Interoperability demo user.">
        <superiorClasses>
            <objectclassDefinitionReference name="urn:oasis:names:tc:SPML:standard:person"/>
        </superiorClasses>
        <memberAttributes>
            <attributeDefinitionReference attributeName="memberLevel"/>
            <attributeDefinitionReference attributeName="company"/>
            <attributeDefinitionReference attributeName="registrationTime"/>
        </memberAttributes>
    </objectclassDefinition>
</schema>
```

3.6.4 Attributes, Typing & Referencing

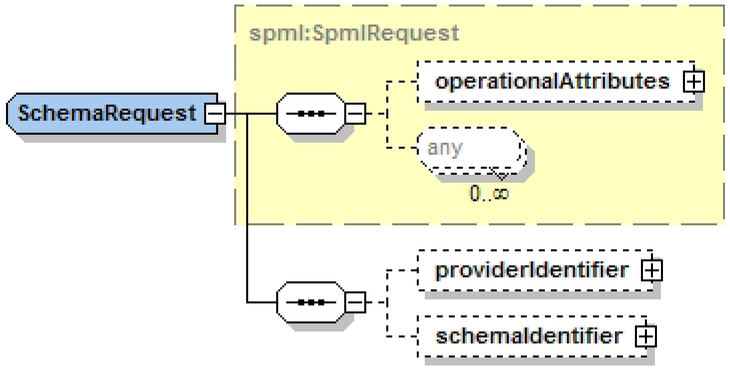
The SPML V1.0 schema model supports the definition of service schema attributes as XML Schema type definitions. NOTE the base <*AttributeDefinition*> element definition shown below to be a default type of “**xsd:string**” and provides support for multi-valued attributes with a default of “**false**”.

```
    <xsd:complexType name="AttributeDefinition">
        <xsd:sequence>
            <xsd:element name="properties" type="spml:Properties" minOccurs="0"/>
            <xsd:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="name" type="dsml:AttributeDescriptionValue" use="required"/>
        <xsd:attribute name="description" type="xsd:string" use="optional"/>
        <xsd:attribute name="multivalued" type="xsd:boolean" use="optional" default="false"/>
        <xsd:attribute name="type" type="xsd:string" use="optional" default="xsd:string"/>
        <xsd:anyAttribute namespace="##other" processContents="lax"/>
    </xsd:complexType>
```

3.7 Schema Operations

Schema operations provide a simple request response model that allows for the exchange of provisioning schema between SPML V1.0 clients and servers. The <*SchemaRequest*> and <*SchemaResponse*> elements are derived from the base <*SpmlRequest*> and <*SpmlResponse*> elements and as such provide the same operational models described in section 3.5 of this document.

3.7.1 Schema Request

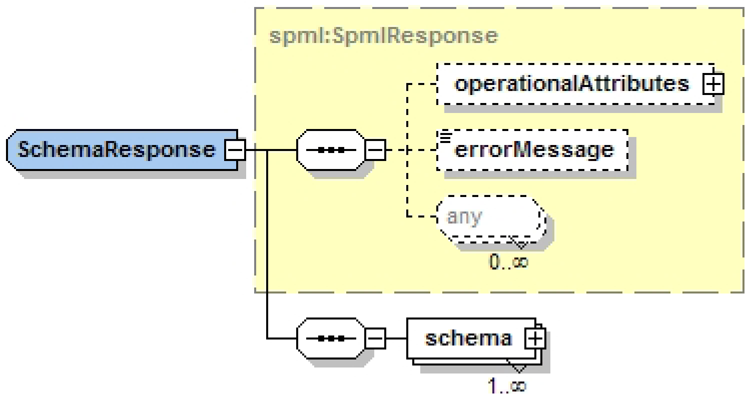


A <*SchemaRequest*> is used to request the retrieval of a specific provisioning schema. The specififc schema for retrieval is identified using the <*providerIdentifier*> and <*schemaIdentifier*> elements. Note if no <*schemaIdentifier*> element is provided the PSP will return all schema definitions visable by the requesting party.

In the following example an RA requests the retrieval of the provisioning schema defined in section 3.6.3 above **urn:oasis:names:tc:SPML:interop.**

```
<schemaRequest requestID="REQ1.4.5.6.AKS"
                    execution="urn:oasis:names:tc:SPML:1:0#synchronous">

    <providerIdentifier providerIDType="urn:oasis:names:tc:SPML:1:0#URN">
        <providerID> urn:oasis:names:tc:SPML</providerID>
    </providerIdentifier>
    <schemaIdentifier schemaIDType="urn:oasis:names:tc:SPML:1:0#GenericString">
        <schemaID>interop</schemaID>
    </schemaIdentifier>
</schemaRequest>
```

3.7.2 Schema Response



The <*SchemResponse*> element is used to provide the results of an <*SchemaRequest*>. The <*SchemaRequest*> shown in section 3.7.1 above would simply result in the following elements:

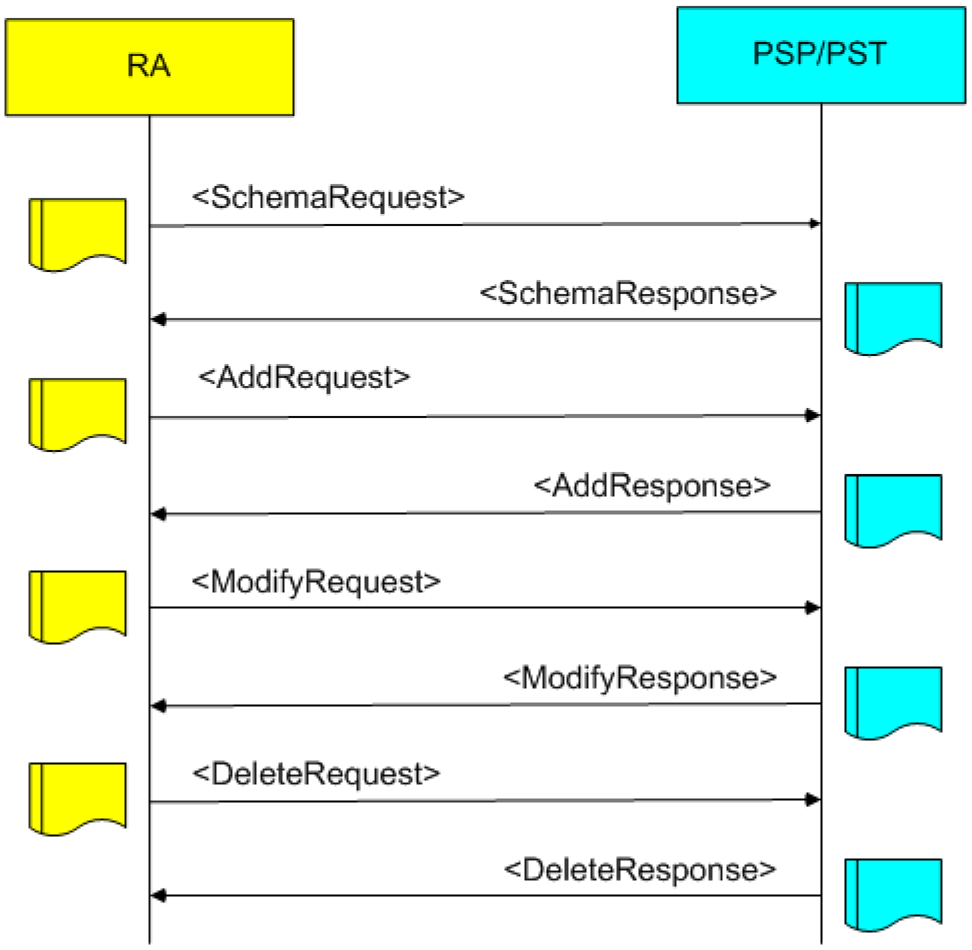
```
<schemaResponse result = "urn:oasis:names:tc:SPML:1:0#success">
        <Schema ....

        < providerIdentifier providerIDType="urn:oasis:names:tc:SPML:1:0#URN">
            <providerID>urn:oasis:names:tc:SPML</providerID>
        </ providerIdentifier>
        <schemaIdentifier schemaIDType="urn:oasis:names:tc:SPML:1:0#GenericString">
            <schemaID>interop</schemaID>
        </schemaIdentifier>
        <attributeDefinition name="memberLevel"/>
        <attributeDefinition name="company"/>
        <attributeDefinition name="registrationTime"/>
        <objectclassDefinition name="interopUser" description="Interoperability demo user.">
            <superiorClasses>
                <objectclassDefinitionReference
name="urn:oasis:names:tc:SPML:standard:person"/>
            </superiorClasses>
            <memberAttributes>
                <attributeDefinitionReference attributeName="memberLevel"/>
                <attributeDefinitionReference attributeName="company"/>
                <attributeDefinitionReference attributeName="registrationTime"/>
            </memberAttributes>
        </objectclassDefinition>
    </schema>
</schemaResponse>
```

4 Examples (non-normative)

The following non-normative section provides two examples that build of the SPML use cases [**PSTC-UC**]. Example 1 describes a simple synchronous flow in which the Requesting Authority asks for the creation of a new service object. Example 2 describes a more complex asynchronous request for a batch operation with subsequent status query.

4.1 Example One

The following schema fragments follow an operational flow between an RA and PSP for the creation, modification and subsequent deletion of specific service instance. It shows a simple synchronous non-batch exchange. This example does not include the specifics of any given SPML binding. The following outlines the request and response flow:



4.1.1 <SchemaRequest>

First the RA makes a synchronous request for the return of a given sevice schema. From the returned schema definition the RA will build an <*AddRequest*>. Note the RA is looking for the interop schema from the **urn:oasis:names:tc:SPML** provder:

```
<schemaRequest execution="urn:oasis:names:tc:SPML:1:0#synchronous">
    <providerIdentifier providerIDType="urn:oasis:names:tc:SPML:1:0#URN">
        <providerID> urn:oasis:names:tc:SPML</providerID>
    </providerIdentifier>
    <schemaIdentifier schemaIDType="urn:oasis:names:tc:SPML:1:0#GenericString">
```

```
        <schemaID>interop</schemaID>
    </schemaIdentifier>
</schemaRequest>
```

4.1.2 <SchemaResponse>

The PSP successfully returns the requested schema:

```
<schemaResponse result = "urn:oasis:names:tc:SPML:1:0#success">
        <schema ....

        <providerIdentifier providerIDType="urn:oasis:names:tc:SPML:1:0#URN">
            <providerID>urn:oasis:names:tc:SPML</providerID>
        </providerIdentifier>
        <schemaIdentifier schemaIDType="urn:oasis:names:tc:SPML:1:0#GenericString">
            <schemaID>interop</schemaID>
        </schemaIdentifier>
        <attributeDefinition name="memberLevel"/>
        <attributeDefinition name="company"/>
        <attributeDefinition name="registrationTime"/>
        <objectclassDefinition name="interopUser" description="Interoperability demo user.">
            <superiorClasses>
                <objectclassDefinitionReference
name="urn:oasis:names:tc:SPML:standard:person"/>
            </superiorClasses>
            <memberAttributes>
                <attributeDefinitionReference attributeName="memberLevel"/>
                <attributeDefinitionReference attributeName="company"/>
                <attributeDefinitionReference attributeName="registrationTime"/>
            </memberAttributes>
        </objectclassDefinition>
    </schema>
</schemaResponse>
```

4.1.3 <AddRequest>

Based on the schema returned the RA then issues a synchronous <*AddRequest*> including the attributes required to subscribe to this service:

```
<addRequest>
    <attributes>
        <attr name="objectclass">
          <value>urn:oasis:names:tc:SPML:interop:interopUser</value>
        </attr>
        <attr name="cn">
            <value>Jane Doe</value>
        </attr>
        <attr name="mail">
            <value>jdoe@acme.com</value>
        </attr>
        <attr name="description">
            <value>Jane Doe interopUser Subscription</value>
        </attr>
        <attr name="memberLevel">
            <value>1</value>
        </attr>
        <attr name="company">
            <value>Jane Doe Supply Co</value>
        </attr>
        <attr name="registrationTime">
            <value>17-Nov-2002 12:00 </value>
        </attr>
    </attributes>
</addRequest>
```

4.1.4 <AddResponse>

The PSP returns an <*AddResponse*> and includes an additional <*attributes*> elelement that includes informational data on the fulfillment of the request:

```
<addResponse result = “urn:oasis:names:tc:SPML:1:0#success">
    <identifier type= "urn:oasis:names:tc:SPML:1:0#EMailAddress">
        <id>Jane.Doe@acme.com</id>
    </identifier>
    <attributes>
        <attr name="mailBoxLimit">
            <value>50MB</value>
        </attr>
    </attributes>
</addResponse>
```

4.1.5 <ModifyRequest>

Using <identifier> elelemt to uniquely identify the PSTD the RA issues a <*ModifyRequest*> to change the subscribers **memberLevel** attribute to value “**2**”:

```
<modifyRequest>
    <identifier type= "urn:oasis:names:tc:SPML:1:0#EMailAddress">
        <id>Jane.Doe@acme.com</id>
    </identifier>
    <modifications>
        <modification name="memberLevel">
            <value>2</value>
        </modification>
    </modifications>
</modifyRequest>
```

4.1.6 <ModifyResponse>

The PSP responds with a <*ModifyResponse*> with the **result** attribte set to **urn:oasis:names:tc:SPML:1:0#success** indicating that the request was successfully executed:

```
<modifyResponse result = "urn:oasis:names:tc:SPML:1:0#success" />
```

4.1.7 <DeleteRequest>

Finally the RA issues a <*DeleteRequest*> to remove the PSTD-ID from the service:

```
<deleteRequest>
    <identifier type= "urn:oasis:names:tc:SPML:1:0#EMailAddress">
        <id>Jane.Doe@acme.com</id>
    </identifier>
</deleteRequest>
```

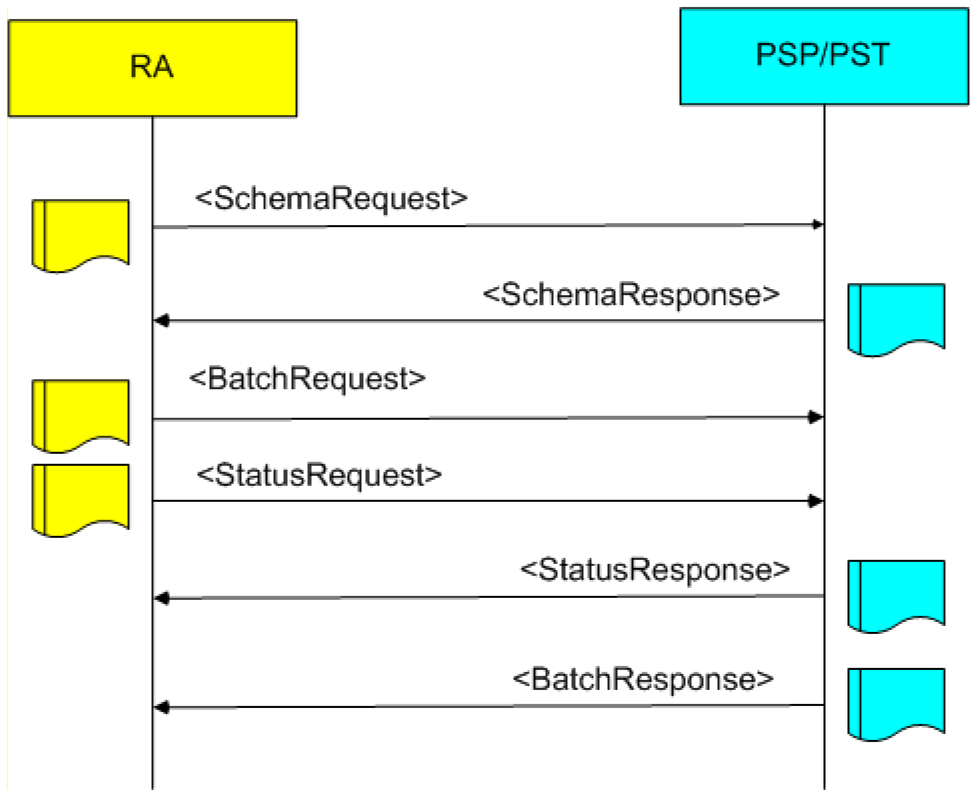
4.1.8 <DeleteResponse>

The PSP responds with a <*DeleteResponse*> with a **result** attribute of **urn:oasis:names:tc:SPML:1:0#success** to indicate that the specified PSTD was removed:

```
<deleteResponse result = "urn:oasis:names:tc:SPML:1:0#success" />
```

4.2 Example Two

The following schema fragments follow an operational flow between an RA and PSP for the batch creation of service instances. The batch is requested to be executed asynchronously with the onError=continue schematics. NOTE the RA follows the batch request with a synchronous status query for status of the pending batch. Again this example does not include the specifics of any given SPML binding. The following outlines the request and response flow:



4.2.1 <SchemaRequest>

First the RA makes a synchronous request for the return of a given sevice schema. From the returned schema definition the RA will build an <*AddRequest*>. Note the RA is looking for the interop schema from the **urn:oasis:names:tc:SPML** provder:

```
<schemaRequest = requestID="REQ1.4.5.6.AKS"
                         execution="urn:oasis:names:tc:SPML:1:0#synchronous">

    <providerIdentifier providerIDType="urn:oasis:names:tc:SPML:1:0#URN">
        <providerID> urn:oasis:names:tc:SPML</providerID>
    </providerIdentifier>
    <schemaIdentifier schemaIDType="urn:oasis:names:tc:SPML:1:0#GenericString">
        <schemaID>interop</schemaID>
    </schemaIdentifier>
</schemaRequest>
```

4.2.2 <SchemaResponse>

The PSP successfully returns the requested schema:

```
<schemaResponse result = "urn:oasis:names:tc:SPML:1:0#success"
                            requestID = "REQ.4.5.6.AKS" >
        <schema ....
        <providerIdentifier providerIDType="urn:oasis:names:tc:SPML:1:0#URN">
            <providerID>urn:oasis:names:tc:SPML</providerID>
        </providerIdentifier>
        <schemaIdentifier schemaIDType="urn:oasis:names:tc:SPML:1:0#GenericString">
            <schemaID>interop</schemaID>
        </schemaIdentifier>
        <attributeDefinition name="memberLevel"/>
        <attributeDefinition name="company"/>
        <attributeDefinition name="registrationTime"/>
        <objectclassDefinition name="interopUser" description="Interoperability demo user.">
            <superiorClasses>
                <objectclassDefinitionReference
name="urn:oasis:names:tc:SPML:standard:person"/>
            </superiorClasses>
            <memberAttributes>
                <attributeDefinitionReference attributeName="memberLevel"/>
                <attributeDefinitionReference attributeName="company"/>
                <attributeDefinitionReference attributeName="registrationTime"/>
            </memberAttributes>
        </objectclassDefinition>
    </schema>
</schemaResponse>
```

4.2.3 <BatchRequest>

Based on the schema definition returned, the RA requests an asynchronous batch containing two <*AddRequests*> elements both of which are given unique <*requested*> identifiers:

```
<batchRequest requestID="A4DF567HGD"
                  processing ="parallel" execution="asynchronous" onError ="resume" >
    <addRequest requestID="A4DF567HGD-1001”>
      <attributes>
          <attr name="objectclass">
              <value>urn:oasis:names:tc:SPML:interop:interopUser</value>
          </attr>
          <attr name="cn"><value>Jane Doe</value></attr>
          <attr name="mail"><value>jdoe@acme.com</value></attr>
          <attr name="description"><value>Jane Doe interopUser Subscription</value></attr>
          <attr name="memberLevel"><value>1</value></attr>
          <attr name="company"><value>Jane Doe Supply Co</value></attr>
          <attr name="registrationTime"><value>17-Nov-2002 12:00 </value></attr>
      </attributes>
    </addRequest>

    <addRequest requestID="A4DF567HGD-1002”>
      <attributes>
          <attr name="objectclass">
              <value>urn:oasis:names:tc:SPML:interop:interopUser</value>
          </attr>
          <attr name="cn"><value>John Doe</value></attr>
          <attr name="mail"><value>johndoe@acme.com</value></attr>
          <attr name="description"><value>John Doe another interopUser</value></attr>
          <attr name="memberLevel"><value>2</value></attr>
```

```
            <attr name="company"><value>Jane Doe Supply Co</value></attr>
            <attr name="registrationTime"><value>17-Nov-2002 12:01 </value></attr>
        </attributes>
    </addRequest>
</batchRequest>
```

4.2.4 <StatusRequest>

Having issued the asynchronous batch request, the RA then decides to query the PSP for the status of the request using the <*StatusRequest*> operation:

```
<statusRequest requestID="A4DF567HGD"/>
```

4.2.5 <StatusResponse>

In response to the <*StatusRequest*> the PSP returns the following <*StatusReposnse*>. Note the **Result** attribute for the batch is set to **urn:oasis:names:tc:SPML:1:0#pending** to indicate that at least one of the request within that batch has not yet completed.

```
<statusResponse requestID=" A4DF567HGD " Result="urn:oasis:names:tc:SPML:1:0#pending">
<addResponse requestID="A4DF567HGD-1001
                         result="urn:oasis:names:tc:SPML:1:0#success">
    <identifier type= "urn:oasis:names:tc:SPML:1:0#EMailAddress">
        <id>jdoe@acme.com</id>
    </identifier>
    <attributes>
        <attr name="mailBoxLimit">
            <value>50MB</value>
        </attr>
    </attributes>
</addResponse>
<addResponse requestID="A4DF567HGD-1002
                        result="urn:oasis:names:tc:SPML:1:0#pending">
</addResponse>
</statusResponse>
```

4.2.6 <BatchResponse>

Finally the PSP completes the batch and issues a <*BatchResponse*>. Note the batch **Result** attribute is now set to **urn:oasis:names:tc:SPML:1:0#failure** as the second request in the batch failed:

```
<batchResponse requestID=" A4DF567HGD " Result="urn:oasis:names:tc:SPML:1:0#failure">
<addResponse requestID="A4DF567HGD-1001
                  result="urn:oasis:names:tc:SPML:1:0#success">
    <identifier type= "urn:oasis:names:tc:SPML:1:0#EMailAddress">
        <id>jdoe@acme.com</id>
    </identifier>
    <attributes>
        <attr name="mailBoxLimit">
            <value>50MB</value>
        </attr>
    </attributes>
</addResponse>
<addResponse requestID="A4DF567HGD-1002
                   result="urn:oasis:names:tc:SPML:1:0#failure">
</addResponse>
</statusResponse>
```

5 SPML Operations (normative, with the exception of the schema fragments)

5.1 Schema Header and Namespace Declarations

The following schema fragment defines the XML namespaces and header information for the SPML schema.

```
<schema  targetNamespace="urn:oasis:names:tc:SPML:1:0"
            xmlns:spml="urn:oasis:names:tc:SPML:1:0"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:dsml="urn:oasis:names:tc:DSML:2:0"
            xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
            xmlns="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="qualified">

  <import namespace="urn:oasis:names:tc:DSML:2:0"
   schemaLocation="http://www.oasis-open.org/committees/dsml/docs/DSMLv2.xsd" />

  <import namespace="urn:oasis:names:tc:SAML:1.0:assertion"
   schemaLocation="http://www.oasis-open.org/committees/security/docs/cs-sstc-schema-
assertion-01.xsd"/>
...
</schema>
```

5.2 Complex Type Identifier

The Identifier type is used to specify either a PSO-ID or PSTD-ID the semantics of the identifier are determined from context. It is the type of both the <*identifier*> and <*searchBase*> elements. It MUST contain a "**type**" attribute, MUST contain either an <*id*> or <*subject*> element, and MAY contain any number of <*identifierAttributes*> elements.

type [Required]	The type of the identifier. It is of type IdentifierType
<id>	An element wrapping all identifiers except SAML Subjects. The contents of the element MUST be consistent with the value of the "**type**" attribute

	in the containing element. The element MUST contain either PCDATA or a single element, but not both.
<subject>	An element of type SubjectStatementAbstractType defined by SAML
< *identifierAttributes* >	An element of type DsmlAttr defined by DSML. These MAY be used to provide additional information about the identifier

The following schema fragment defines the Identifier complex type and the IdentifierType simple type:

```
<xsd:sequence>
  <xsd:choice>
    <xsd:element name="id" type="xsd:anyType" minOccurs="0"/>
    <xsd:element name="subject" type="saml:SubjectStatementAbstractType" minOccurs="0"/>
  </xsd:choice>
  <xsd:element name="attr" type="dsml:DsmlAttr" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:any minOccurs="0" maxOccurs="unbounded" processContents="lax" />
 </xsd:sequence>
 <xsd:attribute name="type" type="spml:IdentifierType" use="required"/>
 <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<xsd:simpleType name="IdentifierType">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#EMailAddress"/>
  <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#DN"/>
  <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#UserIDAndOrDomainName"/>
  <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#LibertyUniqueID"/>
  <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#PassportUniqueID"/>
  <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#GUID"/>
  <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#URN"/>
  <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#GenericString"/>
  <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#SAMLSubject"/>
  <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#OID"/>
 <xsd:restriction>
</xsd:simpleType>
```

5.3 Simple Type IdentifierType

The IdentifierType simple type defines the possible values for the "type" attribute of the Identifier complex type, and determines the semantics of the <*id*> element. If the <*id*> element contains text, IdentifierType defines how the text is to be interpreted. If the <*id*> element contains an element, the IdentiferType must be consistent with that element.

IdentifierType is an enumeration of the following values:

EMailAddress	An e-mail address.
DN	A distinguished name.
UserIDAndOrDomainName	A user id, possibly qualified with a domain name
LibertyUniqueID	A Liberty Alliance unique id

PassportUniqueID	A Microsoft Passport unique id
GUID	A unique account identifier such as those generated by Unix systems
URN	A vendor specific URN
GenericString	A generic text string
SAMLSubject	A SAML <Subject> element
OID	A ITU-T X.208 (ASN.1) Object Identifier

5.4 Element <Identifier>

The <*identifier*> element is of type "**Identifier**" and is used to specify a PSO-ID or PST-ID in a number of request and response elements. The following schema fragment is used to define the <*identifier*>.

```
<xsd:element name="identifier" type="spml:Identifier" minOccurs="0" maxOccurs="1"/>
```

5.5 Element <AddRequest>

The <*AddRequest*> is used to request the addition of an object of a service defined by the PSP. It is of type AddRequest which extends complex type SpmlRequest. It MAY contain an <identifier> element that uniquely identifies the service object [question: is "**service object**" commonly used for this?]. If an <*identifier*> is not specified, the PSP MUST return an <identifier> in the corresponding <*AddResponse*> if the object was created.

It MUST contain one <*attributes*> element that MAY contain any number of <*attr*> elements that specify the attributes of the service object to be added.

The attribute named "**objectclass**" MAY be used to specify the class of the object to be added. The names and semantics of all other attributes are defined by the PSP.

```
<xsd:element name="AddRequest" type="spml:AddRequest" />
<xsd:complexType name="AddRequest">
  <xsd:complexContent>
    <xsd:extension base="spml:SpmlRequest">
      <xsd:sequence>
        <xsd:element name="identifier" type="spml:Identifier" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="attributes" type="spml:Attributes" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Attributes">
  <xsd:sequence>
    <xsd:element name="attr" type="dsml:DsmlAttr" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:any minOccurs="0" maxOccurs="unbounded" processContents="lax" />
  </xsd:sequence>
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>
```

5.6 Element <AddResponse>

The <*AddResponse*> element is used to convey the results of a request specified with an <*AddRequest*> element. It is of type AddResponse which extends complex type SpmlResponse.

If the corresponding <*AddRequest*> element did not include an <*identifier*> element, and the requested service object was added without error, the <*addResoponse*> element MUST include an <identifier> element that identifies the new service object.

The element MAY include one <attributes> element which MAY contain any number of <attr> elements. These may be used to convey attributes of the service object that were not specified in the <*AddRequest*> or which differ from those specified in the <*AddRequest*>. The PSP is not required to return these attributes.

The following schema fragment defines the <*AddResponse*> element and its AddResponse complex type:

```
<xsd:element name="AddResponse" type="spml:AddResponse" />
<xsd:complexType name="AddResponse">
  <xsd:complexContent>
    <xsd:extension base="spml:SpmlResponse">
      <xsd:sequence>
        <xsd:element name="identifier" type="spml:Identifier" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="attributes" type="spml:Attributes" minOccurs="0" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

5.7 Element <ModifyRequest>

The <*ModifyRequest*> element is used to request the modification of an existing service object. It is of type ModifyRequest which extends type SpmlRequest. This element MUST contain an <*identifier*> element which uniquely identifies the service object to be modified. It MUST contain an <*modifications*> element that MAY contain any number of <*modification*> elements that define the modifications to be performed.

The <*modification*> element is of type dsml:Modification which is defined by DSML **[DSML]**.

The following schema fragment defines the <*ModifyRequest*> element and the ModifiRequest complex type:

```
<xsd:element name="ModifyRequest" type="spml:ModifyRequest" />
<xsd:complexType name="ModifyRequest">
  <xsd:complexContent>
    <xsd:extension base="spml:SpmlRequest">
      <xsd:sequence>
        <xsd:element name="identifier" type="spml:Identifier" minOccurs="1" maxOccurs="1" />
        <xsd:element name="modifications" type="spml:Modifications"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

```
<xsd:complexType name="Modifications">
  <xsd:sequence>
    <xsd:element name="modification" type="dsml:DsmlModification" minOccurs="0"
maxOccurs="unbounded"/>
    <xsd:any minOccurs="0" maxOccurs="unbounded" processContents="lax" />
  </xsd:sequence>
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>
```

5.8 Element <ModifyResponse>

The <*ModifyResponse*> element is used to convey the results of a request specified with a <*ModifyRequest*> element. It is of type ModifyResponse which extends SpmlResponse.

It MAY contain a <*modifications*> element which in turn MAY contain any number of <*modification*> elements. The <*modification*> elements may be returned by the PSP to convey modifications that were made to the object as a side effect and not specified in the original <*ModifyRequest*>.

The following schema fragment defines the <*ModifyResponse*> element and its ModifyResponse complex type:

```
<xsd:element name="ModifyResponse" type="spml:ModifyResponse" />
<xsd:complexType name="ModifyResponse">
  <xsd:complexContent>
    <xsd:extension base="spml:SpmlResponse">
      <xsd:sequence>
        <xsd:element name="modifications" type="spml:Modifications" minOccurs="0"
maxOccurs="1"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

5.10 Element <DeleteRequest>

The <*DeleteRequest*> element is used to request the deletion of an existing service object. It is of type DeleteRequest which extends SpmlRequest.

The element MUST contain an <*identifier*> element which uniquely identifies the service object to be deleted.

The following schema fragment defines the <*DeleteRequest*> element and its DeleteRequest complex type:

```
<xsd:element name="DeleteRequest" type="spml:DeleteRequest" />
<xsd:complexType name="DeleteRequest">
  <xsd:complexContent>
    <xsd:extension base="spml:SpmlRequest">
      <xsd:sequence>
        <xsd:element name="identifier" type="spml:Identifier" minOccurs="1" maxOccurs="1" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

5.10 Element <DeleteRespose>

The <*DeleteResponse*> element is used to convey the results of a request specified with the <DeleteRequest> element. It is of type DeleteResponse which extends SpmlResponse. It does not specify any additional elements or attributes beyond those specified by SpmlResponse.

The following schema fragment defines the <*DeleteResponse*> element and its DeleteResponse complex type:

```
<xsd:element name="DeleteResponse" type="spml:DeleteResponse" />
<xsd:complexType name="DeleteResponse">
  <xsd:complexContent>
    <xsd:extension base="spml:SpmlResponse">
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

5.11 Element <SearchRequest>

The <*SearchRequest*> element is used to request the retrieval of attributes of existing service objects. The element is of type SearchRequest which extends type SpmlRequest.

The element MAY contain a <*searchBase*> element which the PSP may use to constrain the search. The <*searchBase*> element is of complex type Identifier. The semantics of the <*searchBase*> element are defined by the PSP.

The element MAY contain a <*filter*> element which is used to restrict the search to objects whose attributes adhere to specified criteria. The <*filter*> element is of type dsml:Filter defined by DSML.

The element MAY contain an <attributes> element which is used to specify which attributes of the objects matching the search are to be retrieved. The element is of type dsml:AttributeDescriptions defined by DSML [**DSML**]

The following schema fragment defines the <*SearchRequest*> element and its SearchRequest complex type:

```
<xsd:element name="SearchRequest" type="spml:SearchRequest" />
<xsd:complexType name="SearchRequest">
  <xsd:complexContent>
    <xsd:extension base="spml:SpmlRequest">
      <xsd:sequence>
        <xsd:element name="searchBase" type="spml:Identifier" minOccurs="0" maxOccurs="1" />
        <xsd:element name="filter" type="dsml:Filter" minOccurs="0" maxOccurs="1" />
      <xsd:element name="attributes" type="dsml:AttributeDescriptions" minOccurs="0"
maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

5.12 Element <SearchResponse>

The <*SearchResponse*> element is used to convey the results of a request specified with the <SearchRequest> element. It is of type SearchResponse which extends SpmlResponse.

The element MAY contain any number of <*SearchResultEntry*> elements which contain information about the service objects that matched the search criteria.

The following schema fragment defines the <*SearchResponse*> element and its related types:

```
<xsd:element name="SearchResponse" type="spml:SearchResponse" />
<xsd:complexType name="SearchResponse">
  <xsd:complexContent>
    <xsd:extension base="spml:SpmlResponse">
      <xsd:sequence>
        <xsd:element name="SearchResultEntry" type="spml:SearchResultEntry" minOccurs="0"
maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

5.13 Element <SearchResultEntry>

Element <*SearchResultEntry*> found within element <*SearchResponse*> are of type SearchResultEntry. This type extends dsml:DsmlMessage which is defined by DSML. Beyond the attributes and elements specified DsmlMessage, elements of this type MAY include one <*identifier*> element and MAY include one <*attributes*> element.

The <*identifier*> element is used to convey the identity of a service object that matched the search criteria.

The <*attributes*> element is used to convey the names and values of attributes of the matching service object.

The following schema fragment defines the SearchResultEntry type:

```
<xsd:complexType name="SearchResultEntry">
  <xsd:complexContent>
      <xsd:sequence>
        <xsd:element name="identifier" type="spml:Identifier" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="attributes" type="spml:Attributes" minOccurs="0"
maxOccurs="unbounded"/>
        <xsd:any minOccurs="0" maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:complexContent>
</xsd:complexType>
```

5.14 Element <ExtendedRequest>

The <*ExtendedRequest*> element is used to request services provided by a PSP that are not specifically defined by SPML. The semantics of an extended request are defined solely by the PSP. The element is of type ExtendedRequest which extends SpmlRequest.

The element MUST contain an <*operationIdentifier*> element that specifies the service being requested.

The element MUST contain a <*providerIdentifier*> element that further identifies the service. If the <*ExtendedRequestDefinition*> that defines an extended request specifies both an <*operationIdentifier*> and a <*providerIdentifier*> then the <*providerIdentifier*> element MUST be included in the <*ExtendedRequest*>.

The element MAY contain an <*identifier*> element that identifies a particular service object to be associated with the request.

The element MAY contain an <attributes> element that in turn may contain any number of <*attr*> elements. These are used to convey arbitrary information specific to the operation.

The following schema fragment defines the <*ExtendedRequest*> element and the ExtendedRequest type:

```
<xsd:element name="ExtendedRequest" type="spml:ExtendedRequest" />
<xsd:complexType name="ExtendedRequest">
  <xsd:complexContent>
    <xsd:extension base="spml:SpmlRequest">
      <xsd:sequence>
        <xsd:element name="providerIdentifier" type="spml:ProviderIdentifier" minOccurs="1"
maxOccurs="1" />
        <xsd:element name="operationIdentifier" type="spml:OperationIdentifier" minOccurs="1"
maxOccurs="1"/>
        <xsd:element name="identifier" type="spml:Identifier" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="attributes" type="spml:Attributes" minOccurs="0"
maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

5.15 Element <ExtendedResposne>

The <*ExtendedResponse* > element is used to convey the result of a request specified with the <*ExtendedRequest*> element. It is of type ExtendedResponse which extends SpmlResponse.

It MAY include one <*attributes*> element which may include any number of <attr> elements.

The following schema fragment defines the <*ExtendedResponse* > element and the ExtendedResponse type:

```
<xsd:element name="ExtendedResponse " type="spml:ExtendedResponse " />
<xsd:complexType name="ExtendedResponse ">
  <xsd:complexContent>
    <xsd:extension base="spml:SpmlResponse">
      <xsd:sequence>
        <xsd:element name="attributes" type="spml:Attributes" minOccurs="0"
```

```
maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

5.16 Element <providerIdentifier>

The providerIdentifier element allows an SPML service to annotate new operations and published services such that a set of extended requests or service schema elements can be correlated back to a specific provider. The element MUST contain a <*providerID*> element. The content of <*providerID*> is determined by "providerIDType" attribute.

The element MUST contain a "providerIDType" attribute which MUST be one of the following values:

URN	A vendor specific URN
OID	A ITU-T X.208 (ASN.1) Object Identifier

The following schema fragment defines the <*providerIdentifier*> element and ProviderIdentifier type:

```
<xsd:element name="providerIdentifier" type="spml:ProviderIdentifier" minOccurs="1"
maxOccurs="1" />
<xsd:complexType name="ProviderIdentifier">
  <xsd:sequence>
    <xsd:element name="providerID" type="xsd:anyType" />
    <xsd:any minOccurs="0" maxOccurs="unbounded" processContents="lax" />
  </xsd:sequence>
  <xsd:attribute name="providerIDType" use="required">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#URN"/>
        <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#OID"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>
```

5.17 Element <operationIdentifier>

The <*operationIdentifier*> element is used to identify a service that may be requested in an <*ExtendedRequest*>.

The element MUST contain an "operationIDType" attribute that specifies the type of the identifier. It MUST be one of:

URN	A vendor specific URN
OID	A ITU-T X.208 (ASN.1) Object Identifier
GenericString	A string whose structure is specific to the PSP.

The element MUST contain an <*operationID*> element whose content is determined by the value of the "operationIDType" attribute.

The following schema fragment defines the <*operationIdentifier*> element and OperationIdentifier type:

```
<xsd:element name="operationIdentifier" type="spml:OperationIdentifier" minOccurs="1"
maxOccurs="1"/>
<xsd:complexType name="OperationIdentifier">
 <xsd:sequence>
  <xsd:element name="operationID" type="xsd:anyType" />
  <xsd:any minOccurs="0" maxOccurs="unbounded" processContents="lax" />
 </xsd:sequence>
 <xsd:attribute name="operationIDType" use="required">
  <xsd:simpleType>
   <xsd:restriction base="xsd:string">
    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#URN"/>
    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#OID"/>
    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#GenericString"/>
   </xsd:restriction>
  </xsd:simpleType>
 </xsd:attribute>
 <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>
```

6 SPML Request / Response (normative, with the exception of the schema fragments)

6.1 Complex Type SpmlRequest

The complex type SpmlRequest specifies information common to all SPML requests. It includes the following attributes and elements:

execution	Specifies the desired execution mode for the request. It is of type ExecutionType. If not specified, execution mode defaults to "**synchronous**".
requestID	A globally unique identifier for the request. Used with asynchronous requests to correlate requests with their responses. If the value of the "execution" attribute is "**asynchronous**", the requestID attribute is required.
<operationalAttributes>	An optional element that MAY contain any number of <*attr*> elements. These are used to specify PSP-specific information with the request

The following schema fragment defines the SpmlRequest, ExecutionType, and <*operationalAttributes*> elements:

```
<xsd:complexType name="SpmlRequest">
  <xsd:sequence>
    <xsd:element name="operationalAttributes" type="spml:Attributes" minOccurs="0"
maxOccurs="1"/>
    <xsd:any minOccurs="0" maxOccurs="unbounded" processContents="lax" />
  </xsd:sequence>
  <xsd:attribute name="requestID" type="dsmluestID" use="optional"/>
  <xsd:attribute name="execution" type="spml:ExecutionType" use="optional"/>
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<xsd:simpleType name="ExecutionType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#synchronous"/>
    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#asynchronous"/>
  </xsd:restriction>
</xsd:simpleType>
```

6.2 Simple Type ExecutionType

Type ExecutionType is used to specify how a PSP is expected to respond to an SPML request. It is an enumeration of the following values:

Requested	An optional globally unique identifier used to correlate the response with the corresponding request. This MUST be returned by the PSP if one was specified in the SpmlRequest
result [required]	An indication of the overall status of the request
Error	An indication of the nature of a request error. This MUST be returned by the PSP only if the result attribute is equal to "**failure**".
<errorMessage>	An optional element containing text describing a request error in human readable form
<operationalAttributes>	An optional element that MAY contain any number of <*attr*> elements. These may be used to return PSP-specific information about the processing of the request.

The following schema fragment defines complex type SpmlResponse and related simple types:

```
<xsd:complexType name="SpmlResponse">
  <xsd:sequence>
    <xsd:element name="operationalAttributes" type="spml:Attributes" minOccurs="0"
maxOccurs="1"/>
    <xsd:element name="errorMessage" type="xsd:string" minOccurs="0"/>
    <xsd:any minOccurs="0" maxOccurs="unbounded" processContents="lax" />
  </xsd:sequence>
  <xsd:attribute name="result" type="spml:Result" use="required"/>
  <xsd:attribute name="requestID" type="dsmluestID" use="optional"/>
  <xsd:attribute name="error" type="spml:ErrorCode" use="optional"/>
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<xsd:simpleType name="Result">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#success"/>
    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#failure"/>
    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#pending"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ErrorCode">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#malformedRequest"/>
    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#unsupportedOperation"/>
    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#unsupportedIdentifierType"/>
    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#noSuchIdentifier"/>
    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#noSuchRequest "/>
    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#customError"/>
  </xsd:restriction>
</xsd:simpleType>
```

6.3 Simple type Result

The type Result defines the allowed values for the "result" attribute of type SpmlResponse.

Success	The request succeeded. For <*batchRequest*> elements, this indicates that all requests within the batch succeeded.
Failure	The request failed. For <*batchRequest*> elements, this indicates that at least one request within the batch failed.
Pending	The request has not yet been executed. This may be returned only if the "**execution**" attribute of the corresponding SpmlRequest was "asynchronous". For <*batchRequest*> elements, this indicates that at least one request within the batch is pending.

6.4 Simple type ErrorCode

The type ErrorCode is used to convey more detailed information about a request failure. It is an enumeration of the following values:

malformedRequest	Indicates a syntax error in the request. A PSP is required to return a well formed SPML response with this error code even if the request was not syntactically or semantically valid. If it’s syntactically valid but sematically invalid then you’ll get back this value.

unsupportedOperation	Indicates that the requested operation is not supported by this PSP.
unsupportedIdentifierType	Indicates that the identifier type used in the request is not supported by this PSP. This error may apply to any <*identifier*> <*providerIdentifier*> or <*operationIdentifier*> elements in the request.
noSuchIdentifier	Indicates that an identifier included in the request did not correspond to any service or service object supported by this PSP. This error may apply to any <identifier>, <*providerIdentifier*> or <*operationIdentifier*> elements in the request.
noSuchRequest	Indicates that an extended request was not recognized by the PSP. This error may apply to either the<*providerIdentifier*> or <*operationIdentifier*> elements in the corresponding <*ExtendedRequest*>.
customError	The error was PSP-specific and additional error text can be found in operationalAttributes.

6.5 Element <BatchRequest>

The <*batchRequest*> element is used to specify a collection of related SPML requests to be executed. It is of type BatchRequest which extends SpmlRequest. It defines the following attributes:

processing	Specifies the manner in which the PSP is expected to execute the requests in the batch. Values are defined by type processingType. If not specified, the default processing mode is "sequential".
onError	Specifies the manner in which the PSP responds to errors in individual requests within the batch. Values are defined by type OnErrorType. If not specified, the default error handling mode is "**exit**".

The <*batchRequest*> element may contain any number of <*AddRequest*>, <*ModifyRequest*>, <*DeleteRequest*>, and <*ExtendedRequest*> elements in any order.

The following schema fragment defines the <*batchRequest*> element and related types:

```
<xsd:complexType name="BatchRequest">
  <xsd:complexContent>
    <xsd:extension base="spml:SpmlRequest">
      <xsd:sequence>
        <xsd:group ref="spml:BatchRequests" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="processing" type="spml:ProcessingType" use="optional"
default="urn:oasis:names:tc:SPML:1:0#sequential" />
      <xsd:attribute name="onError" type="spml:OnErrorType" use="optional"
default="urn:oasis:names:tc:SPML:1:0#exit" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:simpleType name="ProcessingType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#sequential"/>
    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#parallel"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="OnErrorType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#resume"/>
    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#exit"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:group name="BatchRequests">
  <xsd:choice>
    <xsd:element name="ModifyRequest" type="spml:ModifyRequest" minOccurs="0"
maxOccurs="unbounded"/>
    <xsd:element name="AddRequest" type="spml:AddRequest" minOccurs="0"
maxOccurs="unbounded"/>
    <xsd:element name="DeleteRequest" type="spml:DeleteRequest" minOccurs="0"
maxOccurs="unbounded"/>
    <xsd:element name="ExtendedRequest" type="spml:ExtendedRequest" minOccurs="0"
maxOccurs="unbounded"/>
  </xsd:choice>
</xsd:group>
```

6.6 Simple Type ProcessingType

Type ProcessingType defines the allowed values for the "**processing**" attribute of element <*batchRequest*>. It is an enumeration of the following values:

sequential	Indicates that sequential processing is required. The PSP MUST execute the requests in the order in which they were specified.
parallel	Indicates that parallel processing is allowed. The PSP may execute the requests in any order.

6.7 Simple Type OnErrorType

Type OnErrorType defines the allowed values for the "**onError**" attribute of element <*batchRequest*>. It is an enumeration of the following values:

resume	Indicates that execution of the requests in the batch is allowed to continue when any individual request fails.

exit	Indicates that execution of the requests in the batch will terminate once any individual request fails.

6.8 Element <BatchResponse>

The <*batchResponse*> element is used to convey the result of a request specified with the <*batchRequest*> element. It is of type BatchResponse which extends SpmlResponse. The element may contain any number of <*AddResponse*>, <*ModifyResponse*>, <*DeleteResponse*>, and <*ExtendedResponse* > elements.

The PSP MUST return one response element for each request element within the <*batchRequest*>. The PSP MUST return response elements in the same order as the corresponding request elements. The PSP MUST return response elements whose types match the corresponding request element, for example an <*AddResponse*> must be returned for an <*AddRequest*>.

The number and order of response elements is unaffected by "sequential" or "parallel" processing, or by errors in individual requests.

If an error occurs in one request, and the <*batchRequest*> specifies an "**onError**" attribute value of "**exit**", response elements for all unprocessed requests will be returned with "**result**" attributes set to "**error**".

The following schema fragments define the <*batchResponse*> element and related types:

```
<xsd:complexType name="BatchResponse">
  <xsd:complexContent>
    <xsd:extension base="spml:SpmlResponse">
      <xsd:sequence>
        <xsd:group ref="spml:BatchResponses" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:group name="BatchResponses">
  <xsd:choice>
    <xsd:element name="ModifyResponse" type="spml:ModifyResponse" minOccurs="0"
maxOccurs="unbounded" />
    <xsd:element name="AddResponse" type="spml:AddResponse" minOccurs="0"
maxOccurs="unbounded"/>
    <xsd:element name="DeleteResponse" type="spml:DeleteResponse" minOccurs="0"
maxOccurs="unbounded"/>
    <xsd:element name="ExtendedResponse " type="spml:ExtendedResponse " minOccurs="0"
maxOccurs="unbounded"/>
  </xsd:choice>
</xsd:group>
```

6.9 Element <StatusRequest>

The <*StatusRequest*> element is used to retrieve the status of a request previously submitted to a PSP using an execution type of "asynchronous". It is of type StatusRequest which extends SpmlRequest. It defines the following attributes:

requestID [required]	Defined by type SpmlRequest, it must be specified to identify the previously submitted request. Note that unlike most SpmlRequests, this use of requestID applies not to this request but to another request.
statusReturns	Determines the type of status requested. The default is "**result**". The "**statusReturns**" attribute may have the following values: • Status - Indicates that the status of each request is to be returned. • Results - Indicates that both the status and request results are to be returned. Results may not yet be available or may not be complete.

The <*StatusRequest*> element MUST NOT specify an "**execution**" attribute value of "**synchronous**".

The following schema fragments define the <*StatusRequest*> element and related types:

```
<xsd:complexType name="StatusRequest">
  <xsd:complexContent>
    <xsd:extension base="spml:SpmlRequest">
      <xsd:attribute name="statusReturns" type="spml:StatusReturnsType" use="optional"
default="urn:oasis:names:tc:SPML:1:0#result" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:simpleType name="StatusReturnsType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#status "/>
    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#result"/>
  </xsd:restriction>
</xsd:simpleType>
```

6.10 Element <StatusResponse>

The <*StatusResponse*> element is used to convey the result of a request specified with the <*StatusRequest*> element. It is of type StatusResponse which extends SpmlResponse.

The element MUST contain a "statusResult" attribute which will have one of the following values:

success	The asynchronous request has completed.
pending	The asynchronous request has either not begun executing or is still executing.
nosuchRequest	The "requestID" specified in the <StatusRequest> was not a valid request identifier.

If the "statusReturns" attribute in the corresponding <StatusRequest> element was set to "results", the <StatusResponse> element MUST contain one of the response elements <AddResponse>, <ModifyResponse>, <DeleteResponse>, <SearchResponse>, <ExtendedResponse >, <SchemaResponse>, or <BatchResponse>.

If the "statusReturns" attribute in the corresponding <StatusRequest> element was set to "status" the <StatusResponse> elements MUST NOT contain any response elements.

If a response element is returned, it MUST match the corresponding request element, for example an <AddResponse> must be returned for an <AddRequest>.

The following schema fragment defines the <StatusResponse> element and related types:

```
<xsd:complexType name="StatusResponse">
  <xsd:complexContent>
    <xsd:extension base="spml:SpmlResponse">
      <xsd:sequence>
        <xsd:group ref="spml:StatusResponses" minOccurs="0" maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attribute name="statusResult" type="spml:StatusResultType" use="required"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:group name="StatusResponses">
  <xsd:choice>
    <xsd:element name="ModifyResponse" type="spml:ModifyResponse" minOccurs="0"
maxOccurs="unbounded" />
    <xsd:element name="AddResponse" type="spml:AddResponse" minOccurs="0"
maxOccurs="unbounded"/>
    <xsd:element name="DeleteResponse" type="spml:DeleteResponse" minOccurs="0"
maxOccurs="unbounded"/>
    <xsd:element name="ExtendedResponse " type="spml:ExtendedResponse " minOccurs="0"
maxOccurs="unbounded"/>
    <xsd:element name="SearchResponse" type="spml:SearchResponse" minOccurs="0"
maxOccurs="unbounded"/>
    <xsd:element name="SchemaResponse" type="SpmlResponse" minOccurs="0"
maxOccurs="unbounded"/>
    <xsd:element name="batchResponse" type="spml:BatchResponse" minOccurs="0"
maxOccurs="unbounded" />
  </xsd:choice>
</xsd:group>
```

6.12 Element <CancelRequest>

The <cancelRequest> element is used to terminate a request previously submitted to a PSP using an execution type of "asynchronous". It is of type StatusRequest which extends SpmlRequest. It defines the following attributes:

requestID [required]	Defined by type SpmlRequest, it MUST match the "requestID" attribute used by a previously submitted asynchronous request. Note that unlike most SpmlRequests, this use of requestID does not apply to this request but to another request.

The following schema fragment defines the <cancelRequest> element and related types:

```
 <xsd:element name="cancelRequest" type="spml:CancelRequest" />
<xsd:complexType name="CancelRequest">
```

```
  <xsd:complexContent>
    <xsd:extension base="spml:SpmlRequest">
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

6.12 Element <CancelResponse>

The <cancelResponse> element is used to convey the result of a request specifed with the <cancelRequest> element. It is of type StatusResponse which extends SpmlResponse.

The element MUST contain a "cancelResults" attributes with one of the following values:

canceled	The request was successfully canceled.
couldNotCancel	The request could not be canceled.
noSuchRequest	The "requestID" specified in the <cancelRequest> was not a valid request identifier.

```
<xsd:complexType name="CancelResponse">
  <xsd:complexContent>
    <xsd:extension base="spml:SpmlResponse">
      <xsd:attribute name="cancelResults" type="spml:CancelResultsType" use="required"  />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:simpleType name="CancelResultsType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#noSuchRequest "/>
    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#canceled"/>
    <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#couldNotCancel"/>
  </xsd:restriction>
</xsd:simpleType>
```

7 SPML Provisioning Schema (normative, with the exception of the schema fragments)

This section documents the SPML elements related to the specification of object schemas by the PSP.

7.2 Element <schemRequest>

The <SchemaRequest> element is used to request the descriptions of one or more schemas supported by a PSP. The element is of type SchemaRequest which extends SpmlRequest.

The element MAY contain a <providerIdentifier> element and MAY contain a <schemaIdentifier>. The content of both these identifiers is defined by the PSP.

If the request has neither <providerIdentifier> or <schemaIdentifier> then all schemas accessible through the PSP are requested.

If the request has a <providerIdentifier> but no <schemaIdentifier>, then all schemas associated with the given provider identifier are requested.

If the request has a <schemaIdentifier> but no <providerIdentifier>, then the <schemaIdentifier> must be specified as a URN that unambiguously identifies one schema.

The following schema fragment defines the <schemaRequest> element and related types:

```
<xsd:element name="schemaRequest" type="spml:SchemaRequest" />
<xsd:complexType name="SchemaRequest">
  <xsd:complexContent>
    <xsd:extension base="spml:SpmlRequest">
      <xsd:sequence>
        <xsd:element name="providerIdentifier" type="spml:ProviderIdentifier" minOccurs="0"
maxOccurs="1"/>
        <xsd:element name="schemaIdentifier" type="spml:SchemaIdentifier" minOccurs="0"
maxOccurs="1"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

7.2 Element <schemaResponse>

The <schemaResponse> element is used to convey the result of a request specified with the <schemaRequest> element. It is of type SchemaResponse which extends SpmlResponse.

If successful, the element MUST contain one or more <schema> elements which describe the requested schemas.

The following schema fragment defines the <schemaResponse> element and related types:

```
<xsd:element name="schemaResponse" type="spml:SchemaResponse" />
<xsd:complexType name="SchemaResponse">
  <xsd:complexContent>
    <xsd:extension base="spml:SpmlResponse">
      <xsd:sequence>
        <xsd:element name="schema" type="spml:Schema" minOccurs="1"
maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

7.3 Element <schema>

The <schema> element conveys the description of one schema supported by a PSP.

The element MUST contain a <providerIdentifier> which identifies the provider associated with the schema.

The element MUST contain a <schemaIdentifier> which identifies the schema within the context of the associated provider identifier. If the <schemaIdentifier> is a URN, then references to this schema MAY be made using only the <schemeIdentifier>. If the <schemaIdentifier> is not a URN, then references to the schema MUST be made using both <providerIdentifier> and <schemaIdentifer>.

The element MAY contain zero or more <objectclassDefinition> elements which describe the object classes in the schema. The "name" attribute of all <objectclassDefinition> elements within the <schema> MUST be unique. Different schemas MAY contain object class definitions withthe same name.

The element MAY contain zero or more <attributeDefinition> elements which describe the attributes used in the object class and extended request definitions. The "name" attribute of all <attributeDefinition> elements within the <schema> MUST be unique. Different schemas MAY contain attribute definitions with the same name.

The element MAY contain zero or more <extendedRequestDefinition> elements. The <operationIdentifier> elements of all <extendedRequestDefinition> elements within the <schema> MUST be unique. Different schemas MAY contain extended request definitions with the same name.

The following schema fragment defines the <schema> element and related types:

```
<xsd:element name="schema" type="spml:Schema" minOccurs="1" maxOccurs="unbounded"/>
<xsd:complexType name="Schema">
 <xsd:sequence>
  <xsd:element name="providerIdentifier" type="spml:ProviderIdentifier" minOccurs="1" 
maxOccurs="1" />
  <xsd:element name="schemaIdentifier" type="spml:SchemaIdentifier" minOccurs="1" 
maxOccurs="1" />
  <xsd:element name="objectclassDefinition" type="spml:ObjectclassDefinition" minOccurs="0" 
maxOccurs="unbounded"/>
  <xsd:element name="attributeDefinition" type="spml:AttributeDefinition" minOccurs="0" 
maxOccurs="unbounded"/>
  <xsd:element name="extendedRequestDefinition" type="spml:ExtendedRequestDefinition" 
minOccurs="0" maxOccurs="unbounded"/>
  <xsd:any minOccurs="0" maxOccurs="unbounded" processContents="lax" />
 </xsd:sequence>
 <xsd:attribute name="majorVersion" type="xsd:string"/>
 <xsd:attribute name="minorVersion" type="xsd:string"/>
 <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>
```

7.4 Element <schemaIdentifier>

The <schemaIdentifier> element is used to identify a schema that may be requested in a <schemaRequest>, or referenced in an <objectclassDefinitionReference> or <attributeDefinitionReference>.

The element MUST contain a "schemaIDType" attribute that specifies the type of the identifier. It MUST be one of:

URN	The identifier is a URN.
OID	The identifier is an OID.
GenericString	The identifier is a string whose structure is specific to the PSP.

The element MUST contain a <schemaID> element whose content is consistent with the value of the "schemaIDType" attribute.

The following schema fragment defines the <schemaIdentifier> element and related types:

```
 <xsd:element name="schemaIdentifier" type="spml:SchemaIdentifier" minOccurs="0"
maxOccurs="1" />
<xsd:complexType name="SchemaIdentifier">
  <xsd:sequence>
    <xsd:element name="schemaID" type="xsd:anyType" />
    <xsd:any minOccurs="0" maxOccurs="unbounded" processContents="lax" />
  </xsd:sequence>
  <xsd:attribute name="schemaIDType" use="required">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#URN"/>
        <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#OID"/>
        <xsd:enumeration value="urn:oasis:names:tc:SPML:1:0#GenericString"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>
```

7.5 Element <properties>

The <properties> element MAY be used in <objectclassDefinition>, <attributeDefinition>, and <extendedRequestDefinition> elements to convey PSP-specific information about the semantics of the related definition.

The element MAY contain any number of DSML <attr> elements. The names and semantics of the property attributes is defined by the PSP.

The following schema fragment defines the <properties> element and related types:

```
 xsd:element name="properties" type="spml:Properties" minOccurs="0" maxOccurs="1"/>
<xsd:complexType name="Properties">
  <xsd:sequence>
    <xsd:element name="attr" type="dsml:DsmlAttr" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

7.6 Element <attributeDefinition>

The <attributeDefinition> element is used to describe the definition of one attribute within a schema.

The element MUST have a "name" attribute which MUST be unique among <attributeDefinition> elements within a <schema>.

The element MAY have a "type" attribute which specifies the fundamental data type of this attribute. If type is not specified, the default is "xsd:string" indicating that the attribute is a character string.

The element MAY have a "multi-valued" attribute whose value is either "true" or "false". When the value is "true", it indicates that more than one value is allowed for this attribute.

The element MAY have a "description" attribute which should contain readable text that describes the semantics of this attribute.

The element MAY have <properties> element to convey PSP-specific information about an attribute.

The following schema fragment defines the <attributeDefinition> element and related types:

```
 <<xsd:element name="attributeDefinition" type="spml:AttributeDefinition" minOccurs="0"
maxOccurs="unbounded"/>
<xsd:complexType name="AttributeDefinition">
  <xsd:sequence>
    <xsd:element name="properties" type="spml:Properties" minOccurs="0" maxOccurs="1"/>
    <xsd:any minOccurs="0" maxOccurs="unbounded" processContents="lax" />
  </xsd:sequence>
  <xsd:attribute name="name" type="dsml:AttributeDescriptionValue" use="required"/>
  <xsd:attribute name="description" type="xsd:string" use="optional"/>
  <xsd:attribute name="multivalued" type="xsd:boolean" use="optional" default="false"/>
  <xsd:attribute name="type" type="xsd:string" use="optional" default="xsd:string"/>
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>
```

7.7 Element <attributeDefinitionReference> and Type AttributeDefinitionReferences

The <attributeDefinitionReference> element is used to refer to an <attributeDefinition> element from within another element.

The element MUST have a "name" attribute which MUST correspond to the name of an <attributeDefinition> defined in a <schema>.

The element MAY have a <providerIdentifier> element and MAY have a <schemaIdentifier> element. These two elements are used to refer to attributes defined in a schema other than the one in which the reference is contained.

If both the <providerIdentifier> and <schemaIdentifier> are omitted, then an <attributeDefinition> whose "name" attribute matches the "name" of the reference MUST be defined in the <schema> that contains the reference.

If both the <providerIdentifier> and <schemaIdentifier> are specified, <providerIdentifier> must be a URN, and <schemaIdentifier> must be a non-URN.

If only <schemaIdentifier> is specified, it MUST be a URN. If only <providerIdentifier> is specified, the request is malformed.

The combination of the <providerIdentifier> and <schemaIdentifier> MUST identify a schema supported by the PSP.

The element MAY have a "required" attribute whose values may be "true" or "false". When "true" it indicates that a value for this attribute is required. The semantics of a required attribute are defined by the context of the reference which will be either an <objectclassDefinition> or <extendedRequesteDefinition>.

The type AttributeDefinitionReferences defines an unbounded sequence of <attributeDefinitionReference> elements.

The following schema fragment defines the <attributeDefinitionReference> element and related types:

```
 <xsd:element name="attributeDefinitionReference" type="spml:AttributeDefinitionReference"
  minOccurs="0" maxOccurs="unbounded"/>
<xsd:complexType name="AttributeDefinitionReference">
  <xsd:sequence>
    <xsd:element name="providerIdentifier" type="spml:ProviderIdentifier" minOccurs="0"
maxOccurs="1" />
    <xsd:element name="schemaIdentifier" type="spml:SchemaIdentifier" minOccurs="0"
maxOccurs="1" />
    <xsd:any minOccurs="0" maxOccurs="unbounded" processContents="lax" />
  </xsd:sequence>
  <xsd:attribute name="name" type="dsml:AttributeDescriptionValue" use="required"/>
  <xsd:attribute name="required" type="xsd:boolean" use="optional" default="false"/>
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<xsd:complexType name="AttributeDefinitionReferences">
  <xsd:sequence>
    <xsd:element name="attributeDefinitionReference" type="spml:AttributeDefinitionReference"
minOccurs="0" maxOccurs="unbounded"/>
    <xsd:any minOccurs="0" maxOccurs="unbounded" processContents="lax" />
  </xsd:sequence>
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>
```

7.8 Element <objectclassDefinition>

The <objectclassDefinition> element is used to describe the definition of one object class in a schema.

The element MUST have a "name" attribute which MUST be unique among <objectclassDefinition> elements within a <schema>.

The element MAY have a "description" attribute which should contain readable text that describes the semantics of this class.

The element MAY have <properties> element to convey PSP-specific information about a class.

The element MAY have a <superiorClasses> element which is defined by type ObjectclassDefinitionReferences and contains any number of <objectclassReference> elements. When a class defines superior classes, it indicates that all attributes defined by a superior class are also defined for the referencing class.

The element MAY have a <memberAttributes> element which is defined by type AttributeDefinitionReferences and contains any number of <attributeDefinitionReference> elements.

The following schema fragment defines the <objectclassDefinition> element and related types:

```
 <xsd:element name="objectclassDefinition" type="spml:ObjectclassDefinition" minOccurs="0"
maxOccurs="unbounded"/>
<xsd:complexType name="ObjectclassDefinition">
  <xsd:sequence>
    <xsd:element name="properties" type="spml:Properties" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="memberAttributes" type="spml:AttributeDefinitionReferences"
```

```
minOccurs="0" maxOccurs="1"/>
    <xsd:element name="superiorClasses" type="spml:ObjectclassDefinitionReferences"
minOccurs="0" maxOccurs="1"/>
    <xsd:any minOccurs="0" maxOccurs="unbounded" processContents="lax" />
  </xsd:sequence>
  <xsd:attribute name="name" type="dsml:AttributeDescriptionValue" use="required"/>
  <xsd:attribute name="description" type="xsd:string" use="optional"/>
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<xsd:complexType name="ObjectclassDefinitionReferences">
  <xsd:sequence>
    <xsd:element name="objectclassDefinitionReference"
type="spml:ObjectclassDefinitionReference" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:any minOccurs="0" maxOccurs="unbounded" processContents="lax" />
  </xsd:sequence>
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>
```

7.9 Element <objectclassDefinition>

The <objectclassDefinitionReference> element is used to refer to an <objectclassDefinition> element from within the <superiorClasses> element of another <objectclassDefinition>.

The element MUST have a "name" attribute which MUST correspond to the name of an <objectclassDefinition> defined in a <schema>.

The element MAY have a <providerIdentifier> element and MAY have a <schemaIdentifier> element. These two elements are used to refer to classes defined in a schema other than the one in which the reference is contained.

If both the <providerIdentifier> and <schemaIdentifier> are omitted, then an <objectclassDefinition> whose "name" attribute matches the "name" of the reference MUST be defined in the <schema> that contains the reference.

If both the <providerIdentifier> and <schemaIdentifier> are specified, <providerIdentifier> must be a URN, and <schemaIdentifier> must be a non-URN.

If only <schemaIdentifier> is specified, it MUST be a URN.

If only <providerIdentifier> is specified, the request is malformed.

The combination of the <providerIdentifier> and <schemaIdentifier> MUST identify a schema supported by the PSP.

The following schema fragment defines the <objectclassDefinitionReference> element and related types:

```
 <xsd:element name="objectclassDefinitionReference" type="spml:ObjectclassDefinitionReference"
  minOccurs="0" maxOccurs="unbounded"/>
<xsd:complexType name="ObjectclassDefinitionReference">
  <xsd:sequence>
    <xsd:element name="providerIdentifier" type="spml:ProviderIdentifier" minOccurs="0"
maxOccurs="1" />
    <xsd:element name="schemaIdentifier" type="spml:SchemaIdentifier" minOccurs="0"
maxOccurs="1" />
    <xsd:any minOccurs="0" maxOccurs="unbounded" processContents="lax" />
  </xsd:sequence>
  <xsd:attribute name="name" type="dsml:AttributeDescriptionValue" use="required"/>
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>
```

7.10 Element <extendedRequestDefinition>

The <extendedRequestDefinition> element is used to describe the definition of one extended request supported by a PSP.

The element MUST have a <operationIdentifier> element which uniquely identifies this request among requests in the schema.

The element MAY have a "description" attribute which should contain readable text that describes the semantics of this class.

The element MAY have <properties> element to convey PSP-specific information about the extended request.

The element MAY have a <parameters> element of type AttributeDefinitionReferences containing any number of <attributeDefinitionReference> elements. These define the parameters that may be passed in the <attributes> element of an <extendedRequest>. If an <attributeDefinitionReference> has a "required" attribute of "true", then a value for that attribute MUST be specified in the <extendedRequest>.

The element MAY have a <returnValues> element of type AttributeDefinitionReferences containing any number of <attributeDefinitionReference> elements. These define the attributes that may be returned to the RA in the <attributes> element of an <extendedResponse>. If an <attributeDefinitionReference> has a "required" attribute of "true", then the PSP MUST include a value for that attribute in the <extendedResponse>.

The following schema fragment defines the <extendedRequestDefinition> and related types:

```
 <xsd:element name="extendedRequestDefinition" type="spml:ExtendedRequestDefinition"
  minOccurs="0" maxOccurs="unbounded"/>
<xsd:complexType name="ExtendedRequestDefinition">
  <xsd:sequence>
    <xsd:element name="operationIdentifier" type="spml:OperationIdentifier" minOccurs="1"
maxOccurs="1" />
    <xsd:element name="properties" type="spml:Properties" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="parameters" type="spml:AttributeDefinitionReferences" minOccurs="0"
maxOccurs="1"/>
    <xsd:element name="returnValues" type="spml:AttributeDefinitionReferences" minOccurs="0"
maxOccurs="1"/>
    <xsd:any minOccurs="0" maxOccurs="unbounded" processContents="lax" />
  </xsd:sequence>
  <xsd:attribute name="description" type="xsd:string" use="optional"/>
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>
```

8. Security and privacy considerations (non-normative)

This section identifies possible security and privacy compromise scenarios that should be considered when implementing an SPML-based system. The section is informative only. It is left to the implementer to decide whether these compromise scenarios are practical in their environment and to select appropriate safeguards.

8.1. Threat model

We assume here that the adversary has access to the communication channel between the SPML actors and is able to interpret, insert, delete and modify messages or parts of messages.

8.1.1. Unauthorized disclosure

SPML does not specify any inherent mechanisms for confidentiality of the messages exchanged between actors. Therefore, an adversary could observe the messages in transit. Under certain security policies, disclosure of this information is a violation. Disclosure of provisioning data may have significant repercussions. In the commercial sector, the consequences of unauthorized disclosure of personal data may range from embarrassment to the custodian to imprisonment and large fines in the case of medical or financial data.

Unauthorized disclosure is addressed by confidentiality mechanisms.

8.1.2. Message Replay

A message replay attack is one in which the adversary records and replays legitimate messages between SPML actors. This attack may lead to denial of service, the use of out-of-date information or impersonation.

Prevention of replay attacks requires the use of message freshness mechanisms.

Note that encryption of the message does not mitigate a replay attack since the message is just replayed and does not have to be understood by the adversary.

8.1.2.1. Message insertion

A message insertion attack is one in which the adversary inserts messages in the sequence of messages between SPML actors.

The solution to a message insertion attack is to use mutual authentication and a message sequence integrity mechanism between the actors. It should be noted that just using SSL mutual authentication is not sufficient. This only proves that the other party is the one identified by the subject of the X.509 certificate. In order to be effective, it is necessary to confirm that the certificate subject is authorized to send the message.

8.1.2.2. Message deletion

A message deletion attack is one in which the adversary deletes messages in the sequence of messages between SPML actors. Message deletion may lead to denial of service. However, a properly designed SPML system should not trigger false provisioning on as the result of a message deletion attack.

The solution to a message deletion attack is to use a message integrity mechanism between the actors.

8.1.2.3. Message modification

If an adversary can intercept a message and change its contents, then they may be able to alter a provisioning request. Message integrity mechanisms can prevent a successful message modification attack.

8.2. Safeguards

8.2.1. Authentication

Authentication provides the means for one party in a transaction to determine the identity of the other party in the transaction. Authentication may be in one direction, or it may be bilateral.

Given the sensitive nature of many provisioning requests and systems it is important for an ***RA*** to authenticate the identity of the ***PSP*** to which it issues SPML requests. Otherwise, there is a risk that an adversary could provide false or invalid ***PSP***, leading to a possible security violation.

It is equally important for a ***PSP*** to authenticate the identity of the ***RA*** and assess the level of trust and to determine if the ***RA*** is authorized to request this service/operation.

Many different techniques may be used to provide authentication, such as co-located code, a private network, a VPN or digital signatures. Authentication may also be performed as part of the communication protocol used to exchange the requests. In this case, authentication may be performed at the message level or at the session level.

8.2.2. Confidentiality

Confidentiality mechanisms ensure that the contents of a message can be read only by the desired recipients and not by anyone else who encounters the message while it is in transit. The primary concern is confidentiality during transmission.

8.2.2.1. Communication confidentiality

In some environments it is deemed good practice to treat all data within a provisioning domain as confidential. In other environments certain parts of the service schema and required attributes may b openly published. Regardless of the approach chosen, the security of the provisioning system as a whole should not be in any way dependant on the secrecy of the service, its provider or its request data schema.

Any security concerns or requirements related to transmitting or exchanging SPML documents lies outside the scope of the SPML standard. While it is often important to ensure that the integrity and confidentiality of provisioning requests, it is left to the implementers to determine the appropriate mechanisms for their environment.

Communications confidentiality can be provided by a confidentiality mechanism, such as SSL. Using a point-to-point scheme like SSL may lead to other vulnerabilities when one of the end-points is compromised.

8.2.2.2. Trust model

Discussions of authentication, integrity and confidentiality mechanisms necessarily assume an underlying trust model: how can one actor come to believe that a given key is uniquely associated with a specific, identified actor so that the key can be used to encrypt data for that actor or verify signatures (or other integrity structures) from that actor? Many different types of trust model exist, including strict hierarchies, distributed authorities, the Web, the bridge and so on.

8.2.2.3. Privacy

It is important to be aware that any transactions that occur in an SPML model system may contain private and secure information about the actors. Selection and use of privacy mechanisms appropriate to a given environment are outside the scope of this specification. The decision regarding whether, how and when to deploy such mechanisms is left to the implementers associated with the environment.

9. Conformance (normative)

9.1. Introduction

The OASIS procedure for ratification of a committee specification as an OASIS standard requires that three independent implementers attest that they are "successfully using" the committee specification. This requirement has been met and implementation details have been submitted with this specification.

9.2. Conformance tables

This section lists those portions of the specification that MUST be included in an implementation of an RA or an SPML service that claims to conform to SPML v1.0.

Note: "M" means mandatory to implement. "O" means optional to implement. “O* means optional to implement but must implement one of <*AddRequest*>, <*ModifyRequest*> or <*DeleteRequest*>. “NA” means does not apply.

The implementation MUST support ALL those schema elements that are marked “M” and MUST support one of either <*AddRequest*>, <*ModifyRequest*> or <*DeleteRequests*> in the columns marked “O*”

Element name	RA	PSP Server	PSP Client	PST
spml:AddRequest	O*	M	O*	O*
spml:ModifyRequest	O*	M	O*	O*
spml:DeleteRequest	O*	M	O*	O*
spml:SearchRequest	O	O	O	O
spml:ExtendedRequest	O	O	O	O
Support for the synchronous SPML operations model	M	M	M	M

(requires support for spml:BatchRequest)				
Support for the asynchronous SPML operations model (requires support for spml:BatchStatus and spml:BatchCancel)	O	O	O	O
Provide an SPML compliant definition of all published services	NA	M	NA	M
Support the SpmlRequest operation for all published service	O	O	O	O

9.3 Data Types

The implementation MUST support the data types associated with the following identifiers marked "M".

Data-type	M/O
`http://www.w3.org/2001/XMLSchema#string`	M
`http://www.w3.org/2001/XMLSchema#boolean`	M
`http://www.w3.org/2001/XMLSchema#integer`	M
`http://www.w3.org/2001/XMLSchema#double`	M
`http://www.w3.org/2001/XMLSchema#date`	M
`http://www.w3.org/2001/XMLSchema#dateTime`	M
`http://www.w3.org/2001/XMLSchema#anyURI`	M
`http://www.w3.org/2001/XMLSchema#hexBinary`	M
`http://www.w3.org/2001/XMLSchema#base64Binary`	M

Appendix A. References

[ARCHIVE-1]	OASIS Provisioning Services Technical Committee., email archive, **http://www.oasis-open.org/apps/org/workgroup/provision/email/archives/index.html**, OASIS PS-TC
[SPML-REQ]	OASIS Provisioning Services Technical Committee., Requirements, **http://www.oasis-open.org/apps/org/workgroup/provision/download.php/2277/draft-pstc-requirements-01.doc**, OASIS PS-TC
[RFC2119]	S. Bradner., *Key words for use in RFCs to Indicate Requirement Levels*, **http://www.ietf.org/rfc/rfc2119.txt**, IETF
[DSML]	OASIS Directory Services Markup TC., *DSML V2.0 Specification*, **http://www.oasis-open.org/apps/org/workgroup/dsml/documents.php**, OASIS DS-TC
[SAML]	OASIS Security Services Technical Committee., *XMLTitle*, **http://www.oasis-open.org/apps/org/workgroup/sstc/documents.php**, OASIS SS-TC
[DS]	IETF/W3C., *W3C XML Signatures*, **http://www.w3.org/Signature/**, W3C/IETF
[XS]	W3C Schema WG ., *W3C XML Schema*, **http://www.w3.org/TR/xmlschema-1/** W3C
[XRPM]	XRPM Working Group, (Disbanded)
[ADPR]	Active Digital Profile Group., **http://www.adpr-spec.com/**
[PSTC-UC]	OASIS Provisioning Services Technical Committee., SPML V1.0 Use Cases , **http://www.oasis-open.org/apps/org/workgroup/provision/download.php/988/drfat-spml-use-cases-05.doc**, OASIS PS-TC
[SPML-Bind]]	OASIS Provisioning Services Technical Committee., SPML V1.0 Protocol Bindings, **http://www.oasis-open.org/apps/org/workgroup/provision/download.php/1816/draft-pstc-bindings-03.doc**, OASIS PS-TC
[SPML-RoadMap]]	OASIS Provisioning Services Technical Committee,Draft PSTC (SPML Roadmap, **http://www.oasis-open.org/committees/download.php/2368/draft-pstc-roadmap-01.doc**, OASIS PS-TC

Appendix B. Revision history

Rev	Date	By whom	What
CS-1.0	3June 2003	Editor	1.0 Committee Specification

Appendix C. Notices