OASIS

1

Service Provisioning Markup Language (SPML) Version 1.0

2
3

OASIS Standard, Approved October 2003

4

8 Editor:
9   Darran Rolls, Waveset Technologies (Darran.Rolls@waveset.com)
10

11 Contributors:

12   Archie Reed, Critical Path
13   Doron Cohen, BMC
14   Gavenraj Sodhi, Business Layers
15   Gerry Woods, IBM
16   Hal Lockhart, BEA
17   Jeff Bohren, OpenNetwork Technologies
18   Jeff Larson, Waveset Technologies
19   Jesus Fernandez, Computer Associates
20   Matthias Leibmann, Microsoft
21   Mike Polan, IBM
22   Paul Madsen, Entrust
23   Rami Elron, BMC
24   Tony Gallotta, Access, IBM
25   Yoav Kirsh, Business Layers
26

27 Abstract:

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

30

31 Status:

os-pstc-spml-core-1.0.pdf

32   This PSTC Committee Specification was accepted as an OASIS Open Standard in October
33   2003.

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

38

39 Copyright (C) OASIS Open 2002. All Rights Reserved.

Table of contents

167

168 1. Introduction (non-normative)

169 1.1. Glossary

170 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.

171

172   **1.2.   Notation**

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

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

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

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

184   Listings of SPML schemas appear like this.

185

186   Example code listings appear like this.

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

190   • The prefix dsml: stands for the Directory Services Markup Lanuage namespace **[DSML].**

191   • The prefix saml: stands for the SAML assertion namespace [**SAML**].

192   • The prefix ds: stands for the W3C XML Signature namespace [**DS**].

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

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

1.3. Schema organization and namespaces

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

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

2. Background (non-normative)

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

2.2. What does service provisioning mean?

207 Service provisioning means many different things to many different people. In the context of this
208 specification it refers to the "preparation beforehand" of IT systems' "materials or supplies" required
209 to carry out some defined activity. It goes further than the initial "contingency" of providing
210 resources, to the onward management lifecycle of these resources as managed items. This could
211 include the provisioning of purely digital services like user accounts and access privileges on
212 systems, networks and applications. It could also include the provisioning of non-digital or "physical"
213 resources like the requesting of office space, cell phones and credit cards.
214
215 The following short definition has been adopted by the Provisioning Services Technical Committee
216 as its formal definition of the general term "provisioning":

217

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

221

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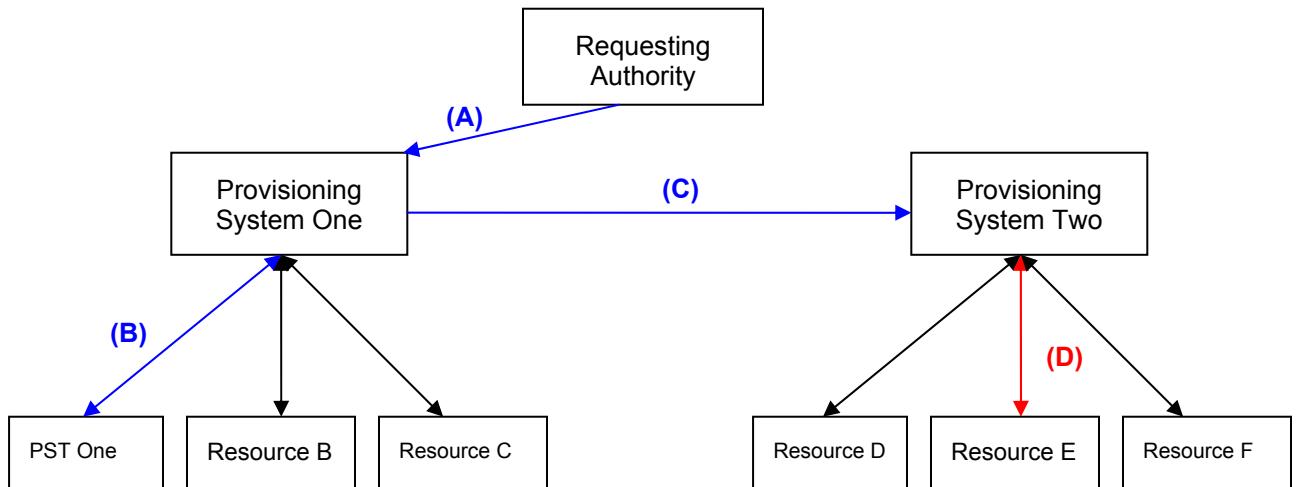
223 At this stage it is not necessary to define the implementation or physical makeup of a service
224 provisioning system. Simply assume the existence of a network service whose sole purpose is the
225 execution and management of provisioning requests. A given Requesting Authority (client) sends
226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

230 Figure 1 shows a high-level schematic of the operational components of an SPML model system. In
231 SPML request flow A, the Requesting Authority (client) constructs an SPML document subscribing
232 to a pre-defined service offered by Provisioning System One (PS One). PS One takes the data

233   passed in this SPML document, constructs its own SPML document and sends it to PST one
234   (SPML request flow B). PST One represents an independent resource that provides an SPML-
235   compliant service interface. In order to fully service the initial Requesting Authority's request, PS
236   One then forwards a provisioning request (SPML request flow C) to a second network service
237   called Provisioning System Two (PS Two). PS Two is autonomously offering a provisioning service
238   it refers to as Resource E. In this case, Resource E is a relational database within which PS Two
239   creates some data set. Having successfully received PS One's request, PS Two carries out the
240   implementation of its service by opening a JDBC connection to Resource E and adding the relevant
241   data (data flow D).

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

248

250 **Figure 1. Provisioning Systems.**

251

2.4.   Why do we need service provisioning standards?

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

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

2.5. Requirements

264

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

268 **2.5.1** To define an extensive set of use cases that model the functional requirements of
269 the proposed protocol and to see these use cases operable by implementing the
270 resulting specification.
271

272 **2.5.2** To define an XML Schema based protocol for exchanging provisioning requests
273 between a Requesting Authority (RA) and a Provisioning Service Point (PSP).
274

275 **2.5.3** To define an XML Schema based protocol for exchanging requests provisioning
276 requests between a PSP and a Provisioning Service Target (PST) AND if possible to
277 implement this and requirement 1 (above) in a single protocol.
278

279 **2.5.4** To provide a query model that MAY allow a RA to discover details about those
280 provisioning elements it is authorized to see and act upon at a given PSP. Implicitly,
281 the "decision" on what services to display to what RA's lies with the implementation and
282 authorization model of the PSP provider.
283

284 **2.5.5** To provide a model that allows a RA and a PSP to dynamically discover the
285 required data values for a given provisioning action.
286

287 **2.5.6** To provide consideration for the security and general operational concerns of such
288 an exchange system.
289

290 **2.5.7** To provide guidelines on binding SPML to the SOAP and HTTP protocols.
291

292 **2.5.8** To provide an open extensible solution that is independent of any one vendors
293 implementation or solutions model.
294

295 **2.5.9** To provide a transactional element to the request/response model that allows for
296 the exchange of ordered batches of requests.
297

298 **2.5.10** To deliver a solution in a timely manor.
299

300 **2.5.11** To where possible and reasonable to re-use and extend existing standards efforts
301 for the benefit of the SPML solution.
302

303 **2.5.12** To provide a standard suitable for use both inside a single organization or
304 enterprise and between multiple separate organizations/enterprises operating on
305 separate network infrastructures.
306

307 **2.5.13** To provide an open protocol that does not dictate the underlying infrastructure or
308 technology used by the implementer of RA, PSP or PST entities to support that
309 protocol.

2.6. Use Cases

310

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

3. Models (non-normative)

315

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

3.1 Protocol Overview

318

319
320 The general model adopted by this protocol is one of clients performing protocol operations against
321 servers. In this model, a client issues an SPML request describing the operation to be performed at
322 a given service point. The service point is then responsible for performing the necessary
323 operation(s) to constitute the implementation of the requested service. Upon completion of the
324 operation(s), the service point returns to the client an SPML response detailing any results or errors
325 pertinent to that request.

326
327 In order to promote standardization of the service subscription and provisioning interface, it is an
328 active goal of this protocol to minimize the complexity of the client interface in order to promote
329 widespread deployment of applications capable of issuing standardized service provisioning
330 requests. With this goal in mind SPML builds on a simplistic core operations model in which the
331 semantics of an individual provisioning action lay in the definition of the underlying service schema.
332 The core operations schema provides a small number of generic operations (Add, Modify, Delete,
333 Search) and an open model for the definition and discovery of that schema as a set of simple
334 name=(multi)value pairs. To complement this, SPML V1.0 also provides an operations extension
335 model based on an <*ExtendedRequest*> operation that allows individual providers to define new
336 operations that do not overlap with V1.0 core operations.

337
338 SPML V1.0 provides both a synchronous and asynchronous batch request model. However, there
339 is no requirement for a blocking synchronous behavior on the part of either clients or servers in
340 either operating model. Requests and responses for multiple operations may be freely exchanged
341 between a client and server in any order, provided the client eventually receives a response for
342 every request that requires one.
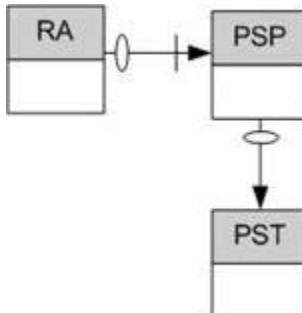
343
344 The SPML V1.0 XML Schema follows an "Open Content Model". In this model, any SPML element
345 in an XML document can have additional child elements and attributes that are not declared in the
346 core SPML V1.0 schema. In contrast, a "Closed Content Model", would prevent the inclusion of
347 any elements not defined in the SPML core XSD.
348

3.2 Domain Model

349

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

353

354

355

356   **Figure 2. Conceptual system elements**

3.2.1 Introduction to RA

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

3.2.2 Introduction to PSP

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

3.2.3 Introduction to PST

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

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

389   3.2.4 Introduction to PSO-ID

390   SPML introduces the concept of a Provisioning Service Object Identifier (PSO-ID). A PSO-ID
391   represents a unique identifier for a collection of individual provisioning requests. An example
392   explains this best.
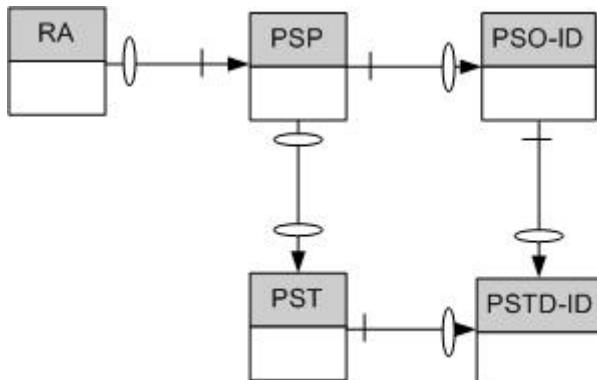
393   Consider the provisioning of IT resource accounts for a new corporate user. The new user requires
394   an account on a Windows NT domain, a Lotus Notes server, a corporate directory server and a
395   UNIX file server. In this example the RA would present the PSP with it's own unique identifier for
396   the "corporate user", say a full name, a list of the PSTD-ID's it would like to create on the target
397   systems (see below) and the set of attributes required to complete the provisioning request. In this
398   example, the PSO-ID would be the full name specified by the RA. The PSO-ID would be used to
399   relate the created PSTD-ID's together. This relationship could be maintained by both the RA and
400   the PSP, the details of which is deliberately left un-defined in the SPML protocol.

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

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

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

408

409

410

411   **Figure 3. High-level system element relationships**

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

419   3.2.5 An Introduction to PSTD-ID

420   SPML introduces the concept of a Provisioning Service Target Data Identifier or PSTD-ID. A
421   PSTD-ID is a unique identifier for a data set (account or managed data) on a PST. An example of a
422   PSTD-ID on a UNIX/Linux server would be the UID; an example of a PSTD-ID for a directory entry

423 would be a Distinguished Name (DN). In some cases PSTD-ID's are specified by the RA issuing
424 the SPML request. In others the PSTD-ID is set by the PSP/PST.

425 It is assumed that a PSTD-ID is unique to a PST (if not implemented by the native resource then
426 implemented by the functioning PST/PSP implementation through some custom namespace
427 mechanism).
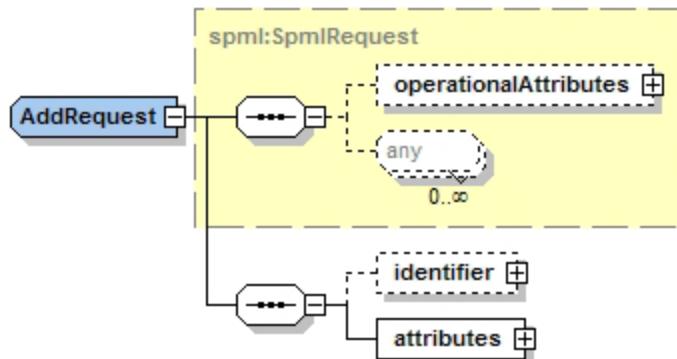
428 The simple ERD shown in Figure 3 shows some of the possible relationships between a RA, PSP,
429 PSO-ID, PST and PSTD-ID. Remember that these relations are not always directly reflected in the
430 SPML 1.0 protocol; often they explain the model behavior of the entire system.
431

432 ## 3.3   Operations Overview
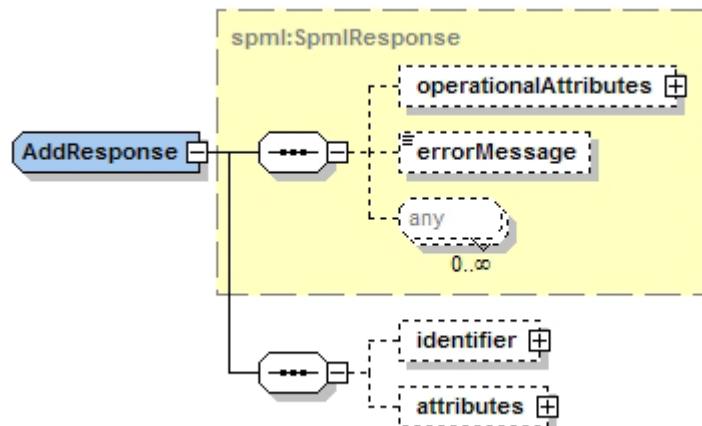
433 This section provides a non-normative discussion of the specific operations defined in SPML
434 version 1.0 and describes them in relative terms to the overall design and purpose of SPML.

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

437 ### 3.3.1 SPML Add Operation

438

439

440 The *<AddRequest>* element allows an RA to request the creation of a new object of an object as
441 defined by its object classes. The *<attributes>* sub element is used to envelope the set of

442   name=(multi)value pairs required to subscribe to a given published service. Note the use of the
443   special attribute "**objectclass**". This attribute is used to define the target service schema the RA
444   wishes to create an object of. The example below shows a request to "**create**" an object of the
445   "**emailUser**" service. The RA supplies the "**cn**" "**gn**" and "**sn**" attributes required to subscribe to this
446   service. By issuing this request, the RA hopes to create a new email account for the "**cn**" Jane
447   Doe.
448
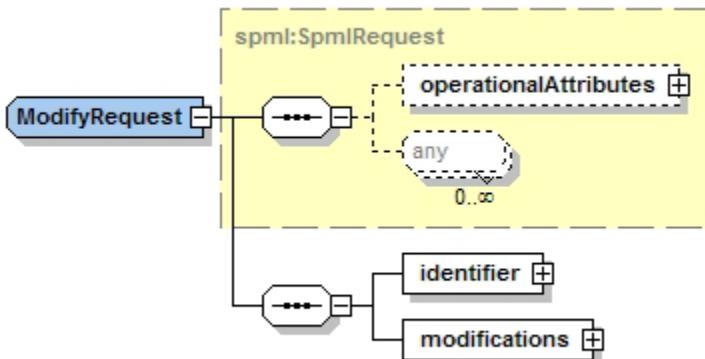
```xml
<addRequest>
    <attributes>
        <attr name="objectclass">
          <value>emailUser</value>
        </attr>
        <attr name="cn">
            <value>Jane Doe</value>
        </attr>
        <attr name="gn">
            <value>Jane</value>
        </attr>
        <attr name="sn">
            <value>Doe</value>
        </attr>
    </attributes>
</addRequest>
```

449
450   The resulting *<AddResponse>* element returned by the PSP supplies the RA with a result code
451   attribute to indicate that the add request was successfully processed. In this example, the PSP
452   returns an *<identifier>* element containing the primary record identifier for the newly created
453   service object (PSTD). This value will be required for subsequent operations on this newly created
454   PSTD-ID. The response element also includes an optional *<attributes>* element to envelope a set
455   of data generated by the functioning PST. In this example, the PSP uses these returned attributes
456   to notify the RA that the "mailBoxLimit" attribute was automatically set to "50MB".
457

```xml
<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>
```
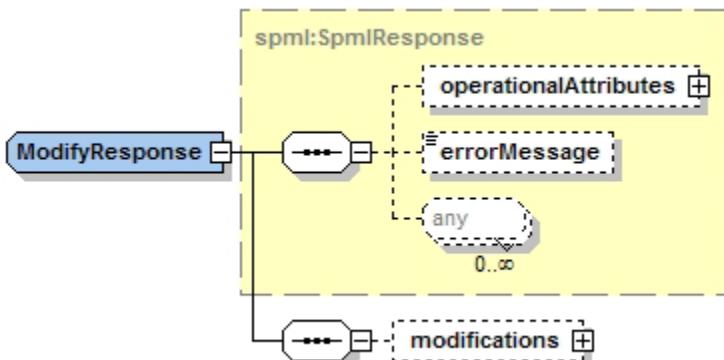
3.3.2 SPML Modify Operation

459

460

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**".

467

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

468

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" />
```
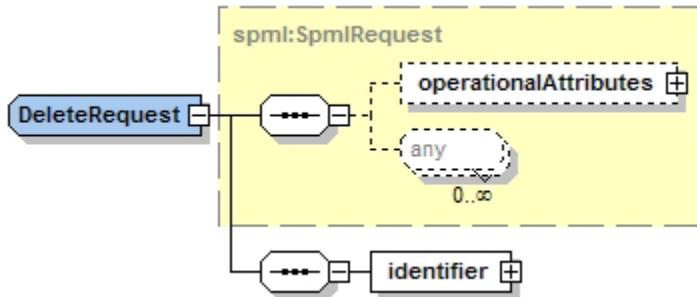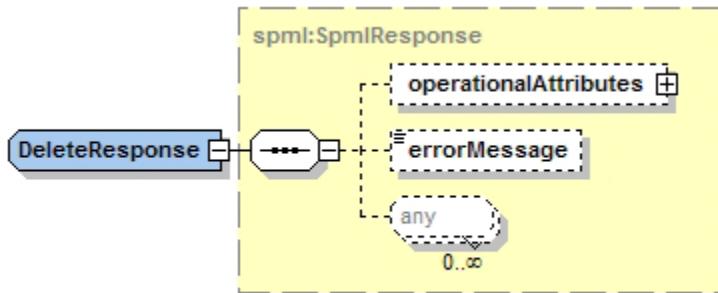
471

472   ### 3.3.3 SPML Delete Operation

473

474

475   The *<DeleteRequest>* element is used by a RA to request the deletion of a specific PSTD-ID.
476   Implicit in the deletion of a service object is the identification of the exact PSTD-ID to be deleted. In
477   this example, the RA specifies an *<IdentifierType>* of EmailAddress and uniquely identifies the
478   PSTD via the *<id>* of Jane.Doe@acme.com.
479

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

480
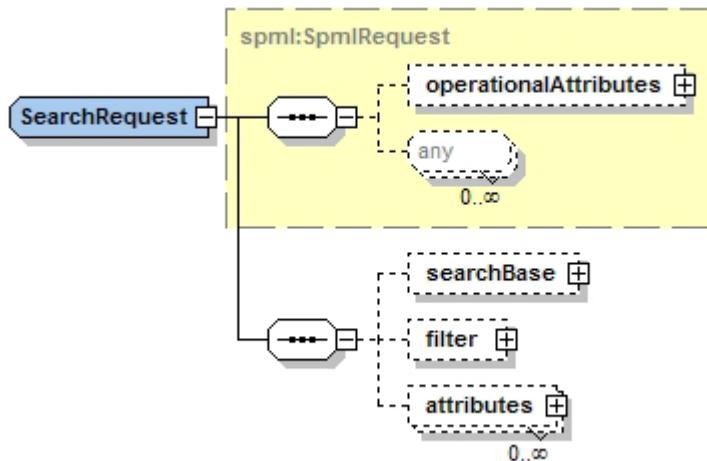
481   Using the *<DeleteResponse>* element, the functioning PSP returns the status of the
482   *<DeleteRequest>.*
483

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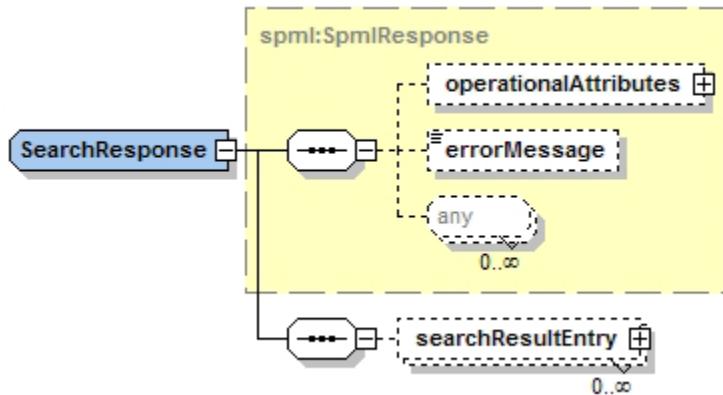
3.3.4 SPML Search Operations

485

486

487   The <*SearchRequest*> element is used to allow a SPML V1.0 client to request a search be
488   performed on its behalf by a PSP. A search is used to read attributes from service objects
489   managed by the PSP.   A search request specifies a <*searchBase*> as the identifiable start point for
490   a search operation, a search criteria <*filter*> and the attributes to be returned <*attributes*>. Note
491   you can not specify what values can be returned. If no <*attributes*> are specified on the search
492   request, all attributes are returned in the corresponding <*SearchResponse*>. In the following
493   example, the RA requests a search for all accounts in the acme.com domain that have a "**cn**"
494   attribute that ends in "**Doe**". The <*filter*> element defines the search criteria. This sample request
495   also defines the list of data values it wants returned in the <*SearchResponse*>. Using the
496   <*attributes*> sub element the client constrains the target schema attributes returned in the results to
497   "**cn**" and "**email**":
498

```
<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>
```

499
500   The results of the search request are returned to the RA using the <SearchResponse> element.
501   Search responses include an "**result**" attribute and a series of *<SearchResultEntry>* elements.
502   Each entry returned in a *<SearchResultEntry>* will contain all requested attributes, complete with
503   associated values, as specified in the attributes field of the Search Request. The specific return of
504   attributes is subject to access control and administrative policy defined by the PSP.

505   The following example shows the result of the search requested above. Note that the "**result**"
506   defines the successful completion of the request and that each *<SearchResultEntry>* includes an
507   *<identifier>* with its corresponding *<IdentifierType>* to uniquely identify each record. Following this
508   identifier is the list of *<attributes>* requested in the search request. The ordering of the returned
509   attributes corresponds to the order in which they are requested in the *<attributes>* element in the
510   search request.
511

```
<searchResponse result = "urn:oasis:names:tc:SPML:1:0#success">
    <searchResultEntry>
        <spml: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>
```

512   ### 3.3.5 Operational Attributes

513   SPML V1.0 provides a convenient way for requestors and responders to provide additional
514   operational attributes when issuing and responding to SPML requests. The *<SpmlRequest>* and
515   *<SpmlResponse>* elements both provide an *<operationalAttributes>* sequence element that allows
516   either functioning party to specify additional information pertinent to the execution of a given
517   operation. In the example below, the functioning RA issues the delete request as defined in section
518   7.3.3 or this document. Notice that in this example, the RA adds *<operationalAttributes>* to the
519   delete request specifying an archival policy for the deleted mail box and a time at which the request
520   should be executed.
521

```
<deleteRequest>
        <spml: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

An 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 the 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

547 sample deletion response that shows a syntax error in the request.
548
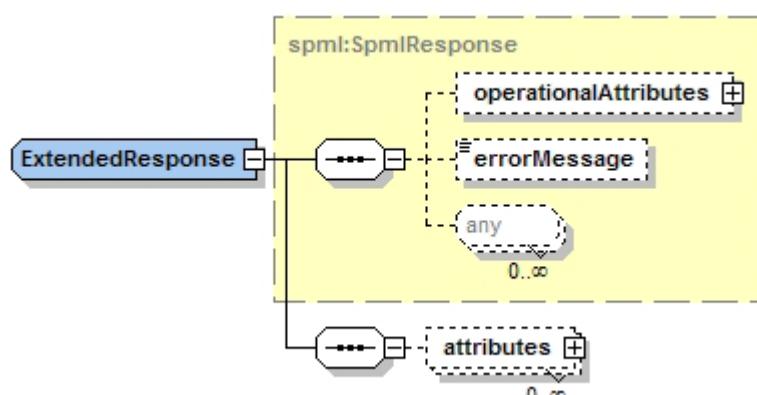
<deleteResponse result = "urn:oasis:names:tc:SPML:1:0#malformedRequest" />

549
550 To further enable implementation specific error handling SPML V1.0 provides the
551 "**ErrorCode**" value of "**urn:oasis:names:tc:SPML:1:0#customError**". A PSP may
552 use this error code in conjunction with <*operationalAttributes*> from the
553 <*SpmlResponse*> element to report implementation specific error codes.
554
555

3.3.7 SPML Extended Operations

556

557

558
559 An extension mechanism has been added to the SPML v1.0 specification to allow additional
560 operations to be defined for services not available elsewhere in this protocol. The extended
561 operation allows clients to make requests and receive responses with predefined syntaxes and
562 semantics. Extended request operations may be defined by other standards bodies or may be
563 private to particular vendor implementations. A RA uses the <*ExtendedRequest*> element to
564 request an extended SPML operation.
565
566 Each new extended request operation defines a unique <*operationIdentifier*> that names and
567 identified the new operation. In the example below, the PSP has provided an extended operation
568 "**urn:acme.com.mailservice.ops:purge**" that allows its clients to request the purging of a defined

569 mail box. In this implementation no service schema attributes are required when requesting this
570 operation. The RA simply provides the unique *<identifier>* for the PSTD-ID it wished to operate
571 against.
572

```
<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>
```

573
574 Using the *<ExtendedResponse>* element, the functioning PSP returns the status of the
575 *<ExtendedRequest>* as shown below.
576

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

577
578 SPML V1.0 also defines a mandatory *<providerIdentifier>* sub-element that allows an extended
579 request provider to annotate their new operations and provide an aggregation mechanism such that
580 a set of extended requests can be correlated back to a specific provider. In the examples below,
581 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
582 operations ***purge*** and ***compress***.
583
584 For a full explanation of *<providerIdentifier>* and *<operationIdenfifier>* elements see section 3.4 of
585 this document.
586

```
<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>
```

587

588   3.4   SPML Identifiers

589   3.4.1 Target Identifiers

590 In the SPML V1.0 each of operations Add, Modify, Delete and Extended requests require the RA to
591 specify a unique *<Identifier>* element which defines the PSTD-ID the requested operation applies.
592 This is used to directly identify the PSTD to be operated against. The *<IdentifierType>* type defines
593 the allowable types for an <Identifier>. The following types are supported:
594

```
<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>
```

595
596   Using one of these identifiers the RA issues an SPML operational request directed at a specific
597   PSTD-ID. In the example below, the RA is requesting the deletion of a mail box identified by the
598   <*Identifier*> of Jane.Doe@acme.com. Here the ***urn:oasis:names:tc:SPML:1:0#EMailAddress***
599   type is used as the target identifier type:

600

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

601
602   More complex identifiers can be specified using the other <*IdentifierTypes*>. For example, if an RA
603   wishes to subscribe to a published SPML service and wishes to use a SAML subject statement as
604   it's identification for the service object.
605

606   3.4.2 Provider & Operation Identifiers

607   SPML <*ExtendedRequest*> operations and the service schema definition model both implement a
608   <*providerIdentifier*> element. The <*providerIdentifier*> element allows a PSP to annotate new
609   operations and published services such that a set of extended requests or service schema
610   elements can be correlated back to a specific provider. The <*providerIdentifier*> element is used in
611   conjunction with <*operationIdentifier*> element to control the globally-unique identification of
612   extended requests. The <*providerIdentifier*> element is used in conjunction with the
613   <*schemaIdentifier*> element to control the globally-unique identification of service schema
614   definitions.
615
616   For information about the <*schemaIdentifier*> elements see section 3.6.2 in this document.

617
618

619   3.5   Request / Response Model Overview

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

625 the use of batch and non-batch operation and the use and semantics of the differing operational
626 models supported by this protocol.

3.5.1 Execution Types

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

3.5.2 Request Status

638 When SPML operations are being executed asynchronously, the RA needs a way to query the
639 status of requests. SPML V1.0 provides the *<StatusRequest>* operation to allow clients to request
640 processing status from the corresponding service. The example below shows a RA request status
641 and corresponding server response. Note the status response includes an explicit typed "**result***"*
642 attribute.
643

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

644

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

645
646 SPML V1.0 provides a fixed set of result codes for *<StatusResponse>* elements; these are defined
647 by the *<StatusReturnsType>* element shown below:
648

```
<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>
```

649

650 In certain circumstances it is desirable for an RA to be able to query not only the status of an
651 asynchronous request but to also get the current request results set. SPML V1.0 provides this
652 capability via the "**statusReturns"** attribute on the *<StatusRequest>* element. In the following
653 example, a RA has issued an asynchronous *<SearchRequest>*. The same RA then issues an
654 *<StatusRequest>* with "**statusReturns=urn:oasis:names:tc:SPML:1:0#result***".* The resulting
655 *<StatusResponse>* element is then returned with the currently processed search results.
656

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

657

```
<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>
```

658   3.5.3 Cancel Request

659 When SPML operations are being executed asynchronously, the RA also needs a way to cancel a
660 pending request. SPML V1.0 provides the *<CancelRequest>* operation to allow clients to request
661 the cancellation of a requested batch from the corresponding service. The example below shows
662 an RA requesting the cancellation of a previously requested operation, followed by the resulting
663 response from the PSP. Note the cancellation response includes an explicit typed
664 "**CancelResults**" attribute.
665

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

666

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

667
668 SPML V1.0 provides a fixed set of result codes for *<CancelResponse>* elements; these are defined
669 by the *<CancelResultsType>* element shown below:
670

```
<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>
```

671

672   3.5.4 Singleton Requests

673 Each of the SPML core operations implicitly ties together a request and a response. Individual
674 requests can be constructed by an RA and bound to the chosen protocol as defined in the SPML
675 bindings document [**SPML-Bind**] and dispatched to a given PSP. A singleton request follows the
676 basic *"execution"* model defined in section 7.5.1 of this document.   The default execution mode
677 for all singleton requests is synchronous. The following example shows a basic singleton
678 *<AddRequest>*:
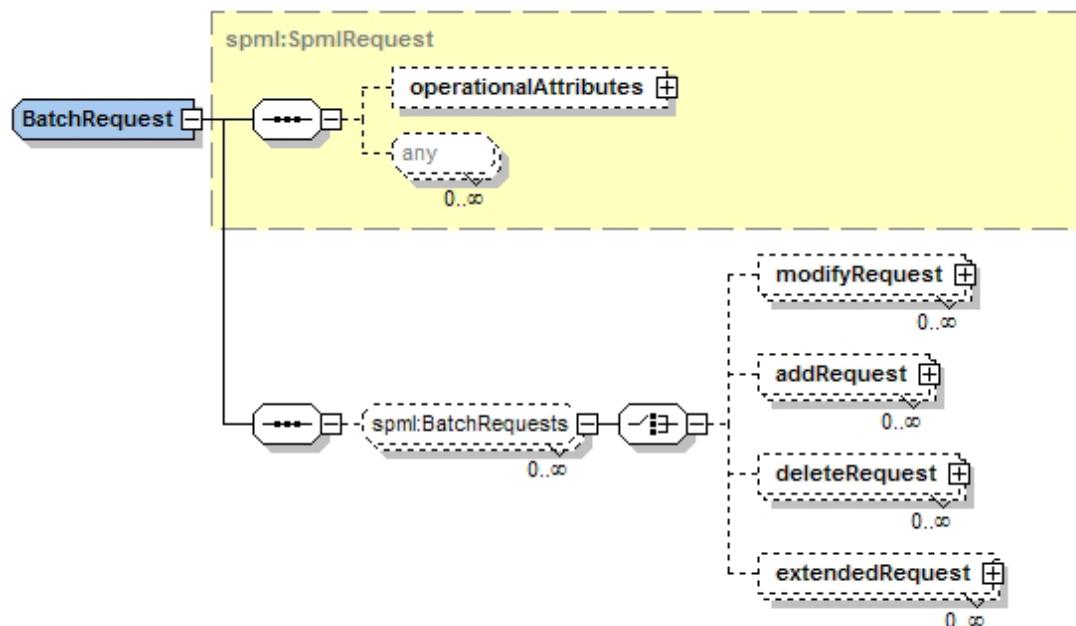
```
<addRequest requestID="Request-92.168.1.1-992A1">
   <spml:attributes>
      <attr name="objectclass"> <value>emailUser</value></attr>
      <attr name="cn"> <value>Jane Doe</value></attr>
      <attr name="gn"><value>Jane</value></attr>
      <attr name="sn"><value>Doe</value></attr>
   </attributes>
</addRequest>
```

679

680 ### 3.5.5 Batch Requests

681

spml:SpmlResponse

BatchResponse

operationalAttributes

errorMessage

any

0..∞

spml:BatchResponses

0..∞

modifyResponse

0..∞

addResponse

0..∞

deleteResponse

0..∞

extendedResponse

0..∞
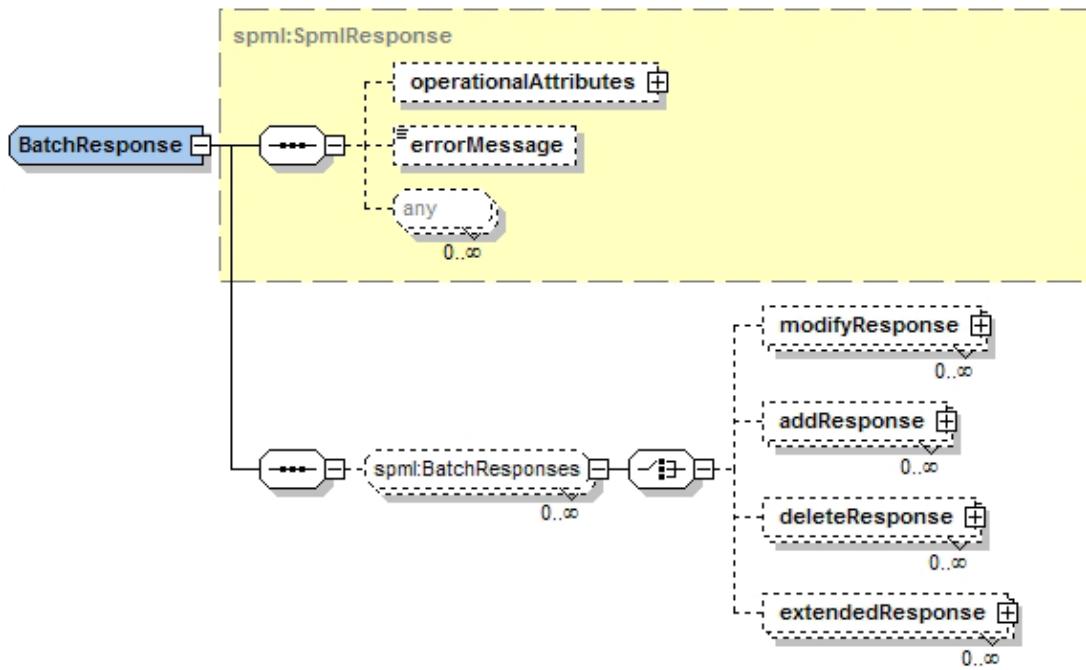
682

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

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

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

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

700   ### 3.5.5.1 Processing Types

701   Batch requests support two distinct processing types *sequential* and *parallel*. When an RA
702   constructs a batch request it uses the *"***processing***"* attribute to instruct the PSP of the processing
703   order of the individual operation in that batch. Sequentially processed batch operations are
704   executed by the PSP in the exact order they are defined in the *<B*atchRequest>. Parallel batch
705   operations may be executed by the PSP in any order. In both cases, the *<BatchResponse>*
706   maintains positional correspondence in the return of individual operation response elements. In the
707   below example the RA request the asynchronous execution of parallel batch of two add requests.
708   Note the mandatory use of the **requestID** attribute on asynchronous batches (see 3.5.2.3 below):
709

```
<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

710

711   SPML V1.0 batches support two basic error handling modes "*resume*" and "*exit*". The
712   <BatchRequest> base element provides an *onError* attribute to control this behavior. When an RA
713   issues a batch request with "*onError=resume*" errors encountered processing individual operations
714   within that batch are handles by the PSP and do not effect the attempted execution of the
715   remaining operations in the batch. It is the responsibility of the PSP to maintain the positional
716   correspondence of the individual operations and provide appropriate error reporting as described in
717   section 3.3.6.
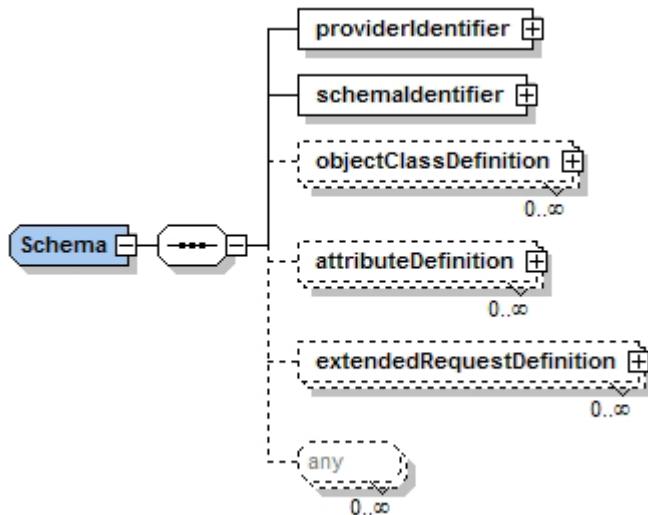
718   When an RA issues a batch request with *onError="exit",* an error encountered processing an
719   individual operation within that batch results in the termination of processing for the entire batch and
720   all of the requests that did not get processes are marked as failed. When used with the **processing**
721   attribute, **onError** provides the RA with the ability to guarantee execution order and pre-requisite
722   processing in batch operations.

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

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

3.6   SPML Schema Overview

732

733

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

739 Each of the SPML V1.0 core operational requests represent an action against a given service
740 schema. In each operation the service schema is defined by the special attribute "**objectclass**" as
741 shown below. The **objectclass** attribute is a direct reference to a known SPML V1.0
742 *<objectclassDefinition>* element. In the example below, an *<AddRequest>* operation is requesting
743 the creation of an object of the pre defined object class "**urn:oasis:names:tc:SPML:standard**".
744

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

745
746 The SPML service schema model provides the *<schema>* element to contain the definition of the
747 service schema and a simple request/response protocol for exchanging schema definitions
748 between SPML actors.

3.6.1 The Schema Element

750 The SPML V1.0 service schema is represented by the *<schema>* element. This element is used to
751 define the reusable definition of an SPML service as a uniquely identifiable set of XML Schema
752 typed data attributes and object class definitions with corresponding attributes and properties. The
753 following example shows an SPML schema definition that defines a simple service schema object
754 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>
```

755
756   The "**standard**" schema defined above is uniquely identified by the combination of the globally
757   unique "**urn:oasis:names:tc:SPML:1:0#URN**" *<providerIdentifier>* element and a locally unique
758   "**urn:oasis:names:tc:SPML:1:0#GenericString**" *<schemaIdentifier>* element. This schema then
759   defines three attributes that may be used in locally defined object class definitions or reference from
760   external schema. The *<objectclassDefinition>* brings all this together in a single definition of the
761   services schema named "**person**". The person service object class includes a text description
762   attribute and in this simple example, references the attributes defined in the local schema element
763   itself.

764   ## 3.6.2 Schema Qualification

765   One of the requirements of SPML is to provide a model for the definition and exchange of service
766   schema and in doing so to enable the definition and re-use of both entire schema definitions AND
767   individual schema attributes. Vital to this goal is the judicious and complicit use of scoped and
768   qualified names for defined object class definitions and contained attributes. In the SPML V1.0
769   model, attribute namespace is controlled by the use of a URN based schema identification model
770   that reuses the core SPML *<providerIdentifier>* and an additional *<schemaIdentifier>* element.
771
772   The *<providerIdentifier>* element is described in section 3.4.2 of this document. The
773   *<schemaIdentifier>* is used in conjunction with th,e *<providerIdentifier>* element to provide a unique
774   name for a given service schema. The following *<schemaIdentifier>* types are supported by SPML
775   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>
```

776

777 The following example the *GenericString* *<schemaIdentifier>* of "**GoldEmailService**" and the
778 unique **OID** of "**1.3.6.1.4.868.2.4.1.2.1.1.1.3.3562**" are used to create a globally unique identifier for
779 a service schema.
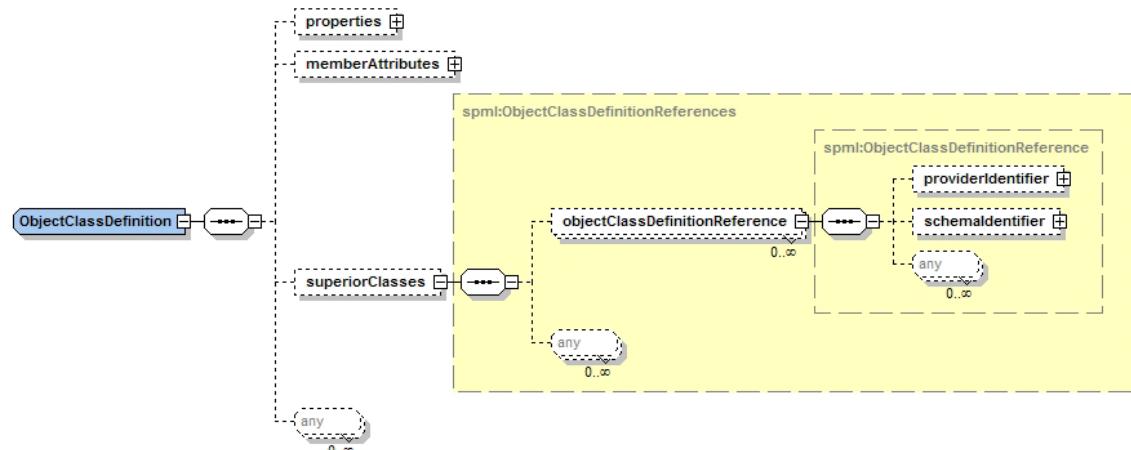
```
<schema …>
    ……..
    <providerIdentifier providerIDType="urn:oasis:names:tc:SPML:1:0#OID">
        <providerID>1.3.6.1.4.868.2.4.1.2.1.1.1.3.3562</providerID>
    </providerIdentifier>

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

780 ### 3.6.3 Service Object Class Definition

781

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

791 The *<objectclassDefinition>* *<superiorClasses>* sub element implements an attribute inheritance
792 model for SPML V1.0 service schema. By defining a *<superiorClasses>* element a given schema
793 can automatically inherit the set of *<objectclassDefinition>* elements from another SPML schema.

794 In the first example below the SPML service schema defines a "**standard**" service schema which is
795 then referenced in the second example to create the new SPML service schema "**interopUser**".
796

```xml
<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>
```

797

```xml
<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>
```

798 3.6.4 Attributes, Typing & Referencing

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

```xml
<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>
```

803
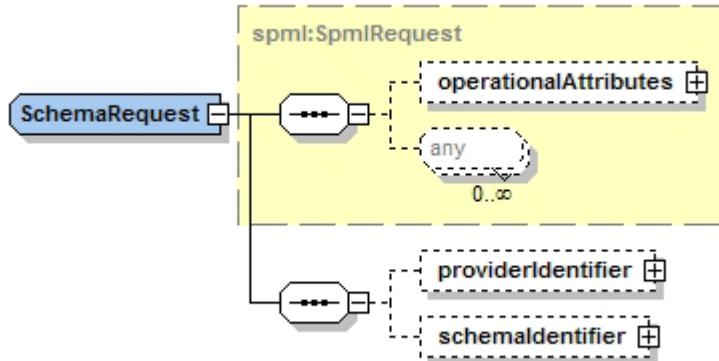
3.7 Schema Operations

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

3.7.1 Schema Request

811

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

816 In the following example an RA requests the retrieval of the provisioning schema defined in section
817 3.6.3 above **urn:oasis:names:tc:SPML:interop.**

818
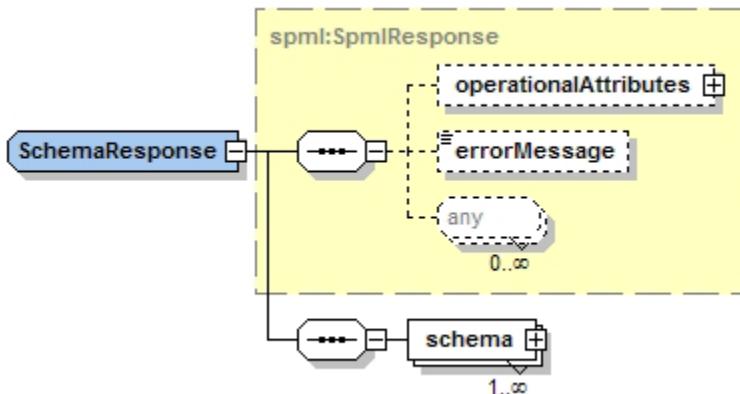
```
<schemaRequest requestID="REQ1.4.5.6.AKS"
                execution="urn:oasis:names:tc:SPML:1:0#synchronous">

    <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>
```

819

3.7.2 Schema Response

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

824

```
<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>
```

825

4 Examples (non-normative)
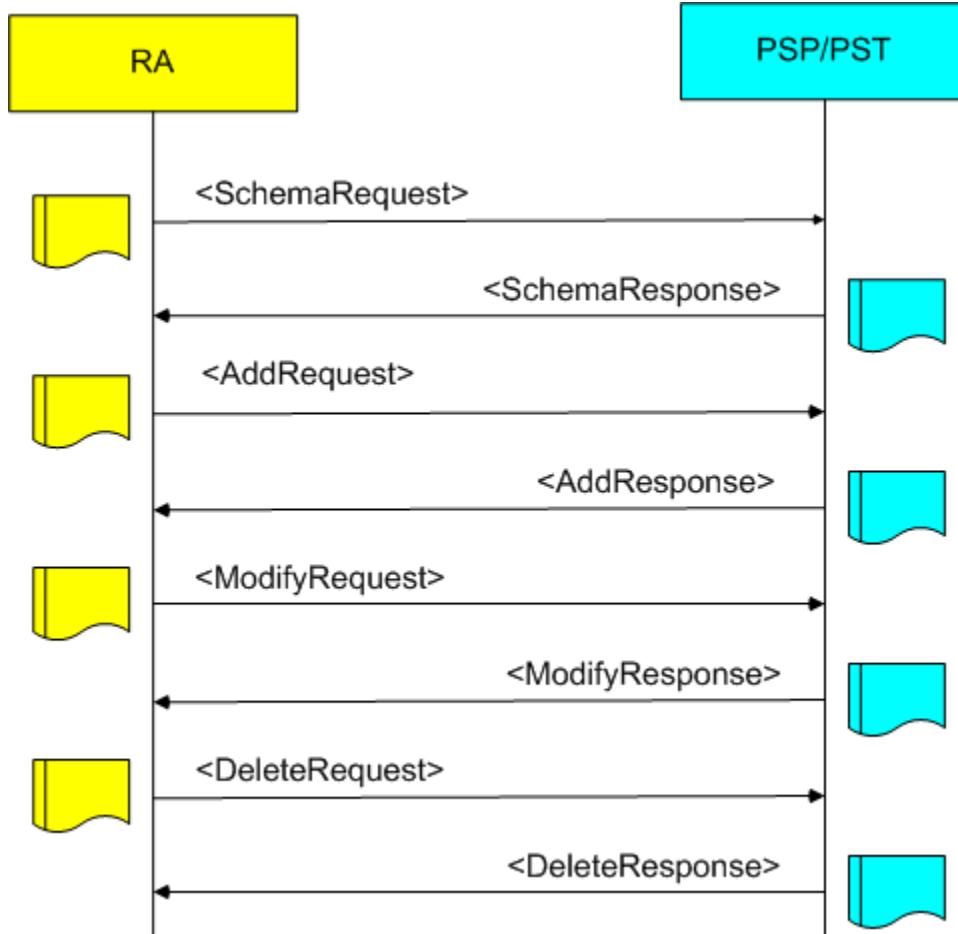
827   The following non-normative section provides two examples that build of the SPML use cases
828   [**PSTC-UC**]. Example 1 describes a simple synchronous flow in which the Requesting Authority

829 asks for the creation of a new service object. Example 2 describes a more complex asynchronous
830 request for a batch operation with subsequent status query.

831 ## 4.1 Example One

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

836

837

838 ### 4.1.1 <SchemaRequest>

839 First the RA makes a synchronous request for the return of a given sevice schema. From the
840 returned schema definition the RA will build an *<AddRequest>*. Note the RA is looking for the
841 interop schema from the **urn:oasis:names:tc:SPML** provder:
842

```
<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>
```

843

844 4.1.2 <SchemaResponse>

845 The PSP successfully returns the requested schema:

846

```
<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>
```

847

848 4.1.3 <AddRequest>

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

851

```
<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>
```

852

4.1.4 <AddResponse>

854 The PSP returns an *<AddResponse>* and includes an additional *<attributes>* elelement that
855 includes informational data on the fulfillment of the request:
856

```
<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>
```

857

4.1.5 <ModifyRequest>

859 Using <identifier> elelemt to uniquely identify the PSTD the RA issues a *<ModifyRequest>* to
860 change the subscribers **memberLevel** attribute to value "**2**":
861

```
<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>
```

862

863   ## 4.1.6 <ModifyResponse>

864   The PSP responds with a *<ModifyResponse>* with the **result** attribte set to
865   **urn:oasis:names:tc:SPML:1:0#success** indicating that the request was successfully executed:
866

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

867

868   ## 4.1.7 <DeleteRequest>

869   Finally the RA issues a *<DeleteRequest>* to remove the PSTD-ID from the service:
870

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

871

872   ## 4.1.8 <DeleteResponse>

873   The PSP responds with a *<DeleteResponse>* with a **result** attribute of
874   **urn:oasis:names:tc:SPML:1:0#success** to indicate that the specified PSTD was removed:
875

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

876

877   ## 4.2   Example Two

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

883

885

4.2.1 <SchemaRequest>

886

887 First the RA makes a synchronous request for the return of a given sevice schema. From the
888 returned schema definition the RA will build an *<AddRequest>*. Note the RA is looking for the
889 interop schema from the **urn:oasis:names:tc:SPML** provder:
890
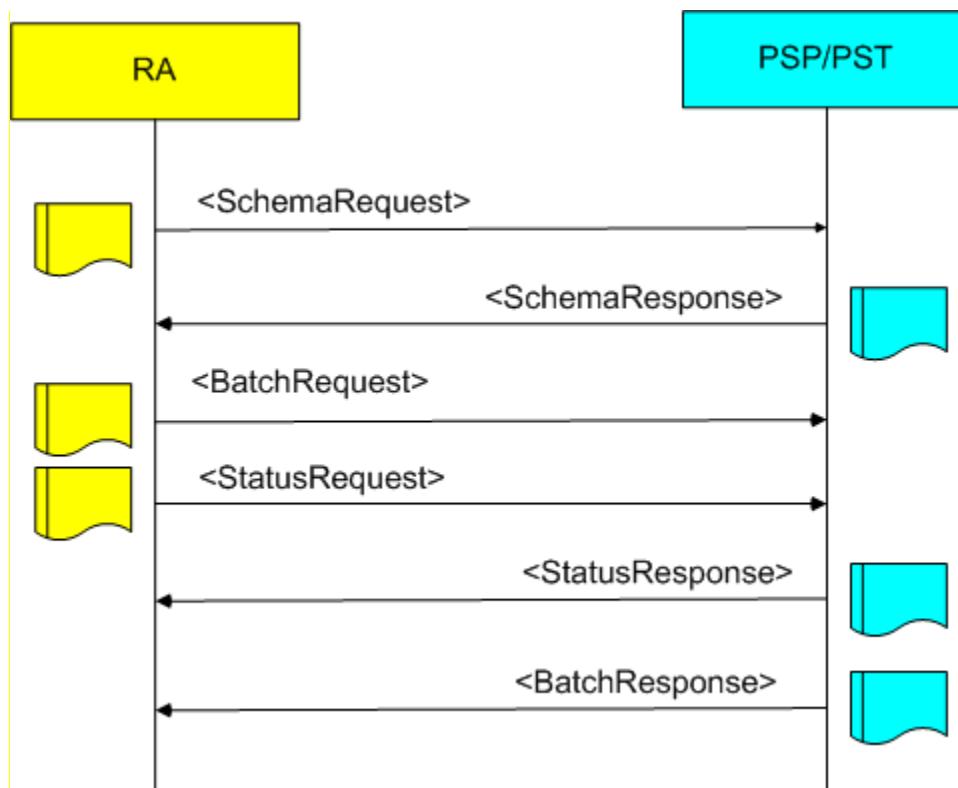
```
<schemaRequest = requestID="REQ1.4.5.6.AKS"
                execution="urn:oasis:names:tc:SPML:1:0#synchronous">

   <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>
```

891

4.2.2 <SchemaResponse>

892

893 The PSP successfully returns the requested schema:
894

```
<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>
```

895

4.2.3 <BatchRequest>

897 Based on the schema definition returned, the RA requests an asynchronous batch containing two
898 *<AddRequests>* elements both of which are given unique *<requested>* identifiers:
899

```
<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>
```

900

4.2.4 <StatusRequest>

902 Having issued the asynchronous batch request, the RA then decides to query the PSP for the
903 status of the request using the *<StatusRequest>* operation:
904

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

905

4.2.5 <StatusResponse>

907 In response to the *<StatusRequest>* the PSP returns the following *<StatusReposnse>*. Note the
908 **Result** attribute for the batch is set to **urn:oasis:names:tc:SPML:1:0#pending** to indicate that at
909 least one of the request within that batch has not yet completed.
910

```
<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>
```

911

4.2.6 <BatchResponse>

913 Finally the PSP completes the batch and issues a *<BatchResponse>*. Note the batch **Result**
914 attribute is now set to **urn:oasis:names:tc:SPML:1:0#failure** as the second request in the batch
915 failed:
916

```
<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>
```

917

918 5 SPML Operations (normative, with the exception
919   of the schema fragments)

920 5.1 Schema Header and Namespace Declarations

921 The following schema fragment defines the XML namespaces and header information for the SPML
922 schema.
923

```
<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>
```

924

925 5.2 Complex Type Identifier

926 The Identifier type is used to specify either a PSO-ID or PSTD-ID the semantics of the identifier are
927 determined from context. It is the type of both the *<identifier>* and *<searchBase>* elements. It
928 MUST contain a "**type**" attribute, MUST contain either an *<id>* or *<subject>* element, and MAY
929 contain any number of *<identifierAttributes>* elements.
930

| 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

931

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

934

```
<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>
```

935 **5.3   Simple Type IdentifierType**

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

940 IdentifierType is an enumeration of the following values:
941

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

942   5.4   Element <Identifier>

943   The *<identifier>* element is of type "**Identifier**" and is used to specify a PSO-ID or PST-ID in a
944   number of request and response elements. The following schema fragment is used to define the
945   *<identifier>*.
946

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

947   5.5   Element <AddRequest>

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

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

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

957

```
<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>
```

958   5.6   Element <AddResponse>

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

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

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

968 The following schema fragment defines the *<AddResponse>* element and its AddResponse
969 complex type:
970

```
<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>
```

971

972   5.7   Element <ModifyRequest>

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

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

979 The following schema fragment defines the *<ModifyRequest>* element and the ModifiRequest
980 complex type:
981

```
<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>
```

982

983   5.8   Element <ModifyResponse>

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

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

989   The following schema fragment defines the *<ModifyResponse>* element and its ModifyResponse
990   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>
```

991

992   5.10 Element <DeleteRequest>

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

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

997   The following schema fragment defines the *<DeleteRequest>* element and its DeleteRequest
998   complex type:
999

```
<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>
```

1000

1001 5.10 Element <DeleteRespose>

1002 The *<DeleteResponse>* element is used to convey the results of a request specified with the
1003 <DeleteRequest> element. It is of type DeleteResponse which extends SpmlResponse. It does
1004 not specify any additional elements or attributes beyond those specified by SpmlResponse.

1005 The following schema fragment defines the *<DeleteResponse>* element and its DeleteResponse
1006 complex type:
1007

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

1008

1009 5.11 Element <SearchRequest>

1010 The *<SearchRequest>* element is used to request the retrieval of attributes of existing service
1011 objects. The element is of type SearchRequest which extends type SpmlRequest.

1012

1013 The element MAY contain a *<searchBase>* element which the PSP may use to constrain the
1014 search. The *<searchBase>* element is of complex type Identifier. The semantics of the
1015 *<searchBase>* element are defined by the PSP.

1016 The element MAY contain a *<filter>* element which is used to restrict the search to objects whose
1017 attributes adhere to specified criteria. The *<filter>* element is of type dsml:Filter defined by DSML.

1018 The element MAY contain an <attributes> element which is used to specify which attributes of the
1019 objects matching the search are to be retrieved. The element is of type dsml:AttributeDescriptions
1020 defined by DSML [**DSML**]

1021 The following schema fragment defines the *<SearchRequest>* element and its SearchRequest
1022 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>
```

1023

5.12 Element <SearchResponse>

1025 The <*SearchResponse*> element is used to convey the results of a request specified with the
1026 <SearchRequest> element. It is of type SearchResponse which extends SpmlResponse.

1027 The element MAY contain any number of <*SearchResultEntry*> elements which contain information
1028 about the service objects that matched the search criteria.

1029 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>
```

1030

5.13 Element <SearchResultEntry>

1032 Element <*SearchResultEntry*> found within element <*SearchResponse*> are of type
1033 SearchResultEntry. This type extends dsml:DsmlMessage which is defined by DSML. Beyond the
1034 attributes and elements specified DsmlMessage, elements of this type MAY include one <*identifier*>
1035 element and MAY include one <*attributes*> element.

1036 The <*identifier*> element is used to convey the identity of a service object that matched the search
1037 criteria.

1038 The <*attributes*> element is used to convey the names and values of attributes of the matching
1039 service object.

1040 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>
```

1041

5.14 Element <ExtendedRequest>

1042

1043 The *<ExtendedRequest>* element is used to request services provided by a PSP that are not
1044 specifically defined by SPML. The semantics of an extended request are defined solely by the
1045 PSP. The element is of type ExtendedRequest which extends SpmlRequest.

1046 The element MUST contain an *<operationIdentifier>* element that specifies the service being
1047 requested.

1048 The element MUST contain a *<providerIdentifier>* element that further identifies the service. If the
1049 *<ExtendedRequestDefinition>* that defines an extended request specifies both an
1050 *<operationIdentifier>* and a *<providerIdentifier>* then the *<providerIdentifier>* element MUST be
1051 included in the *<ExtendedRequest>*.

1052 The element MAY contain an *<identifier>* element that identifies a particular service object to be
1053 associated with the request.

1054 The element MAY contain an <attributes> element that in turn may contain any number of *<attr>*
1055 elements. These are used to convey arbitrary information specific to the operation.

1056 The following schema fragment defines the *<ExtendedRequest>* element and the
1057 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>
```

1058

5.15 Element <ExtendedResposne>

1059

1060 The *<ExtendedResponse >* element is used to convey the result of a request specified with the
1061 *<ExtendedRequest>* element. It is of type ExtendedResponse which extends SpmlResponse.

1062 It MAY include one *<attributes>* element which may include any number of <attr> elements.

1063 The following schema fragment defines the *<ExtendedResponse >* element and the
1064 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>
```

1065   5.16   Element <providerIdentifier>

1066   The providerIdentifier element allows an SPML service to annotate new operations and published
1067   services such that a set of extended requests or service schema elements can be correlated back
1068   to a specific provider. The element MUST contain a *<providerID>* element. The content of
1069   *<providerID>* is determined by "providerIDType" attribute.

1070   The element MUST contain a "providerIDType" attribute which MUST be one of the following
1071   values:

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

1072

1073   The following schema fragment defines the *<providerIdentifier>* element and ProviderIdentifier type:
1074

```
<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>
```

1075   5.17   Element <operationIdentifier>

1076   The *<operationIdentifier>* element is used to identify a service that may be requested in an
1077   *<ExtendedRequest>*.

1078   The element MUST contain an "operationIDType" attribute that specifies the type of the identifier. It
1079   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.

1080
1081 The element MUST contain an *<operationID>* element whose content is determined by the value of
1082 the "operationIDType" attribute.

1083 The following schema fragment defines the *<operationIdentifier>* element and OperationIdentifier
1084 type:
1085

```
<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>
```

1086

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

6.1 Complex Type SpmlRequest

1090 The complex type SpmlRequest specifies information common to all SPML requests. It includes
1091 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

1092

1093   The following schema fragment defines the SpmlRequest, ExecutionType, and
1094   <*operationalAttributes*> elements:
1095

```
<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>
```

1096   6.2   Simple Type ExecutionType

1097   Type ExecutionType is used to specify how a PSP is expected to respond to an SPML request. It
1098   is an enumeration of the following values:
1099

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.

1100

1101   The following schema fragment defines complex type SpmlResponse and related simple types:
1102

```
<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>
```

1103

1104   6.3   Simple type Result

1105   The type Result defines the allowed values for the "result" attribute of type SpmlResponse.
1106

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.

1107   6.4   Simple type ErrorCode

1108   The type ErrorCode is used to convey more detailed information about a request failure. It is an
1109   enumeration of the following values:
1110

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

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

1111

6.5   Element <BatchRequest>

1112

1113 The *<batchRequest>* element is used to specify a collection of related SPML requests to be
1114 executed. It is of type BatchRequest which extends SpmlRequest. It defines the following
1115 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**".

1116
1117 The *<batchRequest>* element may contain any number of *<AddRequest>*, *<ModifyRequest>*,
1118 *<DeleteRequest>*, and *<ExtendedRequest>* elements in any order.

1119 The following schema fragment defines the *<batchRequest>* element and related types:
1120

```
<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>
```

1121

1122 6.6   Simple Type ProcessingType

1123 Type ProcessingType defines the allowed values for the "**processing**" attribute of element
1124 <*batchRequest*>. It is an enumeration of the following values:
1125

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.

1126

1127 6.7   Simple Type OnErrorType

1128 Type OnErrorType defines the allowed values for the "**onError**" attribute of element
1129 <*batchRequest*>. It is an enumeration of the following values:
1130

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.

1131

1132 6.8 Element <BatchResponse>

1133 The *<batchResponse>* element is used to convey the result of a request specified with the
1134 *<batchRequest>* element. It is of type BatchResponse which extends SpmlResponse. The
1135 element may contain any number of *<AddResponse>*, *<ModifyResponse>*, *<DeleteResponse>*,
1136 and *<ExtendedResponse >* elements.

1137 The PSP MUST return one response element for each request element within the *<batchRequest>*.
1138 The PSP MUST return response elements in the same order as the corresponding request
1139 elements. The PSP Must return response elements whose types match the corresponding request
1140 element, for example an *<AddResponse>* must be returned for an *<AddRequest>*.

1141 The number and order of response elements is unaffected by "sequential" or "parallel" processing,
1142 or by errors in individual requests.

1143 If an error occurs in one request, and the *<batchRequest>* specifies an "**onError**" attribute value of
1144 "**exit**", response elements for all unprocessed requests will be returned with "**result**" attributes set
1145 to "**error**".

1146 The following schema fragments define the *<batchResponse>* element and related types:
1147

```
<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>
```

1148

1149 6.9 Element <StatusRequest>

1150 The *<StatusRequest>* element is used to retrieve the status of a request previously submitted to a
1151 PSP using an execution type of "asynchronous". It is of type StatusRequest which extends

1152   SpmlRequest. It defines the following attributes:
1153

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

1154   The <*StatusRequest*> element MUST NOT specify an "**execution**" attribute value of
1155   "**synchronous**".
1156

1157   The following schema fragments define the <*StatusRequest*> element and related types:
1158

```
<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>
```

1159

1160   6.10 Element <StatusResponse>

1161   The <*StatusResponse*> element is used to convey the result of a request specified with the
1162   <*StatusRequest*> element. It is of type StatusResponse which extends SpmlResponse.

1163
1164   The element MUST contain a "statusResult" attribute which will have one of the following values:
1165

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

1166

1167   If the "statusReturns" attribute in the corresponding <StatusRequest> element was set to "results",
1168   the <StatusResponse> element MUST contain one of the response elements <AddResponse>,
1169   <ModifyResponse>, <DeleteResponse>, <SearchResponse>, <ExtendedResponse >,
1170   <SchemaResponse>, or <batchResponse>.
1171
1172   If the "statusReturns" attribute in the corresponding <StatusRequest> element was set to "status"
1173   the <StatusResponse> element MUST NOT contain any response elements.
1174
1175   If a response element is returned, it MUST match the corresponding request element, for example
1176   an <AddResponse> must be returned for an <AddRequest>.
1177
1178   The following schema fragment defines the <StatusResponse> element and related types:
1179

```xml
<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>
```

1180

1181   **6.12 Element <CancelRequest>**

1182   The <cancelRequest> element is used to terminate a request previously submitted to a PSP using
1183   an execution type of "asynchronous". It is of type StatusRequest which extends SpmlRequest. It
1184   defines the following attributes:
1185

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 applies not to this request but to another request.

1186

1187 The following schema fragment defines the <cancelRequest> element
1188 and related types:
1189

```
 <xsd:element name="cancelRequest" type="spml:CancelRequest" />
<xsd:complexType name="CancelRequest">
 <xsd:complexContent>
   <xsd:extension base="spml:SpmlRequest">
   </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

1190

1191 6.12 Element <CancelResponse>

1192 The <cancelResponse> element is used to convey the result of a request specififed with the
1193 <cancelRequest> element. It is of type StatusResponse which extends SpmlResponse.
1194
1195 The element MUST contain a "cancelResults" attributes with one of the following values:
1196
1197

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.

1198
1199

```
<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>
```

1200

1201 7 SPML Provisioning Schema (normative, with the
1202 exception of the schema fragments)

1203 This section documents the SPML elements related to the specification of object schemas by the
1204 PSP.

1205 SPML Servers MAY treat attribute names as case-sensitive. SPML Servers MAY NOT use case
1206 sensitivity to distinguish attributes. An SPML client SHOULD use the attribute names in the same
1207 case as defined in the SPML Schema for the SPML service.

7.2   Element <schemRequest>

1208

1209 The <SchemaRequest> element is used to request the descriptions of one or more schemas
1210 supported by a PSP. The element is of type SchemaRequest which extends SpmlRequest.

1211 The element MAY contain a <providerIdentifier> element and MAY contain a <schemaIdentifier>.
1212 The content of both these identifiers is defined by the PSP.

1213 If the request has neither <providerIdentifier> or <schemaIdentifier> then all schemas accessible
1214 through the PSP are requested.

1215 If the request has a <providerIdentifier> but no <schemaIdentifier>, then all schemas associated
1216 with the given provider identifier are requested.

1217 If the request has a <schemaIdentifier> but no <providerIdentifier>, then the <schemaIdentifier>
1218 must be specified as a URN that unambiguously identifies one schema.

1219 The following schema fragment defines the <schemaRequest> element and related types:
1220

```xml
<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>

1221

1222 The <schemaResponse> element is used to convey the result of a request specified with the
1223 <schemaRequest> element. It is of type SchemaResponse which extends SpmlResponse.

1224 If successful, the element MUST contain one or more <schema> elements which describes the
1225 requested schemas.

1226 The following schema fragment defines the <schemaResponse> element and related types:
1227

```xml
<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>
```

1228   7.3   Element <schema>

1229   The <schema> element conveys the description of one schema supported by a PSP.

1230   The element MUST contain a <providerIdentifier> which identifies the provider associated with the
1231   schema.

1232   The element MUST contain a <schemaIdentifier> which identifies the schema within the context of
1233   the associated provider identifier. If the <schemaIdentifier> is a URN, then references to this
1234   schema MAY be made using only the <schemeIdentifier>. If the <schemaIdentifier> is not a URN,
1235   then references to the schema MUST be made using both <providerIdentifier> and
1236   <schemaIdentifer>.

1237   The element MAY contain zero or more <objectclassDefinition> elements which describe the object
1238   classes in the schema. The "name" attribute of all <objectclassDefinition> elements within the
1239   <schema> MUST be unique. Different schemas MAY contain object class definitions withthe same
1240   name.

1241   The element MAY contain zero or more <attributeDefinition> elements which describe the attributes
1242   used in the object class and extended request definitions. The "name" attribute of all
1243   <attributeDefinition> elements within the <schema> MUST be unique. Different schemas MAY
1244   contain attribute definitions with the same name.

1245   The element MAY contain zero or more <extendedRequestDefinition> elements. The
1246   <operationIdentifier> elements of all <extendedRequestDefinition> elements within the <schema>
1247   MUST be unique. Different schemas MAY contain extended request definitions with the same
1248   name.

1249   The following schema fragment defines the <schema> element and related types:
1250

```
<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>
```

1251　7.4　Element <schemaIdentifier>

1252 The <schemaIdentifier> element is used to identify a schema that may be requested in a
1253 <schemaRequest>, or referenced in an <objectclassDefinitionReference> or
1254 <attributeDefinitionReference>.

1255 The element MUST contain a "schemaIDType" attribute that specifies the type of the identifier. It
1256 MUST be one of:
1257

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.

1258

1259 The element MUST contain a <schemaID> element whose content is consistent with the value of
1260 the "schemaIDType" attribute.

1261 The following schema fragment defines the <schemaIdentifier> element and related types:
1262

```
 <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>
```

1263　7.5　Element <properties>

1264 The <properties> element MAY be used in <objectclassDefinition>, <attributeDefinition>, and
1265 <extendedRequestDefinition> elements to convey PSP-specific information about the semantics of
1266 the related definition.

1267 The element MAY contain any number of DSML <attr> elements. The names and semantics of the
1268 property attributes is defined by the PSP.

1269 The following schema fragment defines the <properties> element and related types:
1270

```
 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>
```

1271   7.6   Element <attributeDefinition>

1272   The <attributeDefinition> element is used to describe the definition of one attribute within a schema.

1273   The element MUST have a "name" attribute which MUST be unique among <attributeDefinition>
1274   elements within a <schema>.

1275   The element MAY have a "type" attribute which specifies the fundamental data type of this attribute.
1276   If type is not specified, the default is "xsd:string" indicating that the attribute is a character string.

1277   The element MAY have a "multi-valued" attribute whose value is either "true" or "false". When the
1278   value is "true", it indicates that more than one value is allowed for this attribute.

1279   The element MAY have a "description" attribute which should contain readable text that describes
1280   the semantics of this attribute.

1281   The element MAY have <properties> element to convey PSP-specific information about an
1282   attribute.

1283   The following schema fragment defines the <attributeDefinition> element and related types:
1284

```
 <<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>
```

1285   7.7   Element <attributeDefinitionReference> and Type
1286   AttributeDefinitionReferences

1287   The <attributeDefinitionReference> element is used to refer to an <attributeDefinition> element
1288   from within another element.

1289   The element MUST have a "name" attribute which MUST correspond to the name of an
1290   <attributeDefinition> defined in a <schema>.

1291   The element MAY have a <providerIdentifier> element and MAY have a <schemaIdentifier>
1292   element. These two elements are used to refer to attributes defined in a schema other than the one
1293   in which the reference is contained.

1294   If both the <providerIdentifier> and <schemaIdentifier> are omitted, then an <attributeDefinition>
1295   whose "name" attribute matches the "name" of the reference MUST be defined in the <schema>
1296   that contains the reference.

1297   If both the <providerIdentifier> and <schemaIdentifier> are specified, <providerIdentifier> must be a
1298   URN, and <schemaIdentifier> must be a non-URN.

1299 If only <schemaIdentifier> is specified, it MUST be a URN. If only <providerIdentifier> is specified,
1300 the request is malformed.

1301 The combination of the <providerIdentifier> and <schemaIdentifier> MUST identify a schema
1302 supported by the PSP.

1303 The element MAY have a "required" attribute whose values may be "true" or "false". When "true" it
1304 indicates that a value for this attribute is required. The semantics of a required attribute are defined
1305 by the context of the reference which will be either an <objectclassDefinition> or
1306 <extendedRequesteDefinition>.

1307 The type AttributeDefinitionReferences defines an unbounded sequence of
1308 <attributeDefinitionReference> elements.

1309 The following schema fragment defines the <attributeDefinitionReference> element and related
1310 types:
1311

```
<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>
```

1312 7.8 Element <objectclassDefinition>

1313 The <objectclassDefinition> element is used to describe the definition of one object class in a
1314 schema.

1315 The element MUST have a "name" attribute which MUST be unique among <objectclassDefinition>
1316 elements within a <schema>.

1317 The element MAY have a "description" attribute which should contain readable text that describes
1318 the semantics of this class.

1319 The element MAY have <properties> element to convey PSP-specific information about a class.

1320 The element MAY have a <superiorClasses> element which is defined by type
1321 ObjectclassDefinitionReferences and contains any number of <objectclassReference> elements.
1322 When a class defines superior classes, it indicates that all attributes defined by a superior class are
1323 also defined for the referencing class.

1324 The element MAY have a <memberAttributes> element which is defined by type
1325 AttributeDefinitionReferences and contains any number of <attributeDefinitionReference> elements.

1326 The following schema fragment defines the <objectclassDefinition> element and related types:
1327

```
 <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>
```

1328   **7.9   Element <objectclassDefinition>**

1329 The <objectclassDefinitionReference> element is used to refer to an <objectclassDefinition>
1330 element from within the <superiorClasses> element of another <objectclassDefinition>.

1331 The element MUST have a "name" attribute which MUST correspond to the name of an
1332 <objectclassDefinition> defined in a <schema>.

1333 The element MAY have a <providerIdentifier> element and MAY have a <schemaIdentifier>
1334 element. These two elements are used to refer to classes defined in a schema other than the one
1335 in which the reference is contained.

1336 If both the <providerIdentifier> and <schemaIdentifier> are omitted, then an <objectclassDefinition>
1337 whose "name" attribute matches the "name" of the reference MUST be defined in the <schema>
1338 that contains the reference.

1339 If both the <providerIdentifier> and <schemaIdentifier> are specified, <providerIdentifier> must be a
1340 URN, and <schemaIdentifier> must be a non-URN.

1341 If only <schemaIdentifier> is specified, it MUST be a URN.

1342 If only <providerIdentifier> is specified, the request is malformed.

1343 The combination of the <providerIdentifier> and <schemaIdentifier> MUST identify a schema
1344 supported by the PSP.

1345 The following schema fragment defines the <objectclassDefinitionReference> element and related
1346 types:
1347

```
<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>
```

1348 ## 7.10 Element <extendedRequestDefinition>

1349 The <extendedRequestDefinition> element is used to describe the definition of one extended
1350 request supported by a PSP.

1351 The element MUST have a <operationIdentifier> element which uniquely identifies this request
1352 among requests in the schema.

1353 [NOTE: Now that extended requests are defined in a schema, it is inconsistent to refer to them
1354 using operationIdetifiers when everything else will use schemaIdentifier URNs in combination with
1355 the "name" attribute. In other words, the <schema identifier> plus "name" is in effect the same as
1356 the <operationIdentifier>. That's what we do for objectclass values, why not extended requests?]

1357 The element MAY have a "description" attribute which should contain readable text that describes
1358 the semantics of this class.

1359 The element MAY have <properties> element to convey PSP-specific information about the
1360 extended request.

1361 The element MAY have a <parameters> element of type AttributeDefinitionReferences containing
1362 any number of <attributeDefinitionReference> elements. These define the parameters that may be
1363 passed in the <attributes> element of an <extendedRequest>. If an <attributeDefinitionReference>
1364 has a "required" attribute of "true", then a value for that attribute MUST be specified in the
1365 <extendedRequest>.

1366 The element MAY have a <returnValues> element of type AttributeDefinitionReferences containing
1367 any number of <attributeDefinitionReference> elements. These define the attributes that may be
1368 returned to the RA in the <attributes> element of an <extendedResponse>. If an
1369 <attributeDefinitionReference> has a "required" attribute of "true", then the PSP MUST include a
1370 value for that attribute in the <extendedResponse>.

1371 The following schema fragment defines the <extendedRequestDefinition> and related types:
1372

```
<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.

1400 The solution to a message insertion attack is to use mutual authentication and a message
1401 sequence integrity mechanism between the actors. It should be noted that just using SSL mutual
1402 authentication is not sufficient. This only proves that the other party is the one identified by the
1403 subject of the X.509 certificate. In order to be effective, it is necessary to confirm that the certificate
1404 subject is authorized to send the message.

1405 8.1.2.2. Message deletion

1406 A message deletion attack is one in which the adversary deletes messages in the sequence of
1407 messages between SPML actors. Message deletion may lead to denial of service. However, a
1408 properly designed SPML system should not trigger false provisioning on as the result of a message
1409 deletion attack.

1410 The solution to a message deletion attack is to use a message integrity mechanism between the
1411 actors.

1412 8.1.2.3. Message modification

1413 If an adversary can intercept a message and change its contents, then they may be able to alter a
1414 provisioning request. Message integrity mechanisms can prevent a successful message
1415 modification attack.

1416 8.2. Safeguards

1417 8.2.1. Authentication

1418 Authentication provides the means for one party in a transaction to determine the identity of the
1419 other party in the transaction. Authentication may be in one direction, or it may be bilateral.

1420 Given the sensitive nature of many provisioning requests and systems it is important for an *RA* to
1421 authenticate the identity of the *PSP* to which it issues SPML requests. Otherwise, there is a risk
1422 that an adversary could provide false or invalid *PSP*, leading to a possible security violation.

1423 It is equally important for a *PSP* to authenticate the identity of the *RA* and assess the level of trust
1424 and to determine if the *RA* is authorized to request this service/operation.

1425 Many different techniques may be used to provide authentication, such as co-located code, a
1426 private network, a VPN or digital signatures. Authentication may also be performed as part of the
1427 communication protocol used to exchange the requests. In this case, authentication may be
1428 performed at the message level or at the session level.

1429 8.2.2. Confidentiality

1430 Confidentiality mechanisms ensure that the contents of a message can be read only by the desired
1431 recipients and not by anyone else who encounters the message while it is in transit. The primary
1432 concern is confidentiality during transmission.

1433 8.2.2.1. Communication confidentiality

1434 In some environments it is deemed good practice to treat all data within a provisioning domain as
1435 confidential. In other environments certain parts of the service schema and required attributes may
1436 b openly published. Regardless of the approach chosen, the security of the provisioning system as

1437 a whole should not be in any way dependant on the secrecy of the service, its provider or its
1438 request data schema.

1439 Any security concerns or requirements related to transmitting or exchanging SPML documents lies
1440 outside the scope of the SPML standard. While it is often important to ensure that the integrity and
1441 confidentiality of provisioning requests, it is left to the implementers to determine the appropriate
1442 mechanisms for their environment.

1443 Communications confidentiality can be provided by a confidentiality mechanism, such as SSL.
1444 Using a point-to-point scheme like SSL may lead to other vulnerabilities when one of the end-points
1445 is compromised.

1446 8.2.2.2. Trust model

1447 Discussions of authentication, integrity and confidentiality mechanisms necessarily assume an
1448 underlying trust model: how can one actor come to believe that a given key is uniquely associated
1449 with a specific, identified actor so that the key can be used to encrypt data for that actor or verify
1450 signatures (or other integrity structures) from that actor? Many different types of trust model exist,
1451 including strict hierarchies, distributed authorities, the Web, the bridge and so on.

1452 8.2.2.3. Privacy

1453 It is important to be aware that any transactions that occur in an SPML model system may contain
1454 private and secure information about the actors. Selection and use of privacy mechanisms
1455 appropriate to a given environment are outside the scope of this specification. The decision
1456 regarding whether, how and when to deploy such mechanisms is left to the implementers
1457 associated with the environment.

1458

1459 9. Conformance (normative)

1460 9.1. Introduction

1461 The OASIS procedure for ratification of a committee specification as an OASIS standard requires
1462 that three independent implementers attest that they are "successfully using" the committee
1463 specification.

1464 9.2. Conformance tables

1465 This section lists those portions of the specification that MUST be included in an implementation of
1466 an RA or an SPML service that claims to conform to SPML v1.0.

1467 Note: "M" means mandatory to implement. "O" means optional to implement. "O* means optional
1468 to implement but must implement one of AddRequest, ModifyRequest or DeleteRequest . "NA"
1469 means does not apply.

1470 The implementation MUST support ALL those schema elements that are marked "M" and MUST
1471 support one of either AddRequest, ModifyRequest or DeleteRequests in the columns marked "O*"

1472

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 (requires support for spml:BatchRequest)	M	M	M	M
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

1473 ## 9.3   Data Types

1474 The implementation MUST support the data types associated with the following identifiers marked
1475 "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

1524

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

1525

1526 Appendix C. Notices

1527 OASIS takes no position regarding the validity or scope of any intellectual property or other rights
1528 that might be claimed to pertain to the implementation or use of the technology described in this
1529 document or the extent to which any license under such rights might or might not be available;
1530 neither does it represent that it has made any effort to identify any such rights. Information on
1531 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS
1532 website. Copies of claims of rights made available for publication and any assurances of licenses to
1533 be made available, or the result of an attempt made to obtain a general license or permission for
1534 the use of such proprietary rights by implementers or users of this specification, can be obtained
1535 from the OASIS Executive Director.

1536 OASIS has been notified of intellectual property rights claimed in regard to some or all of the
1537 contents of this specification. For more information consult the online list of claimed rights.

1538 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
1539 applications, or other proprietary rights which may cover technology that may be required to
1540 implement this specification. Please address the information to the OASIS Executive Director.

1541 Copyright (C) OASIS Open 2002. All Rights Reserved.

1542 This document and translations of it may be copied and furnished to others, and derivative works
1543 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
1544 published and distributed, in whole or in part, without restriction of any kind, provided that the above
1545 copyright notice and this paragraph are included on all such copies and derivative works. However,
1546 this document itself may not be modified in any way, such as by removing the copyright notice or
1547 references to OASIS, except as needed for the purpose of developing OASIS specifications, in
1548 which case the procedures for copyrights defined in the OASIS Intellectual Property Rights
1549 document must be followed, or as required to translate it into languages other than English.

1550 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
1551 successors or assigns.

1552 This document and the information contained herein is provided on an "AS IS" basis and OASIS
1553 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
1554 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY
1555 RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
1556 PARTICULAR PURPOSE.