



Functional Elements Specification

Working Draft 02, 25-Nov-2004

Document identifier:

FWSI-FESC-specifications-02.doc

Location:

<http://www.oasis-open.org/apps/org/workgroup/fws/documents.php>

Editor:

Tan Puay Siew, Singapore Institute of Manufacturing Technology, SIMTech
(pstan@simtech.a-star.edu.sg)

Contributor(s):

Cheng Huang Kheng, SIMTech (jason@simtech.a-star.edu.sg)

Abstract:

The ability to provide robust implementations is a very important aspect to create high quality Web Service-enabled applications and to accelerate the adoption of Web Services. The Framework for Web Services Implementation (FWSI) TC aims to enable robust implementations by defining a practical and extensible methodology consisting of implementation processes and common functional elements that practitioners can adopt to create high quality Web Services systems without reinventing them for each implementation.

This document specifies a set of Functional Elements for practitioners to instantiate into a technical architecture, and should be read in conjunction with the Functional Elements Requirements document. It is the purpose of this specification to define the right level of abstraction for these Functional Elements and to specify the purpose and scope of each Functional Element so as to facilitate efficient and effective implementation of Web Services.

Status:

This document is updated periodically on no particular schedule.

Committee members should send comments on this specification to the fwsifesc@lists.oasis-open.org list. Others should subscribe to and send comments to the fwsicomment@lists.oasis-open.org list. To subscribe, send an email message to fwsicomment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.

For information on whether any patents¹ have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to

¹ This document contains concepts that have been filed as patents. The Intellectual Property Rights declaration and contractual terms on use of document's content will be made available at a later date.

37 the Intellectual Property Rights section of the FWSI TC web page ([http://www.oasis-](http://www.oasis-open.org/committees/fwsi/)
38 [open.org/committees/fwsi/](http://www.oasis-open.org/committees/fwsi/)).

Table of Contents

40	1	<i>Introduction</i>	7
41	1.1	<i>Document Outline</i>	7
42	1.2	<i>Motivation</i>	8
43	1.3	<i>Terminology</i>	8
44	2	<i>List of Functional Elements</i>	9
45	2.1	<i>Event Handler Functional Element</i>	9
46	2.1.1	<i>Motivation</i>	9
47	2.1.2	<i>Terms Used</i>	9
48	2.1.3	<i>Key Features</i>	10
49	2.1.4	<i>Interdependencies</i>	12
50	2.1.5	<i>Related Technologies and Standards</i>	12
51	2.1.6	<i>Model</i>	13
52	2.1.7	<i>Usage Scenarios</i>	14
53	2.2	<i>Group Management Functional Element</i>	31
54	2.2.1	<i>Motivation</i>	31
55	2.2.2	<i>Terms Used</i>	31
56	2.2.3	<i>Key Features</i>	32
57	2.2.4	<i>Interdependency</i>	32
58	2.2.5	<i>Related Technologies and Standards</i>	33
59	2.2.6	<i>Model</i>	33
60	2.2.7	<i>Usage Scenarios</i>	33
61	2.3	<i>Identity Management Functional Element</i>	38
62	2.3.1	<i>Motivation</i>	38
63	2.3.2	<i>Terms Used</i>	38
64	2.3.3	<i>Key Features</i>	40
65	2.3.4	<i>Interdependencies</i>	40
66	2.3.5	<i>Related Technologies and Standards</i>	41
67	2.3.6	<i>Model</i>	42
68	2.3.7	<i>Usage Scenarios</i>	43
69	2.4	<i>Log Utility Functional Element</i>	47
70	2.4.1	<i>Motivation</i>	47
71	2.4.2	<i>Terms Used</i>	47
72	2.4.3	<i>Key Features</i>	47
73	2.4.4	<i>Interdependencies</i>	48
74	2.4.5	<i>Related Technologies and Standards</i>	48
75	2.4.6	<i>Model</i>	49
76	2.4.7	<i>Usage Scenarios</i>	49
77	2.5	<i>Notification Functional Element</i>	56
78	2.5.1	<i>Motivation</i>	56
79	2.5.2	<i>Terms Used</i>	56
80	2.5.3	<i>Key Features</i>	57
81	2.5.4	<i>Interdependencies</i>	57
82	2.5.5	<i>Related Technologies and Standards</i>	57
83	2.5.6	<i>Model</i>	58
84	2.5.7	<i>Usage Scenarios</i>	58

85	2.6 Phase and Lifecycle Management Functional Element.....	64
86	2.6.1 Motivation	64
87	2.6.2 Terms Used	64
88	2.6.3 Key Features	65
89	2.6.4 Interdependencies	65
90	2.6.5 Related Technologies and Standards	65
91	2.6.6 Model.....	66
92	2.6.7 Usage Scenarios	66
93	2.7 Presentation Transformer Functional Element	71
94	2.7.1 Motivation	71
95	2.7.2 Terms Used	71
96	2.7.3 Key Features	71
97	2.7.4 Interdependencies	71
98	2.7.5 Related Technologies and Standards	71
99	2.7.6 Model.....	72
100	2.7.7 Usage Scenario	72
101	2.8 Role and Access Management Functional Element	74
102	2.8.1 Motivation	74
103	2.8.2 Terms Used	74
104	2.8.3 Key Features	75
105	2.8.4 Interdependencies	76
106	2.8.5 Related Technologies and Standards	76
107	2.8.6 Model.....	77
108	2.8.7 Usage Scenario	77
109	2.9 Search Functional Element.....	87
110	2.9.1 Motivation	87
111	2.9.2 Terms Used	87
112	2.9.3 Key Features	88
113	2.9.4 Interdependencies	88
114	2.9.5 Related Technologies and Standards	89
115	2.9.6 Model.....	89
116	2.9.7 Usage Scenario	89
117	2.10 Secure SOAP Management Functional Element.....	93
118	2.10.1 Motivation	93
119	2.10.2 Terms Used	93
120	2.10.3 Key Features	94
121	2.10.4 Interdependencies	94
122	2.10.5 Related Technologies and Standards	94
123	2.10.6 Model.....	95
124	2.10.7 Usage Scenarios	95
125	2.11 Sensory Functional Element.....	99
126	2.11.1 Motivation	99
127	2.11.2 Terms Used	99
128	2.11.3 Key Features	99
129	2.11.4 Interdependencies	100
130	2.11.5 Related Technologies and Standards	100
131	2.11.6 Model.....	100
132	2.11.7 Usage Scenarios	100
133	2.12 Service Management Functional Element	103
134	2.12.1 Motivation	103

135	2.12.2 Terms Used	103
136	2.12.3 Key Features	103
137	2.12.4 Interdependencies	104
138	2.12.5 Related Technologies and Standards	104
139	2.12.6 Model.....	105
140	2.12.7 Usage Scenarios	105
141	2.13 Service Registry Functional Element.....	111
142	2.13.1 Motivation	111
143	2.13.2 Terms Used	111
144	2.13.3 Key Features	112
145	2.13.4 Interdependencies	112
146	2.13.5 Related Technologies and Standards	112
147	2.13.6 Model.....	113
148	2.13.7 Usage Scenario	113
149	2.14 Service Tester Functional Element.....	122
150	2.14.1 Motivation	122
151	2.14.2 Terms Used	122
152	2.14.3 Key Features	122
153	2.14.4 Interdependencies	122
154	2.14.5 Related Technologies and Standards	122
155	2.14.6 Model.....	123
156	2.14.7 Usage Scenarios	123
157	2.15 User Management Functional Element.....	125
158	2.15.1 Motivation	125
159	2.15.2 Terms Used	125
160	2.15.3 Key Features	126
161	2.15.4 Interdependencies	127
162	2.15.5 Related Technologies and Standards	127
163	2.15.6 Model.....	127
164	2.15.7 Usage Scenarios	128
165	2.16 Web Service Aggregator Functional Element.....	135
166	2.16.1 Motivation	135
167	2.16.2 Terms Used	135
168	2.16.3 Key Features	136
169	2.16.4 Interdependencies	137
170	2.16.5 Related Technologies and Standards	137
171	2.16.6 Model.....	137
172	2.16.7 Usage Scenarios	138
173	3 Functional Elements Usage Scenario	143
174	3.1 Service Monitoring	144
175	3.2 Securing SOAP Messages	145
176	3.3 Decoupled User Access Management	146
177	4 References	147
178	Appendix A. Acknowledgments.....	149
179	Appendix B. Revision History.....	150
180	Appendix C. Notices.....	151
181		

1 Introduction

The purpose of OASIS Framework for Web Services Implementation (FWSI) Technical Committee (TC) is to facilitate implementation of robust Web Services by defining a practical and extensible methodology consisting of implementation processes and common functional elements that practitioners can adopt to create high quality Web Services systems without re-inventing them for each implementation. It aims to solve the problem of the slow adoption of Web Services due to a lack of good Web Services methodologies for implementation, cum a lack of understanding and confidence in solutions that have the necessary components to reliably implement Web Service-enabled applications.

One of the FWSI TC's deliverables is the Functional Elements Specification, which is detailed in this document. This Specification specifies a set of functional elements that practical implementation of Web Services-based systems will require. A Functional Element (FE) is defined as a building block representing common reusable functionalities for Web Service-enabled implementations, i.e. from an application Point-Of-View. These FEs are expected to be implemented as reusable components, with Web Services capabilities where appropriate, and to be the foundation for practitioners to instantiate into a technical architecture. The implementations of these FEs are further supported by another complementary work that is also from the FWSI TC, the Web Services Implementation Methodology (WSIM) [1]. As such, the TC hopes that through the implementations of these FEs, robust Web Service-enabled applications can be constructed quickly and deployed in a rapid manner.

The target audiences for this document are expected to be solution providers who intend to use the Functional Elements Specification to create building blocks that can be instantiated into the technical architecture of their solutions or software vendors and independent software vendors (ISVs) that are expected to build the functional elements specified into their products. Individuals and researchers who are interested in Web Services will also be able to benefit from this document. It is recommended that this document should be used in tandem with the Functional Elements Requirements document, to ensure that readers have a holistic view to the thought processes and knowledge that are encapsulated.

1.1 Document Outline

This document describes the Functional Elements in three main sections. In this section, explanation on the motivation for creating this Specification and the kind of impact that it will create for Web Service-enabled implementations and the terminology used in the normative part of this document are included.

Section 2 lists the identified Functional Elements arising from requirements documented in the Functional Elements Requirements document [2]. Under each of the ensuing FE, the following descriptions are provided:

- Motivation

A section for providing a short introduction explaining the motivation of including the FE from an application Point-Of-View, including cross-referencing of the requirements for the Functional Element

- 229 • Terms Used
230 A glossary of the terms used. An explanation or illustration of the runtime capabilities of the
231 Functional Element are also provided where appropriate.
- 232 • Key Features
233 A list of key features to be implemented are provided here and is expressed in the normative
234 form.
- 235 • Interdependencies
236 In this section, the interdependencies between Functional Elements are provided to clarify
237 the linkages between FEs (if any).
- 238 • Related Technologies and Standards
239 Here, the reliance of the Functional Elements on related technologies and specifications (or
240 standards) are provided
241
- 242 Section 3 provides the examples of how the Functional Elements can be assembled to accelerate
243 web service-enabled applications. From these Functional Elements, a variety of solutions can be
244 built.
245

246 1.2 Motivation

247
248 In a Service-Oriented Architecture (SOA) environment, new applications/services are created
249 through the assembly of existing services. One of the key advantages of this loosely coupled
250 model is that it allows the new application/service to leverage on 3rd party services. As a typical
251 3rd party's implementation of the services is done via the software component approach, this
252 specification further proliferate new applications/services by defining a framework for Web
253 Services implementation consisting of Functional Elements. Through these Functional Elements,
254 which are implementation neutral, this Specification hopes to influence future software
255 development towards assembly of services rather than 'pure built only'.

256 1.3 Terminology

257
258 Within this document the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL
259 NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
260 document are to be interpreted as described in RFC2119 [3].
261

262 Cross-references to the Functional Elements Requirements document [2] are designated
263 throughout this specification to the requirement contained where the requirement number is
264 enclosed in square brackets (e.g. **[MANAGEMENT-005]**).
265
266

267 2 List of Functional Elements

268 2.1 Event Handler Functional Element

269 2.1.1 Motivation

270 Information is in abundance in a service-oriented environment. However, not all information is
271 applicable to a particular enterprise and there lies the need to control information flow in an
272 organization. In a Web Service-enabled implementation, the Event Handler Functional Element
273 can help to fulfill this need by:

- 274 • Managing the information flow through a subscription based mechanism,
- 275 • Streamlining information into meaningful categories so as to improve relevancy to a
276 potential consumer of the information, and
- 277 • Refining information flow via a filtering mechanism

278

279 This Functional Element fulfills the following requirements from the Functional Elements
280 Requirements, Working Draft 01a:

- 281 • Primary Requirements
 - 282 • MANAGEMENT-111,
 - 283 • PROCESS-005, and
 - 284 • PROCESS-100 to PROCESS-117.
- 285 • Secondary Requirements
- 286 • None

287 2.1.2 Terms Used

Terms	Description
Active Event Detection	Active Event Detection refers to the capability to periodically detect the occurrence of an external Event.
Channel	A Channel is a logical grouping of similar event types generated by the suppliers. When an Event is routed to a channel, all the Event Consumers who have subscribed to that Channel will be notified.
Event	An Event is an indication of an occurrence of an activity, such as the availability of a discounted air ticket. In such a case, it will trigger a follow-up action such as the URL where the ticket can be bought. Interested event consumer can then proceed with the purchase at the designated URL.
Event Consumer	An Event Consumer is a receiver of the events generated by an Event Supplier.
Event Supplier	An Event Supplier generates Event. It can be an application or a service, or even a person. Note that Event Suppliers are typically external to the Event Handler.

Filter	A Filter is a mechanism for defining Event that is of value to the Event Consumer.
Routing Rule	A Routing Rule defines how an Event is routed. An Event can be routed to a Channel or directly to an Event Consumer.

288

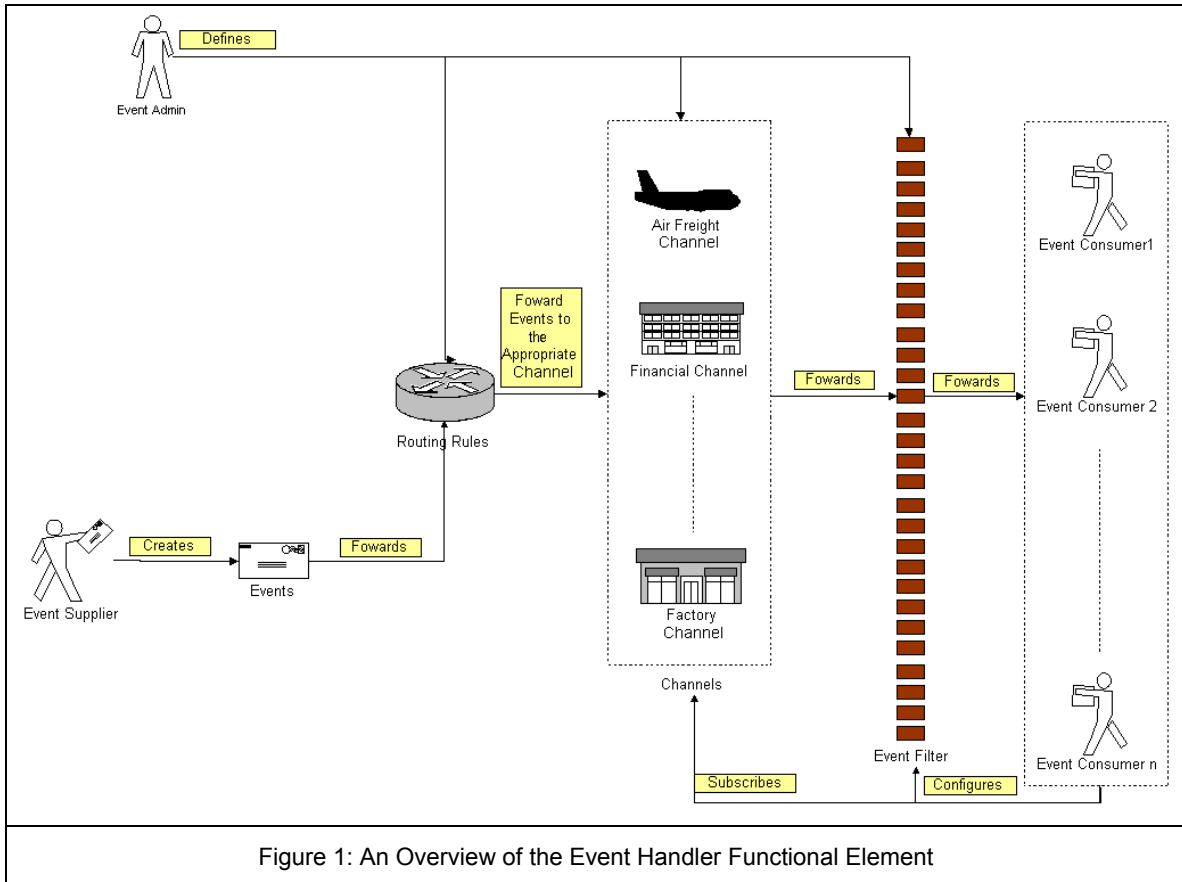


Figure 1: An Overview of the Event Handler Functional Element

289

290 Figure 1 depicts the basic concepts of how the participating entities collaborate together in the
 291 Event Handler Functional Element. Beginning with the event supplier who generates an event,
 292 the event is subsequently routed to the routing rules engine. Depending on the rules specified by
 293 the event administrator on the engine, the event could be routed to an appropriate channel, for
 294 example, the airfreight channel. In this case, a notification message will be sent to the subscribing
 295 event consumers. In between that, there is a filtering engine to determine if a particular event is
 296 meaningful to the intended recipients and this is configurable by the recipients themselves.

297 2.1.3 Key Features

298 Implementations of the Event Handler Functional Element are expected to provide the following
 299 key features:

- 300 1. The Functional Element MUST provide the capability to manage the creation (or registration)
 301 and deletion of instances of the following concepts based on a pre-defined structure:
- 302 1.1. Event Supplier,
- 303 1.2. Event Consumer,
- 304 1.3. Event,

- 305 1.4. Filter,
306 1.5. Channel, and
307 1.6. Routing Rule.
- 308 2. The Functional Element MUST provide the capability to manage all the information (attribute
309 values) stored in such concepts. This includes the capability to retrieve and update
310 attribute's values belonging to the concepts mentioned in Key Feature (1).
- 311 3. The Functional Element MUST provide the capability to enable Event Suppliers to trigger
312 relevant Events.
- 313 4. The Functional Element MUST provide a mechanism to associate/unassociate Routing
314 Rules to an Event.
- 315 *Example: As shown in Figure 1, where an event can be routed to Air Freight or Financial*
316 *Channel or even to all channels based on the Routing Rules that are associated*
317 *with the Event.*
- 318 5. As part of Key Feature (3), the Routing Rules must be able to route an event to all, specified
319 Channels or individual Event Consumers.
- 320 6. The Functional Element MUST enable Event Consumers to execute the following tasks to
321 improve the relevancy of the incoming events”
- 322 6.1. Subscribe/Unsubscribe to relevant Channel(s), and
323 6.2. Apply a filter to the appropriate channel or event, which helps to refine the criteria of a
324 useful event further.
- 325 7. The Functional Element MUST provide the capability to notify relevant Event Consumers
326 when an event occurs.
- 327 Examples of notification types include SMS, email and Web Services invocations.
- 328 8. As part of Key Feature (6), the notification must be able to handle differing requirements
329 arising from different notification formats.
- 330 *Example: If the incoming event contains 2 important attributes, the order or position of*
331 *these 2 attributes must be configurable to suit the convenience of the Event*
332 *Consumer. This is extremely important in the case of Web Service Invocations.*
- 333 10. The Functional Element MUST provide a mechanism for managing the concepts specified
334 across different application domains.
- 335 *Example: Namespace control mechanism*
336
- 337 In addition, the following key features could be provided to enhance the Functional Element
338 further:
- 339 1. The Functional Element MAY provide a mechanism to enable active event detection.
- 340 2. If Key Feature (1) is implemented, then the Functional Element MUST provide the following
341 capabilities also:
- 342 2.1. Non-intrusive detection
343 *Example: The detection of a new event through periodic inspection of the audit log.*
344 2.2. Configurable event detection schedule
345 *Example: To inspect the audit log every 2 hours, where the duration between*
346 *inspections is configurable.*
347 2.3. Ability to retrieve relevant data from external source(s) for further event processing by
348 Event Handler
349 *Example: To retrieve Error Type and Message from audit log.*
- 350 3. The Functional Element MAY provide the capability to record event processing within the
351 Event Handler. The logging of event processing includes the occurrences of event, sending
352 of notifications, warning and error messages generated in the processing of events.

353 4. The Functional Element MAY provide the capability scheduled-based event notification.
354

355 **2.1.4 Interdependencies**

Direct Dependency

Log Utility Functional Element	The Log Utility Functional Element helps to log the audit trail.
--------------------------------	--

356

Interaction Dependency

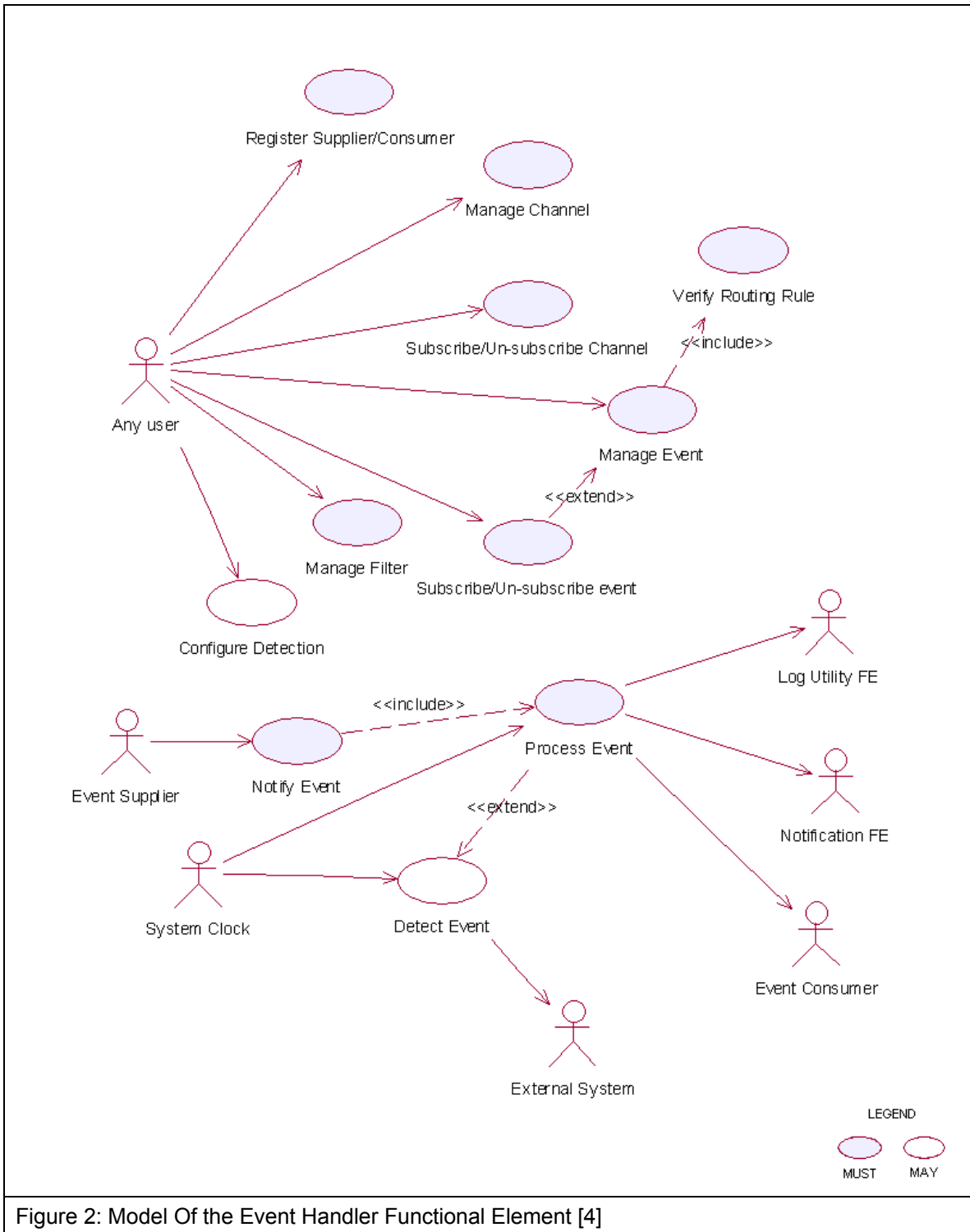
Notification Functional Element	The Notification Functional Element helps to send SMS and email to the appropriate Event Consumer.
---------------------------------	--

357

358 **2.1.5 Related Technologies and Standards**

359 None

2.1.6 Model



361 **2.1.7 Usage Scenarios**

362 **2.1.7.1 Register Supplier/Consumer**

363 **2.1.7.1.1 Description**

364 This use case allows the user to register itself to the Event Handler Functional Element as an
365 event supplier or an event consumer.

366 **2.1.7.1.2 Flow of Events**

367 **2.1.7.1.2.1 Basic Flow**

368 The use case begins when the user of the Event Handler wants to register an event supplier or
369 event consumer with the Event Handler.

370 1: The user sends a request to Event Handler together with its profile data and operation.

371 2: Based on the operation it specified, one of the following sub-flows is executed:

- 372 • If the operation is '**Register as supplier**', then sub-flow 2.1 is executed.
- 373 • If the operation is '**Register as consumer**', then sub-flow 2.2 is executed.
- 374 • If the operation is '**Un-register as supplier**', then sub-flow 2.3 is executed.
- 375 • If the operation is '**Un-register as consumer**', then sub-flow 2.4 is executed.
- 376 • If the operation is '**Update supplier**', then sub-flow 2.5 is executed.
- 377 • If the operation is '**Update consumer**', then sub-flow 2.6 is executed.
- 378 • If the operation is '**Retrieve supplier**', then sub-flow 2.7 is executed.
- 379 • If the operation is '**Retrieve consumer**', then sub-flow 2.8 is executed.

380 2.1: Register as Supplier.

381 2.1.1: The Functional Element gets the user profile data, i.e. namespace, name,
382 description and type.

383 2.1.2: The Functional Element registers the user as event supplier.

384 2.1.3: The Functional Element returns the Supplier Id to the user.

385 2.2: Register as Consumer.

386 2.2.1: The Functional Element gets the user profile data, i.e. namespace, name,
387 description and type.

388 2.2.2: The Functional Element registers the user as event consumer.

389 2.2.3: The Functional Element returns the Consumer Id to the user.

390 2.3: Un-register as Supplier.

391 2.3.1: The Functional Element gets the user namespace and name or User Id.

392 2.3.2: The Functional Element checks whether the user is a supplier.

393 2.3.3: The Functional Element removes the user as supplier.

394 2.4: Un-register as Consumer.

395 2.4.1: The Functional Element gets the user namespace and name or User Id.

396 2.4.2: The Functional Element checks whether the user is a consumer.

397 2.4.3: The Functional Element removes the user as consumer.

398 2.5: Update Supplier.

399 2.5.1: The Functional Element gets the user namespace and name or User Id together
400 with the user profile.

401 2.5.2: The Functional Element checks whether the user is a supplier.

402 2.5.2: The Functional Element updates the user profile.

403 2.6: Update Consumer.

404 2.6.1: The Functional Element gets the user namespace and name or User Id together
405 with the user profile.

406 2.6.2: The Functional Element checks whether the user is a consumer.

407 2.6.3: The Functional Element updates the user profile.

408 2.7: Retrieve Supplier.

409 2.7.1: The Functional Element gets the user namespace and name or User Id.

410 2.7.2: The Functional Element checks whether the user is a supplier.

411 2.7.3: The Functional Element returns the user profile.

412 2.8: Retrieve Consumer.

413 2.8.1: The Functional Element gets the user namespace and name or User Id.

414 2.8.2: The Functional Element checks whether the user is a consumer.

415 2.8.3: The Functional Element returns the user profile.

416 3: The Functional Element returns the results to indicate the success or failure of this operation to
417 the user and the use case ends.

418 **2.1.7.1.2.2 Alternative Flows**

419 1: Supplier Already Registered.

420 1.1: If in the basic flow 2.1.2, the user already registered as supplier, Functional Element will
421 return an error message to the user and the use case ends.

422 2: Consumer Already Registered.

423 2.1: If in the basic flow 2.2.2, the user already registered as consumer, Functional Element
424 will return an error message to the user and the use case ends.

425 3: Supplier or Consumer Not Registered.

426 3.1: If in the basic flow 2.3.2, 2.4.2, 2.5.2, 2.6.2, 2.7.2, and 2.8.2, the user specified is not
427 registered, Functional Element will return an error message to the user and the use case
428 ends.

429 4: Persistency Mechanism Error.

430 4.1: If in the basic flow 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2,7 and 2.8, the Functional Element cannot
431 perform data persistency, Functional Element will return an error message to the user and the
432 use case ends.

433

434 **2.1.7.1.3 Special Requirements**

435 None.

436 **2.1.7.1.4 Pre-Conditions**

437 None.

438 **2.1.7.1.5 Post-Conditions**

439 None.

440 **2.1.7.2 Manage Channel**

441 **2.1.7.2.1 Description**

442 This use case allows the user to manage channels.

443 **2.1.7.2.2 Flow of Events**

444 **2.1.7.2.2.1 Basic Flow**

445 The use case begins when the user wants to create/retrieve/update/delete a channel

446 1: The user sends request to manipulate a channel.

447 2: Based on the operation it specifies, one of the following sub-flows is executed:

- 448 • If the operation is '**Create Channel**', the sub-flow 2.1 is executed.
- 449 • If the operation is '**Retrieve Channel**', the sub-flow 2.2 is executed.
- 450 • If the operation is '**Update Channel**', the sub-flow 2.3 is executed.
- 451 • If the operation is '**Delete Channel**', the sub-flow 2.4 is executed.

452 2.1: Create Channel.

453 2.1.1: The Functional Element gets channel definition, i.e. namespace, channel name
454 and description.

455 2.1.2: The Functional Element checks whether the channel exists.

456 2.1.3: The Functional Element creates the channel.

457 2.2: Retrieve Channel.

458 2.2.1: The Functional Element gets namespace, channel name and retrieve condition.
459 2.2.2: The Functional Element retrieves the channel's information according to the
460 condition.
461 2.3: Update Channel.
462 2.3.1: The Functional Element gets namespace, channel name and description.
463 2.3.2: The Functional Element checks whether the channel exists.
464 2.3.3: The Functional Element updates the channel definition.
465 2.4: Delete Channel.
466 2.4.1: The Functional Element gets namespace and channel name.
467 2.4.2: The Functional Element checks whether the channel exists.
468 2.4.3: The Functional Element removes the channel from the Functional Element.
469 3: The Functional Element returns the results of the operation to the user and the use case ends.

470 **2.1.7.2.2.2 Alternative Flows**

471 1: Channel Already Exists.
472 1.1: If in the basic flow 2.1.2, the channel is already defined, Functional Element returns an
473 error message and the use case ends.
474 2: Conditional Retrieving.
475 2.1: In the basic flow 2.2.2:
476 2.1 1: If the condition is the retrieval by channel name and the channel does not exist,
477 then it will go to Alternative Flow 3.
478 2.1.2: If the condition is the retrieval of one channel definition, it returns the definition of
479 that channel and the use case ends.
480 2.1.3: If the condition is the retrieval of all channels' information, it returns all channels
481 definition and the use case ends.
482 2.1.4: If the condition is the retrieval of channel through channel description, it will return
483 all matched channels and the use case ends.
484 2.1.5: If the condition is the retrieval of registered consumers, it returns the list of
485 consumer registered on the channel and the use case ends.
486 3: Channel Not Found.
487 3.1: If in the basic flow 2.2.2, 2.3.2 and 2.4.2, the channel does not exist, Functional
488 Element will return an error message and the use case ends.
489 4: Consumer Not Found.
490 4.1: If in the basic flow 2.1.3, 2.5.3 and 2.6.3, the event consumer does not exist,
491 Functional Element will return an error message and the use case ends.
492 5: Extension Point.

493 5.1: If in the basic flow 2.1.3, and 2.3.3, the event consumers that subscribed to the
494 channel are provided, the use case Subscribe/un-subscribe channel will be extended.

495 **2.1.7.2.3 Special Requirements**

496 None.

497 **2.1.7.2.4 Pre-Conditions**

498 None.

499 **2.1.7.2.5 Post-Conditions**

500 None.

501 **2.1.7.3 Subscribe/Un-subscribe To Channel**

502 **2.1.7.3.1 Description**

503 This use case performs the subscription or un-subscription on a channel for an event consumer.

504 **2.1.7.3.2 Flow of Events**

505 **2.1.7.3.2.1 Basic Flow**

506 The use case begins when the user wants to subscribe or un-subscribe to a channel.

507 1: The user sends the request.

508 2: Based on the operation it specifies, one of the following sub-flows is executed:

- 509 • If the operation is '**Subscribe to Channel**', then sub-flow 2.1 is executed.
- 510 • If the operation is '**Un-Subscribe to Channel**', then sub-flow 2.2 is executed.

511 2.1: Subscribe To Channel.

512 2.1.1: The Functional Element gets event consumer Id, or consumer namespace and
513 consumer name, together with channel namespace and channel name.

514 2.1.2: The Functional Element checks whether the channel exists.

515 2.1.3: The Functional Element adds the subscription of the consumer to the channel.

516 2.2: Un-Subscribe To Channel.

517 2.2.1: The Functional Element gets event consumer Id, or consumer namespace and
518 consumer name, together with channel namespace and channel name.

519 2.2.2: The Functional Element checks whether the channel exists.

520 2.2.3: The Functional Element removes the subscription of the consumer to the channel.

521 3: The Functional Element returns the results of the operation to the user and the use case ends.

522 **2.1.7.3.2.2 Alternative Flows**

523 1: Channel Not Found.

524 1.1: If in the basic flow 2.1.2 and 2.2.2, the channel specified does not exist, Functional
525 Element will return an error message to the user and the use case ends.

526 2: Event Consumer Not Found.

527 2.1: If in the basic flow 2.1.2 and 2.2.2, the event consumer related does not exist, Functional
528 Element will return an error message to the user and the use case ends.

529 **2.1.7.3.3 Special Requirements**

530 None.

531 **2.1.7.3.4 Pre-Conditions**

532 None.

533 **2.1.7.3.5 Post-Conditions**

534 None.

535 **2.1.7.4 Manage Event**

536 **2.1.7.4.1 Description**

537 This use case describes the scenarios of managing events.

538 **2.1.7.4.2 Flow of Events**

539 **2.1.7.4.2.1 Basic Flow**

540 The use case begins when the user wants to manage events.

541 1: The user sends a request to the Functional Element.

542 2: Based on the operation it specifies, one of the following sub-flows is executed:

543 • If the operation is **'Create Event'**, then sub-flow 2.1 is executed.

544 • If the operation is **'Retrieve Event Information'**, then sub-flow 2.2 is executed.

545 • If the operation is **'Update Event Definition'**, then sub-flow 2.3 is executed.

546 • If the operation is **'Delete Event'**, then sub-flow 2.4 is executed.

547 • If the operation is **'Assign Flow'**, then sub-flow 2.5 is executed.

548 • If the operation is **'Un-Assign Flow'**, then sub-flow 2.6 is executed.

549 2.1: Create Event

550 2.1.1: The Functional Element gets event definition including namespace, event name,
551 event description, event routing rule, and event attributes definition.

552 2.1.2: The Functional Element verifies the parameters.

553 2.1.3: The Functional Element verifies the routing rule through use case verify routing
554 rule.

555 2.1.4: The Functional Element creates event definition by recording the definition of
556 event.

- 557 2.2: Retrieve Event.
- 558 2.2.1: The Functional Element gets namespace, event name, and condition.
- 559 2.2.2: The Functional Element retrieves the event definition according to the condition.
- 560 2.3: Update Event Definition
- 561 2.3.1: The Functional Element gets event definition including namespace, event name,
- 562 event description, event routing rule, and event attributes definition.
- 563 2.3.2: The Functional Element verifies the parameters.
- 564 2.3.3: The Functional Element verifies the routing rule through use case verify routing
- 565 rule.
- 566 2.3.4: The Functional Element updates the event definition.
- 567 2.4: Delete Event.
- 568 2.4.1: The Functional Element gets namespace and event name.
- 569 2.4.2: The Functional Element checks whether the event exists.
- 570 2.4.3: The Functional Element deletes the event definition.
- 571 2.5: Assign Flow.
- 572 2.5.1: The Functional Element gets namespace, event name and flow name.
- 573 2.5.2: The Functional Element checks whether the event exists and flow defined.
- 574 2.5.3: The Functional Element assigns the flow to the event.
- 575 2.6: Un-assign Flow.
- 576 2.6.1: The Functional Element gets namespace, event name and flow name.
- 577 2.6.2: The Functional Element checks whether the event exists and flow defined.
- 578 2.6.3: The Functional Element un-assigns the flow to the event.
- 579 3: The Functional Element returns the results of the operation to the user and the use case ends.

580 **2.1.7.4.2.2 Alternative Flows**

- 581 1: Event Already Exist.
- 582 1.1: If in the basic flow 2.1.2, the event already exists, Functional Element will return an error
- 583 message to the user and the use case ends.
- 584 2: Parameters Are Invalid.
- 585 2.1: If in the basic flow 2.1.2 and 2.3.2, the parameters provided are invalid, Functional
- 586 Element will return an error message to the user and the use case ends.
- 587 3: Event Not Found.
- 588 3.1: If in the basic flow 2.2.2, 2.3.2 and 2.4.2, the event does not exist, Functional Element
- 589 will return an error message to the user and the use case ends.

590 4: Flow Not Defined.

591 4.1: If in the basic flow 2.1.2 and 2.3.2, the flow does not exist, Functional Element will return
592 an error message to the user and the use case ends.

593 5: Condition Retrieve.

594 5.1: In the basic flow 2.2.2:

595 5.1.1: If the retrieving condition is the retrieval of event definition based on event name, it
596 returns event definition and the use case ends.

597 5.1.2: If the retrieving condition is the retrieval of all event definition, it returns all event
598 definition and the use case ends.

599 5.1.3: If the retrieving condition is the retrieval of events assigned to specified channel, it
600 returns the list of event definitions.

601 5.1.4: If the retrieving condition is the retrieval of channels associated with specified
602 event, it returns the list of channel definition.

603 6: Extension Point.

604 6.1: If in the basic flow 2.1.4, and 2.3.4, the event consumers that subscribed to the event are
605 provided, the use case Subscribe/Un-subscribe event will be extended.

606 **2.1.7.4.3 Special Requirements**

607 None.

608 **2.1.7.4.4 Pre-Conditions**

609 None.

610 **2.1.7.4.5 Post-Conditions**

611 None.

612 **2.1.7.5 Subscribe/Un-subscribe To Event**

613 **2.1.7.5.1 Description**

614 This use case performs the subscription or un-subscription on an event for an event consumer.

615 **2.1.7.5.2 Flow of Events**

616 **2.1.7.5.2.1 Basic Flow**

617 The use case begins when the user wants to subscribe or un-subscribe an event.

618 1: The user sends a request.

619 2: Based on the operation it specifies, one of the following sub-flows is executed:

- 620 • If the operation is '**Subscribe to Event**', then sub-flow 2.1 is executed.
- 621 • If the operation is '**Un-Subscribe to Event**', then sub-flow 2.2 is executed.

622 2.1: Subscribe To Event.

623 2.1.1: The Functional Element gets event consumer Id, or consumer namespace and
624 consumer name, together with event namespace and event name.

625 2.1.2: The Functional Element checks whether the event exists.

626 2.1.3: The Functional Element adds the subscription of the consumer to the event.

627 2.2: Un-Subscribe To Event.

628 2.2.1: The Functional Element gets event consumer Id, or consumer namespace and
629 consumer name, together with event namespace and event name.

630 2.2.2: The Functional Element checks whether the event exists.

631 2.2.3: The Functional Element removes the subscription of the consumer to the event.

632 3: The Functional Element returns the results of the operation to the user and the use case ends.

633 **2.1.7.5.2.2 Alternative Flows**

634 1: Event Not Found.

635 1.1: If in the basic flow 2.1.2 and 2.2.2, the event specified does not exist, Functional Element
636 will return an error message to the user and the use case ends.

637 2: Event Consumer Not Found.

638 2.1: If in the basic flow 2.1.2 and 2.2.2, the event consumer related does not exist, Functional
639 Element will return an error message to the user and the use case ends.

640 **2.1.7.5.3 Special Requirements**

641 None.

642 **2.1.7.5.4 Pre-Conditions**

643 None.

644 **2.1.7.5.5 Post-Conditions**

645 None.

646 **2.1.7.6 Verify Routing Rule**

647 **2.1.7.6.1 Description**

648 This use case verifies the syntax of routing rule.

649 **2.1.7.6.2 Flow of Events**

650 **2.1.7.6.2.1 Basic Flow**

651 The use case begins when the user wants to verify the correctness of a routing expression.

652 1: The user sends a request.

653 2: The Functional Element gets the routing expression.

- 654 3: The Functional Element checks the syntax of routing expression.
655 4: The Functional Element verifies the parameters.
656 5: The Functional Element returns the status of the operation to the user and the use case ends.

657 **2.1.7.6.2 Alternative Flows**

658 1: Routing Rule Expression Syntax Error.

659 1.1: If in the basic flow 3, there is a syntax error, Functional Element will return an error
660 message to the user and the use case ends.

661 2: Event Consumer Not Found.

662 2.1: If in the basic flow 4, the event consumer related does not exist, Functional Element will
663 return an error message to the user and the use case ends.

664 **2.1.7.6.3 Special Requirements**

665 None.

666 **2.1.7.6.4 Pre-Conditions**

667 None.

668 **2.1.7.6.5 Post-Conditions**

669 None.

670 **2.1.7.7 Manage Filter**

671 **2.1.7.7.1 Description**

672 A filter is used to filter out certain events to those event consumers even though they are the
673 intended receivers according to the routing rules.

674 **2.1.7.7.2 Flow of Events**

675 **2.1.7.7.2.1 Basic Flow**

676 The use case begins when the user wants to create/retrieve/update/delete a filter.

677 1: The user sends a request to manage a filter.

678 2: Based on the operation it specifies, one of the following sub-flows is executed:

- 679 • If the operation is '**Create Filter**', then sub-flow 2.1 is executed.
- 680 • If the operation is '**Retrieve Filter**', then sub-flow 2.2 s executed.
- 681 • If the operation is '**Update Filter**', then sub-flow 2.3 is executed.
- 682 • If the operation is '**Delete Filter**', then sub-flow 2.4 is executed.

683 2.1: Create Filter.

684 2.1.1: The Functional Element gets filter definition, i.e. consumer namespace, consumer
685 name, filter name, description, event name or channel name.

686 2.1.2: The Functional Element checks whether the event or channel exists.
687 2.1.3: The Functional Element saves the filter definition.
688 2.2: Retrieve Filter.
689 2.2.1: The Functional Element gets the filter name.
690 2.2.2: The Functional Element retrieves the filter information according to the name.
691 2.3: Update Filter.
692 2.3.1: The Functional Element gets filter definition, i.e. consumer namespace, name, filter
693 name, description, event name or channel name.
694 2.3.2: The Functional Element checks the parameters.
695 2.3.3: The Functional Element updates the filter definition.
696 2.4: Delete Filter.
697 2.4.1: The Functional Element gets namespace and filter name.
698 2.4.2: The Functional Element checks whether the filter exists.
699 2.4.3: The Functional Element removes the filter from the Functional Element.
700 3: The Functional Element returns the results of the operation to the user and the use case ends.

701 **2.1.7.7.2 Alternative Flows**

702 1: Filter Already Exists.
703 1.1: If in the basic flow 2.1.2, the filter is already defined, Functional Element will return an
704 error message and the use case ends.
705 2: Event Not Found.
706 2.1: If in the basic flow 2.1.2 and 2.3.2, the event used does not exist, Functional Element will
707 return an error message and the use case ends.
708 3: Channel Not Found.
709 3.1: If in the basic flow 2.1.2 and 2.3.2, the channel used does not exist, Functional Element
710 will return an error message and the use case ends.
711 4: Consumer Not Found.
712 4.1: If in the basic flow 2.1.3, 2.5.3, and 2.6.3, the event consumer does not exist, Functional
713 Element will return an error message and the use case ends.

714 **2.1.7.7.3 Special Requirements**

715 None.

716 **2.1.7.7.4 Pre-Conditions**

717 None.

718 **2.1.7.7.5 Post-Conditions**

719 None.

720 **2.1.7.8 Notify Event**

721 **2.1.7.8.1 Description**

722 This use case allows the event supplier to notify an event to the Event Handler Functional
723 Element. Once the Event Handler Functional Element receives the notification, it will process the
724 event based on the processing logic defined.

725 **2.1.7.8.2 Flow of Events**

726 **2.1.7.8.2.1 Basic Flow**

727 The use case begins when the user wants to notify an event.

728 1: The user sends a notification.

729 2: The Functional Element receives the notification with parameters, i.e. event supplier id or event
730 supplier namespace and name.

731 3: The Functional Element checks whether the event is defined and event supplier is registered.

732 4: Include use case Process Event to process the notification of event.

733 5: The Functional Element returns the status of the operation to the user and the use case ends.

734 **2.1.7.8.2.2 Alternative Flows**

735 1: User Is Not Registered.

736 1.1: If in the basic flow 3, the user is not registered, Functional Element will return an error
737 message to the user and the use case ends.

738 2: Event Not Defined.

739 2.1: If in the basic flow 3, the event is not defined, Functional Element will return an error
740 message to the user and the use case ends.

741 3: Error Returned.

742 3.1: If in the basic flow 4, an error is returned by use case Process event, Functional Element
743 will return an error message to the user and the use case ends.

744 **2.1.7.8.3 Special Requirements**

745 None.

746 **2.1.7.8.4 Pre-Conditions**

747 None.

748 **2.1.7.8.5 Post-Conditions**

749 None.

750 **2.1.7.9 Configure Monitoring**

751 **2.1.7.9.1 Description**

752 This use case describes the capability of configuration on event monitoring. Based on the
753 configuration, Event Handler will pro-actively check whether an event has happened.

754 **2.1.7.9.2 Flow of Events**

755 **2.1.7.9.2.1 Basic Flow**

756 The use case begins when the user wants to configure the event monitoring.

757 1: The user sends a request to manage a filter.

758 2: Based on the operation it specifies, one of the following sub-flows is executed:

- 759 • If the operation is '**Add Configuration**', then sub-flow 2.1 is executed.
- 760 • If the operation is '**Remove Configuration**', then sub-flow 2.2 is executed.

761 2.1: Add Configuration.

762 2.1.1: The Functional Element gets configuration definition, i.e. configuration name,
763 namespace, event name, connection parameters, condition that signifies the events and
764 schedule.

765 2.1.2: The Functional Element saves filter definition.

766 2.2: Remove Configuration.

767 2.2.1: The Functional Element gets configuration name.

768 2.2.2: The Functional Element removes the configuration.

769 3: The Functional Element returns the results of the operation to the user and the use case ends.

770 **2.1.7.9.2.2 Alternative Flows**

771 1: Configuration Exist.

772 1.1: If in the basic flow 2.1.2, the configuration already exists, Functional Element will return
773 an error message and the use case ends.

774 **2.1.7.9.3 Special Requirements**

775 None.

776 **2.1.7.9.4 Pre-Conditions**

777 None.

778 **2.1.7.9.5 Post-Conditions**

779 None.

780 **2.1.7.10 Detect Event**

781 **2.1.7.10.1 Description**

782 This use case describes the event monitoring capability that Event Handler provides. Once Event
783 Handler detects an event, it will trigger the pre-defined process for the event.

784 **2.1.7.10.2 Flow of Events**

785 **2.1.7.10.2.1 Basic Flow**

786 The use case begins when the Functional Element clock generates the trigger.

787 1: The Functional Element clock generates a trigger.

788 2: The Functional Element receives the trigger and checks the condition for pre-defined
789 monitoring sources.

790 3: The Functional Element checks whether the event happens.

791 4: The Functional Element returns the results of the operation and the use case ends.

792 **2.1.7.10.2.2 Alternative Flows**

793 1: External Functional Element Not Available.

794 1.1: If in the basic flow 3, the external Functional Element is not available and the Event
795 Handler cannot make a connection, Functional Element will return an error message and the
796 use case ends.

797 2: Data Not Available.

798 2.1: If in the basic flow 3, the data that signifies the event cannot be accessed, Functional
799 Element will return an error message and the use case ends.

800 3: Extension Point.

801 3.1: If in the basic flow 3, the event happens, Functional Element will extend to use case
802 Process event.

803 **2.1.7.10.3 Special Requirements**

804 None.

805 **2.1.7.10.4 Pre-Conditions**

806 None.

807 **2.1.7.10.5 Post-Conditions**

808 None.

809 **2.1.7.11 Process Event**

810 **2.1.7.11.1 Description**

811 This use case describes the core functionality of Event Handler. It is the engine that processes
812 the events. Actor can be the Functional Element clock that triggers the scheduled event
813 notification, or any user who wants to notify the event.

814 **2.1.7.11.2 Flow of Events**

815 **2.1.7.11.2.1 Basic Flow**

816 The use case begins when there is a request to process the event.

817 1: The user sends a request to process an event.

818 2: Based on the actor of this use case, one of the sub-flows is executed.

819 • If the initiator is the Functional Element clock, then sub-flow '**Initiated By Functional**
820 **Element Clock**' is executed.

821 • If the initiator is other than Functional Element clock, then sub-flow '**Initiated By Any**
822 **User**' is executed.

823 2.1: Initiated By Functional Element Clock.

824 2.1.1: The Functional Element looks up scheduled events defined to find out time-due
825 notification.

826 2.1.2: The Functional Element retrieves the routing rule for the event.

827 2.1.3: The Functional Element looks up the corresponding consumers based on the
828 routing rule.

829 2.1.4: The Functional Element retrieves filters defined and find out the event receivers.

830 2.1.5: The Functional Element notifies or invokes the event consumers based on the
831 routing rule defined.

832 2.2: Initiated By Any User.

833 2.2.1: The Functional Element retrieves the routing rule for the event.

834 2.2.2: The Functional Element looks up the corresponding consumers.

835 2.2.3: The Functional Element retrieves filters defined and find out the event receivers.

836 2.2.4: The Functional Element notifies or invokes the event consumers based on the
837 routing rule defined.

838 3: The Functional Element logs the notification of event and the use case ends.

839 **2.1.7.11.2.2 Alternative Flows**

840 1: Notify Event.

841 In basic flow 2.1.4 and 2.2.4, based on the type of consumer, one of the sub-flows is execute.

842 • If the consumer type is '**SMTP**', then sub-flow Notify via SMTP is executed.

- 843 • If the consumer type is '**SMS Gateway**', then sub-flow Notify via SMS Gateway is
844 executed.
- 845 • If the consumer type is '**Notify RPC-Web Service**', then sub-flow Notify RPC-Web
846 Service is executed.
- 847 • If the consumer type is '**Notify Document Style Web Service**' then sub-flow Notify
848 Document style Web Service is executed.
- 849 1.1: Notify via SMTP.
- 850 1.1.1: The Functional Element gets the pre-defined message for event and forms the
851 parameters.
- 852 1.1.2: The Functional Element gets the parameters for SMTP server.
- 853 1.1.3: The Functional Element sends out the pre-defined message and the use case
854 ends.
- 855 1.2: Notify via SMS Gateway.
- 856 1.2.1: The Functional Element gets the pre-defined message for event and forms the
857 parameters.
- 858 1.2.2: The Functional Element gets the parameters for the SMS gateway.
- 859 1.2.3: The Functional Element sends out the pre-defined message and the use case
860 ends.
- 861 1.3: Notify RPC-Web Service.
- 862 1.3.1: The Functional Element gets the operation parameter.
- 863 1.3.2: The Functional Element gets Web Services endpoint parameters.
- 864 1.3.3: The Functional Element dynamically invokes the Web Service and the use case
865 ends.
- 866 1.4: Notify Document Style Web Service.
- 867 1.4.1: The Functional Element gets the operation parameter.
- 868 1.4.2: The Functional Element gets Web Services endpoint parameters.
- 869 1.4.3: The Functional Element dynamically generates the SOAP message and sends to
870 the Web Services and the use case ends.
- 871 2: Flow Is Defined.
- 872 If in the basic flow 2.1.2 and 2.2.1, a flow is defined for the event, Functional Element will perform
873 the following steps:
- 874 2.1: The Functional Element retrieves all the intended event consumers defined in the flow.
- 875 2.2: The Functional Element will go to basic flow 2.2.
- 876 2.3: The Functional Element will resume the execution from basic flow 2.1.2 or 2.2.1.
- 877 3: Log Utility Not Available.

878 3.1: If in the basic flow 3, the Log Utility Functional Element is not available, Functional
879 Element will return an error message to the user and the use case ends.

880 4: SMS Gateway Not Available.

881 4.1: If in the Alternative Flow 1.2.3, the SMS Gateway is not available, Functional Element will
882 return an error message to the user and the use case ends.

883 5: SMTP Server Not Available.

884 5.1: If in the Alternative Flow 1.1.3, the SMTP server is not available, Functional Element will
885 return an error message to the user and the use case ends.

886 6: RPC Web Service Not Available.

887 6.1: If in the Alternative Flow 1.3.3, the Web Service is not available, Functional Element will
888 return an error message to the user and the use case ends.

889 7: Document Style Web Service Not Available.

890 7.1: If in the Alternative Flow 1.4.3, document style Web Service is not available, Functional
891 Element will return an error message to the user and the use case ends.

892 **2.1.7.11.3 Special Requirements**

893 **2.1.7.11.3.1 Supportability**

894 The application server used must have a JMS service provided.

895 **2.1.7.11.4 Pre-Conditions**

896 None.

897 **2.1.7.11.5 Post-Conditions**

898 None.

899

900

901

902

903

904 2.2 Group Management Functional Element

905 2.2.1 Motivation

906 The Group Management Functional Element is expected to be an integral part of the User Access
907 Management (UAM) functionalities. In a Web Service-enabled implementation, this Functional
908 Element helps to provide the mechanism to manage users in a collective manner. This is
909 important as it provides the flexibility of adopting either coarse or fine-grain access controls, or
910 both.

911

912 This Functional Element fulfills the following requirements from the Functional Elements
913 Requirements, Working Draft 01a:

- 914 • Primary Requirements
 - 915 • MANAGEMENT-050 to MANAGEMENT-053, and
 - 916 • MANAGEMENT-078
- 917 • Secondary Requirements
 - 918 • None

919 2.2.2 Terms Used

Terms	Description
Group	A Group is a collection of individual users, and are typically grouped together as they have certain commonalities
Namespace	Namespace is use to segregate the instantiation of the application across different application domains. If a company has two separate standalone application, for example, an email application and an equipment booking application, then these two are considered as separate application domains.
User	A user is loosely defined to include both human and virtual users. Virtual users could include service users and application (or machine) users that are utilising other services in a SOA environment.
User Access Management / UAM	User Access Management or UAM refer to the concept of managing users in a holistic manner, considering all aspect which includes: <ul style="list-style-type: none">• Defining a set of basic user information that should be stored in any enterprise application.• Providing a means to extend this basic set of user information when needed.• Simplifying management by grouping related users together through certain criteria.• Having the flexibility of adopting both coarse and fine grain access controls.

920

921 **2.2.3 Key Features**

922 Implementations of the Group Management Functional Element are expected to provide the
923 following key features:

- 924 1. The Functional Element MUST provide a basic Group structure with a set of pre-defined
925 attributes.
- 926 2. The Functional Element MUST provide the capability to extend on the basic Group structure
927 dynamically.
- 928 3. As part of Key Feature (2), this dynamic extension MUST be definable and configurable at
929 runtime implementation of the Functional Element.
- 930 4. The Functional Element MUST provide the capability to manage the creation and deletion of
931 instances of Groups based on defined structure.
- 932 5. The Functional Element MUST provide the capability to manage all the information (attribute
933 values) stored in such Groups. This includes the capability to retrieve and update attribute's
934 values belonging to a Group.
- 935 6. The Functional Element MUST provide a mechanism to manage the collection of users in a
936 Group. This includes the capability to create, retrieve, update and delete users belonging to
937 a Group.
- 938 7. The Functional Element MUST provide a mechanism for managing Groups across different
939 application domains.

940 *Example: Namespace control mechanism*

941

942 In addition, the following key features could be provided to enhance the Functional Element
943 further:

- 944 1. The Functional Element MAY provide a mechanism to enable different Groups to be related
945 to one another.
- 946 2. The Functional Element MAY also provide a mechanism to enable hierarchical relationships
947 between Groups.
948 *Example: Parent and Child Relationship.*
- 949 3. As an extension of Key Feature (2), the Functional Element MAY also provide the capability
950 to enable Groups to be part of the collection of "users" of another Group.
951 *Example: Adding of Group "Dept-A" to "Company-XYZ" – "Dept-A" is a Group, and also part
952 of the collection of Group "Company-XYZ".*

- 953 4. The Functional Element MAY provide validity checks when managing information stored in a
954 Group.

955 *Example: Adding of User "john" – A validity check could be imposed to ensure that a user
956 "john" exists before adding to into the Group.*

957

958 **2.2.4 Interdependency**

Direct Dependency	
User Management Functional Element	The User Management Functional Element is used to manage the user's attributes. The Group Management Functional Element in turn provides useful aggregation of the users. Together, they are able to achieve effective and efficient management of user information.

959

960 **2.2.5 Related Technologies and Standards**

961 None.

962

963 **2.2.6 Model**

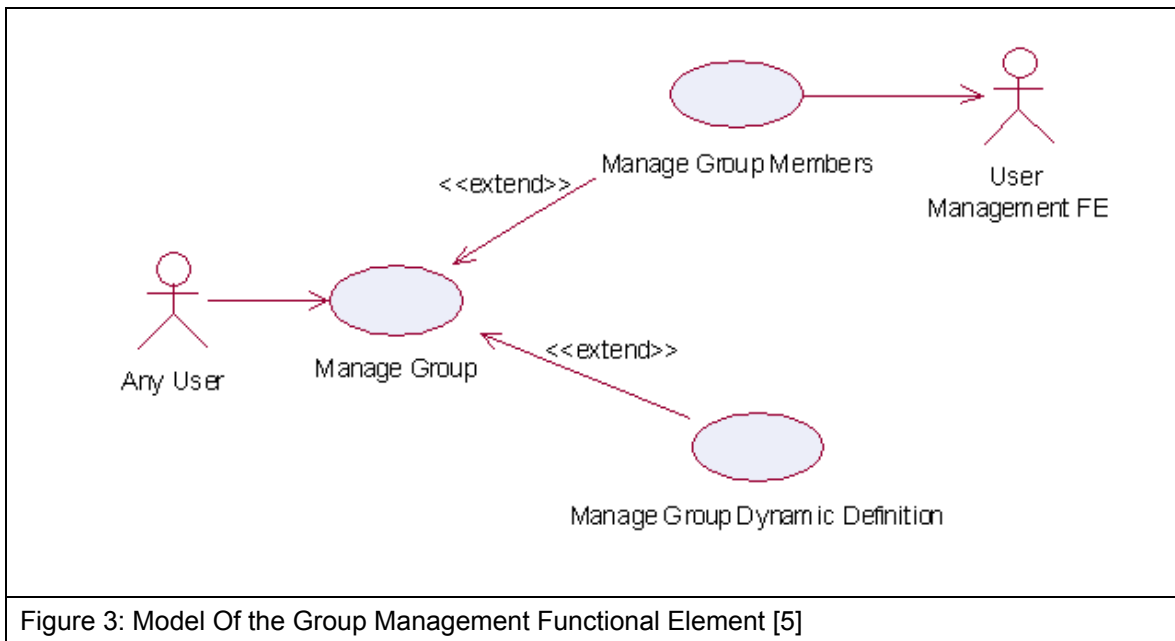


Figure 3: Model Of the Group Management Functional Element [5]

964

965 **2.2.7 Usage Scenarios**

966 **2.2.7.1 Manage Group**

967 This use case describes the management of a group, namely the creation, deletion, retrieval and
968 update of the group.

969 **2.2.7.1.1 Flow of Events**

970 **2.2.7.1.1.1 Basic Flow**

971 This use case starts when the user wants to manage group.

- 972
- If user wants to '**Create Group**', then basic flow 1 is executed.
 - 973 • If user wants to '**Retrieve Group**', then basic flow 2 is executed.
 - 974 • If user wants to '**Update Group**', then basic flow 3 is executed.
 - 975 • If user wants to '**Delete Group**', then basic flow 4 is executed.

976 1: Create Group.

977 1.1: User provides the basic information that is necessary for creating a group.

978 1.2: Functional Element creates the group and the use case ends.

- 979 2: Retrieve Group.
- 980 2.1: User provides the necessary information for retrieving the complete group's attributes.
- 981 2.2: Functional Element returns the group's information and the use case ends.
- 982 3: Update Group.
- 983 3.1: User provides the necessary information for updating the group's attributes.
- 984 3.2: Functional Element updates the group and the use case ends.
- 985 4: Delete Group.
- 986 4.1: User provides the necessary information for deleting a particular group.
- 987 4.2: Functional Element deletes the group and the use case ends.

988 **2.2.7.1.1.2 Alternative Flows**

- 989 1: Group Exist.
- 990 1.1: In basic flow 1.2, Functional Element detects an identical group. Functional Element
991 returns an error message and the use case ends.
- 992 2: Group Does Not Exist.
- 993 2.1: In basic flow 2.2, 3.2 and 4.2, Functional Element cannot find a group that matches the
994 user's criteria. Functional Element returns an error message and the use case ends.
- 995 3: Save Updated Information.
- 996 3.1: In basic flow 1.2, 2.2, 3.2 and 4.2, Functional Element fails to save the updated
997 information. Functional Element returns an error message and the use case ends.

998 **2.2.7.1.2 Special Requirements**

999 None.

1000 **2.2.7.1.3 Pre-Conditions**

1001 None.

1002 **2.2.7.1.4 Post-Conditions**

1003 None.

1004 **2.2.7.2 Manage Group Members**

1005 **2.2.7.2.1 Description**

1006 This use case is an extension of the manage group use case. Specifically, it describes the
1007 scenarios to manage members in the group.

1008 **2.2.7.2.2 Flow of Events**

1009 **2.2.7.2.2.1 Basic Flow**

1010 This use case starts when the user wants to manage members in a group.

- 1011 • If user wants to 'Create Members In A Group', then basic flow 1 is executed.
- 1012 • If user wants to 'Retrieve Members From A Group', then basic flow 2 is executed.
- 1013 • If user wants to 'Delete Members From A Group', then basic flow 3 is executed.

1014 1: Create Members In A Group.

1015 1.1: User provides the necessary information for retrieving the group.

1016 1.2: Functional Element adds members to the group and the use case ends.

1017 2: Retrieve Members In A Group.

1018 2.1: User provides the necessary information for retrieving the group.

1019 2.2: Functional Element returns the members and the use case ends.

1020 3: Delete Members From Group.

1021 3.1: User provides the necessary information for retrieving the group.

1022 3.2: User provides the necessary information for deleting members in the group.

1023 3.3: Functional Element deletes members from group and the use case ends.

1024 **2.2.7.2.2.2 Alternative Flows**

1025 1: Group Does Not Exist.

1026 1.1: In basic flow 1.1, 2.1 and 3.1, Functional Element cannot find the group requested.
1027 Functional Element returns an error message and the use case ends.

1028 2: Members Does Not Exist

1029 2.1: In basic flow 3.3, the Functional Element attempts to delete a non-existence member.
1030 Functional Element returns an error message and the use case ends.

1031 **2.2.7.2.3 Special Requirements**

1032 None.

1033 **2.2.7.2.4 Pre-Conditions**

1034 None.

1035 **2.2.7.2.5 Post-Conditions**

1036 None.

1037 **2.2.7.3 Manage Group Dynamic Definition**

1038 **2.2.7.3.1 Description**

1039 This use case describes scenario involved in managing the dynamic group definition.

1040 **2.2.7.3.2 Flow of Events**

1041 **2.2.7.3.2.1 Basic Flow**

1042 This use case starts when the user wants to manage dynamic group definition. This include
1043 create, retrieve, update and delete dynamic group definition.

- 1044 • If user wants to '**Create Dynamic Definition For A Group**', then basic flow 1 is
1045 executed.
- 1046 • If user wants to '**Retrieve Dynamic Definition For A Group**', then basic flow 2 is
1047 executed.
- 1048 • If user wants to '**Delete Dynamic Definition For A Group**', then basic flow 3 is
1049 executed.
- 1050 • If user wants to '**Update Dynamic Definition For A Group**', then basic flow 4 is
1051 executed.

1052

1053 1: Create Dynamic Definition For A Group.

1054 1.1: User provides the additional definition for the group.

1055 1.2: Functional Element creates the additional definition for the group and the use case ends.

1056 2: Retrieve Dynamic Definition For A Group.

1057 2.1: User provides the necessary information to retrieve a particular group.

1058 2.2: Functional Element returns the additional definition for the group and the use case ends.

1059 3: Delete Dynamic Definition For Group.

1060 3.1: User provides the necessary information to retrieve a particular group.

1061 3.2: Functional Element deletes the dynamic definition belonging to the group and the use
1062 case ends.

1063 4: Update Dynamic Definition For Group.

1064 4.1: User provides the necessary information to retrieve a particular group.

1065 4.2: User provides the necessary dynamic definition that needs to be updated.

1066 4.3: Functional Element update the dynamic definition and the use case ends.

1067 **2.2.7.3.2.2 Alternative Flows**

1068 1: Group Does Not Exist.

1069 1.1: In basic flow 1.1, 2.1, 3.1 and 4.1, Functional Element cannot find the group specified.
1070 Functional Element returns an error message and the use case ends.

- 1071 2: Dynamic Group Definition Already Exists.
- 1072 2.1: In basic flow 1.2, Functional Element returns the error message and the use case ends.
- 1073 3: Dynamic Group Definition Does Not Exist.
- 1074 3.1: In basic flow 4.3, Functional Element cannot update the dynamic group definition.
- 1075 Functional Element returns an error message and the use case ends.

1076 **2.2.7.3.3 Special Requirements**

1077 None.

1078 **2.2.7.3.4 Pre-Conditions**

1079 None.

1080 **2.2.7.3.5 Post-Conditions**

1081 None.

1082 2.3 Identity Management Functional Element

1083 2.3.1 Motivation

1084 As secured Web Services become rampant, with each having its own authentication and
1085 authorisation management, users are finding it difficult to keep track of their accounts and
1086 passwords. Through the use of Identity Management, users can now voluntarily establish links
1087 between their accounts so that they need not sign in multiple times to access enterprise-level
1088 Web Services. This mechanism is known as Single Sign-On (SSO). SSO can further be extended
1089 to access Web Services from across different business organisations that have prior agreements
1090 to trust and transact with each other (also known as a circle of trust). This mechanism, which
1091 involves federating and signing-in of identity's accounts across different trusted organisations, is
1092 known as Federated Identity Single Sign-On.

1093

1094 Identity Management is about the management of information pertaining to an entity as well as
1095 the process of identification, authentication and authorization of resources to that entity.

1096

1097 Identity management generally covers the following aspects:

- 1098 • Basic user accounts management facilities
- 1099 • User authentication mechanism(s)
- 1100 • User authorisation mechanism(s)
- 1101 • Generation of audit trails for user activities

1102

1103 This Functional Element fulfills the following requirements from the Functional Elements
1104 Requirements, Working Draft 01a:

- 1105 • Primary Requirements
 - 1106 • SECURITY-001,
 - 1107 • SECURITY-003 (all),
 - 1108 • SECURITY -004 (all),
 - 1109 • SECURITY -040 and
 - 1110 • SECURITY -041.
- 1111 • Secondary Requirements
 - 1112 • None

1113

1114 2.3.2 Terms Used

Terms	Description
Assertion	Assertion refers to a piece of data produced by an Assertion Authority regarding either an act of authentication performed on a subject, attribute information about a subject, or authorization permissions applying to the subject with respect to a specified resource.

Assertion Authority	An entity within a trusted circle that provides authentication assertions.
Access Policy	A logically defined, executable and testable set of rules or behavior for access control.
Entity	Entity can refer to a person, an organization, a resource or a service.
Federated Identity	An identity that has been associated, connected or binded with other accounts for a same given Principal.
Identity	Identity refers to a set of information that an entity can use to uniquely describe itself.
Identity Provider	An entity that creates, maintains, and manages identity information for Principals and provides Principal authentication to other service providers within a trusted circle.
Identity Repository	Identity Repository refers to the storage of the identity information. Common examples of identity repositories are relational databases, text files etc.
Principal	Principal refers to an entity whose identity can be authenticated. Also known as Subject.
Resource	A resource in an application is defined to encompass users, services, data / information, transaction and security
Security Markup Assertion Language	Security Markup Assertion Language refers to the set of specifications describing assertions that are encoded in XML, profiles for attaching the assertions to various protocols and frameworks, the request/response protocol used to obtain assertions, and bindings of this protocol to various transfer protocols (for example, SOAP and HTTP).
Single Sign-On (SSO)	The ability to use proof of an existing authentication session with an identity provider to create authenticated sessions with other service providers.
Subject	Subject – see Principal.

1115

1116 The following terms mentioned in this document are used in accordance with the terms defined in
1117 the Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) v1.1
1118 specification.

- 1119 • Assertion [section 2.3.2]
- 1120 • AudienceRestrictionCondition [section 2.3.2.1.3]
- 1121 • AuthenticationQuery [section 3.3.3]
- 1122 • AuthenticationStatement [section 2.4.3]
- 1123 • KeyInfo [section 5.4.5]
- 1124 • Request [section 3.2.2]
- 1125 • Response [section 3.4.2]

1126 • Subject [section 2.4.2.1]

1127

1128 2.3.3 Key Features

1129 Implementations of the Identity Management Functional Element are expected to provide the
1130 following key features:

- 1131 1. The Functional Element MUST be have the mechanism to access an Identity Repository.
- 1132 2. The Functional Element MUST provide the capability to manage the creation and deletion of
1133 instances of Identity in the said Identity Repository.
- 1134 3. The Functional Element MUST have the mechanisms to manage all the information
1135 (attribute values) stored in such Identities. This includes the capability to:
 - 1136 3.1. Retrieve and update attribute's values belonging to a Identity,
 - 1137 3.2. Encrypt sensitive user information,
 - 1138 3.3. Authenticate a user, and
 - 1139 3.4. Assign/Unassign Access Policy (or Policies).
- 1140 *Example: Different levels of privileges to access protected resources.*
- 1141 4. As part of Key Feature (3.3), the authentication of an Identity MUST be achieved at least
1142 through the use of a password.
- 1143 5. As part of Key Feature (3.3), the Functional Element MUST also provide the capability to
1144 use an Assertion Authority for Single Sign-On (SSO) authentication.
- 1145 6. As part of Key Feature (5), the SSO message exchange and protocol MUST use an
1146 approved standard.
- 1147 7. As part of Key Feature (3.4), a mechanism MUST be provided to verify the Identity's Access
1148 Policy on protected Resources.
- 1149 8. The Functional Element MUST provide the capability to create audit trails.
1150 *Example: Timestamp of an Identity's access to Resources.*

1151

1152 In addition, the following key features could be provided to enhance the Functional Element
1153 further:

- 1154 1. The Functional Element MAY provide an Identity Repository.
- 1155 2. If Key Feature (1) is provided, the Functional Element MUST provide the capability to
1156 manage the creation and deletion of instances of Identities based on a pre-defined structure.
- 1157 3. The Functional Element MAY provide additional storage in the Identity Repository for an
1158 Identity to customise its preferences.
1159 *Example: Identity's preferred subscription of notifications/alerts for news.*
- 1160 4. The Functional Element MAY provide a capability to use an Identity Provider for Federated
1161 Identity SSO authentication.
- 1162 5. If Key Feature (4) is provided, the Federated Identity SSO message exchange and protocol
1163 MUST use an approved standard.

1164

1165 2.3.4 Interdependencies

Direct Dependencies

User Management Functional Element	The User Management Functional Element is being used for account management.
------------------------------------	--

Role and Access Management Functional Element	The Role and Access Management Functional Element is being used for access control and authorization
Log Utility Functional Element	The Log Utility Functional Element is being used for logging and creation of audit trails.

1166

1167 **2.3.5 Related Technologies and Standards**

Specifications	Specific References
Web Services Security v1.0 [6]	Web Services Security: SOAP Message Security 1.0 (WS-Security 2004) – OASIS Standard 200401, March 2004
Security Assertion Markup Language (SAML) v1.1. [7]	<p>Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1 – OASIS Standard, 2 September 2003</p> <p>Bindings and Profiles for the OASIS Security Assertion Markup Language (SAML) V1.1 – OASIS Standard, 2 September 2003, in particular the two schemas below:</p> <ul style="list-style-type: none"> • Assertion Schema • Protocol Schema
Liberty Alliance Project Specifications	<p>Liberty Alliance ID-FF 1.2 Specifications [8]</p> <p>Liberty Alliance ID-WSF 1.0 Specifications [9]</p>
WS-Federation [10]	Web Services Federation Language (WS-Federation) - 08 July 2003

1168

1169

2.3.6 Model

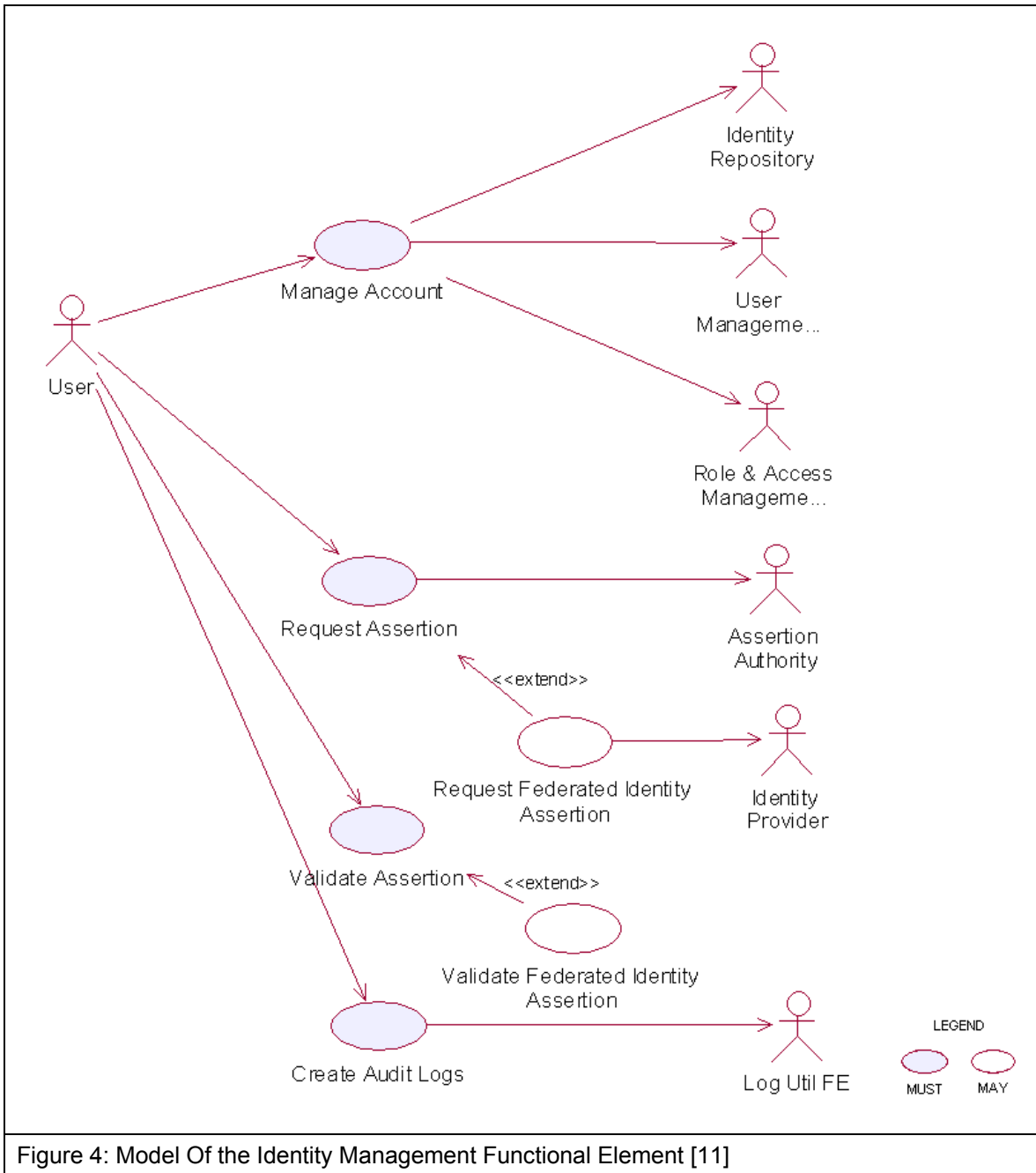


Figure 4: Model Of the Identity Management Functional Element [11]

1172 **2.3.7 Usage Scenarios**

1173 **2.3.7.1 Manage Account**

1174 **2.3.7.1.1 Description**

1175 This use case describes the creation/retrieval/update/deletion of an identity's account. An
1176 identity's account usually consists of two elements: i) the user information and ii) the associated
1177 access policy.

1178 As Identity Management Functional Element leverages on the User Management and Role-and-
1179 Access Management Functional Element to provide for these functionalities, please refer to
1180 sections 2.15 User Management Functional Element and 2.8 Role and Access Management
1181 Functional Element use cases for details.

1182 **2.3.7.2 Request Assertion**

1183 **2.3.7.2.1 Description**

1184 This use case describes the composition of either 1) an authentication query or 2) an
1185 authorisation decision query and sending it to the assertion authority.

1186 **2.3.7.2.2 Flow of Events**

1187 **2.3.7.2.2.1 Basic Flow**

1188 This use case starts when the user wants to compose a query to the assertion authority.

1189 If the user requests for an authentication query, then sub-flow 1 is executed.

1190 If the user requests for an authorisation decision query, then sub-flow 2 is executed.

1191 1: Request for Authentication Assertion

1192 1.1: The user composes a valid SAML Request with an AuthenticationQuery and sends it to
1193 the assertion authority.

1194 1.2: The user waits for an SAML Response from the assertion authority.

1195 1.3: The user obtains the SAML Assertion from the SAML Response and use case ends.

1196 2: Request for Authorisation Decision Assertion

1197 2.1: The user composes a valid SAML Request with an AuthorizationDecisionQuery and
1198 sends it to the assertion authority.

1199 2.2: The user waits for an SAML Response from the assertion authority.

1200 2.3: The user obtains the SAML Assertion from the SAML Response and use case ends.

1201 **2.3.7.2.2.2 Alternative Flows**

1202 1: Invalid Request

1203 1.1: If in basic flow 1.1 or 2.1, if any of the parameters passed into the request is invalid, the
1204 Functional Element flag an exception and use case ends.

1205 2: Error message from assertion authority

1206 2.1: If in basic flow 1.3 or 2.3, the assertion authority is unable to return an assertion (e.g.
1207 user has not logged on etc.), it returns an error code and an error message.

1208 2.2: The Functional Element flag an error with the error message attached and use case
1209 ends.

1210 **2.3.7.2.3 Special Requirements**

1211 None.

1212 **2.3.7.2.4 Pre-Conditions**

1213 None.

1214 **2.3.7.2.5 Post-Conditions**

1215 None.

1216 **2.3.7.3 Validate Assertion**

1217 **2.3.7.3.1 Description**

1218 This use case describes the validation of either 1) the Authentication Assertion or 2) the
1219 Authorisation Decision Assertion

1220 **2.3.7.3.2 Flow of Events**

1221 **2.3.7.3.2.1 Basic Flow**

1222 This use case starts when the user wants to check if the assertion it is a valid assertion from the
1223 assertion authority.

1224 1: The user passes the assertion to the Functional Element for validation.

1225 2: The Functional Element checks if the assertion is signed by the assertion authority.

1226 3: The Functional Element checks for an un-expired assertion.

1227 4: The Functional Element checks if the assertion has an AudienceRestrictionCondition and
1228 verifies that the service provider using the Functional Element is in the audience list.

1229 5: Based on the type of assertion, one of the sub-flows is executed.

1230 • If the user wants to check for a valid authentication assertion, then sub-flow 5.1 is executed.

1231 • If the user wants to check for a valid authorisation decision assertion, then sub-flow 5.2 is
1232 executed.

1233 5.1: Validate Authentication Statement

1234 5.1.1: The Functional Element checks if the assertion has indeed an
1235 AuthenticationStatement.

1236 5.1.2: The Functional Element checks if the Subject in the AuthenticationStatement
1237 matches the userid of the principal.

- 1238 5.1.3: The Functional Element verifies the Subject with its KeyInfo.
- 1239 5.1.4: The Functional Element returns the status code to the user and use case ends.
- 1240 5.2: Validate Authorisation Decision Statement
- 1241 5.2.1: The Functional Element checks if the assertion has indeed an
1242 AuthorizationDecisionStatement.
- 1243 5.2.2: The Functional Element checks if the Resource in the
1244 AuthorizationDecisionStatement matches the id of the requested resource.
- 1245 5.2.3: The Functional Element determines if the decision is Permit.
- 1246 5.2.4: The Functional Element returns the status code to the user and use case ends.
- 1247 **2.3.7.3.2.2 Alternative Flows**
- 1248 1: Signature Error
- 1249 1.1: If in basic flow 2, the Functional Element is unable to verify that the signature is from the
1250 assertion authority, it returns an error and use case ends.
- 1251 2: Expired Assertion
- 1252 2.1: If in basic flow 3, the Functional Element finds that the assertion has already expired, it
1253 returns an error and use case ends.
- 1254 3: Audience Error
- 1255 3.1: If in basic flow 4, the service provider is not in the AudienceRestrictionCondition, the
1256 Functional Element returns an error and use case ends.
- 1257 4: Invalid Authentication Assertion
- 1258 4.1: If in basic flow 5.1.1, the Functional Element is unable to find an
1259 AuthenticationStatement, it returns an error and use case ends.
- 1260 5: Mismatch Subject
- 1261 5.1: If in basic flow 5.1.2, the Functional Element is unable to match the Subject in
1262 AuthenticationStatement, it returns an error and use case ends.
- 1263 6: Subject Error
- 1264 6.1: If in basic flow 5.1.3, the Functional Element is unable to verify the Subject with the
1265 KeyInfo, it returns an error and use case ends.
- 1266 7: Invalid Authorisation Decision Assertion
- 1267 7.1: If in basic flow 5.2.1, the Functional Element is unable to find an
1268 AuthorizationDecisionStatement, it returns an error and use case ends.
- 1269 8: Mismatch Resource
- 1270 8.1: If in basic flow 5.2.2, the Functional Element is unable to match the resource in
1271 AuthorizationDecisionStatement, it returns an error and use case ends.

1272 **2.3.7.3.3 Special Requirements**

1273 None.

1274 **2.3.7.3.4 Pre-Conditions**

1275 None.

1276 **2.3.7.3.5 Post-Conditions**

1277 None.

1278 **2.3.7.4 Create Audit Logs**

1279 **2.3.7.4.1 Description**

1280 This use case describes logging all identity management activities for audit purposes.

1281 **2.3.7.4.2 Flow of Events**

1282 **2.3.7.4.2.1 Basic Flow**

1283 This use case starts when any of other Functional Element use cases are triggered.

1284 1: The Functional Element opens an audit log file.

1285 2: The Functional Element writes a timestamp identity management activity message into the
1286 audit log file.

1287 3: The Functional Element closes the audit log file and the use case ends.

1288 **2.3.7.4.2.2 Alternative Flows**

1289 1: Log File Not Created

1290 1.1: If in the basic flow 1, the Functional Element cannot open the audit file, it creates a new
1291 audit file and use case continues.

1292 2: Error Writing Log

1293 2.1: If in the basic flow 2, the Functional Element has error writing to file, it will flag an
1294 exception and the use case ends.

1295 **2.3.7.4.3 Special Requirements**

1296 None.

1297 **2.3.7.4.4 Pre-Conditions**

1298 None.

1299 **2.3.7.4.5 Post-Conditions**

1300 None.

1301 **2.4 Log Utility Functional Element**

1302 **2.4.1 Motivation**

1303 In a Web Service-enabled implementation, the Log Utility Functional Element can help to
1304 organise the diagnostic output that may be generated by the implementation. In order to achieve
1305 that, the following capabilities should be provided. They include:

- 1306 • Logging information into different data sources,
- 1307 • Allowing user defined log format to be used,
- 1308 • Capability for storing log information, and
- 1309 • Providing the capability to analyse the information log.

1310

1311 This Functional Element fulfills the following requirements from the Functional Elements
1312 Requirements, Working Draft 01a:

- 1313 • Primary Requirements
 - 1314 • MANAGEMENT-007, [**To be fulfilled in next working draft*]
 - 1315 • MANAGEMENT-110,
 - 1316 • MANAGEMENT-112 to MANAGEMENT-114, and
 - 1317 • PROCESS-009.
- 1318 • Secondary Requirements
 - 1319 • MANAGEMENT-006,
 - 1320 • MANAGEMENT-095,
 - 1321 • MANAGEMENT-111,
 - 1322 • PROCESS-008,
 - 1323 • PROCESS-115, and
 - 1324 • PROCESS-118.

1325

1326 **2.4.2 Terms Used**

Terms	Description
Log Category	A Log Category holds information about a log structure. This information includes the name of the log, the data source the log is to be stored and the format of the log.

1327

1328 **2.4.3 Key Features**

1329 Implementations of the Log Utility Functional Element are expected to provide the following key
1330 features:

- 1331 1. The Functional Element MUST provide the capability to define a Log Category and manage
1332 it. This includes:

- 1333 1.1. The capability to define the format of the log information,
1334 1.2. The capability to choose the data source to logged to, and
1335 1.3. The capability to define the name of the log category.
1336 2. The Functional Element MUST provide the capability to manage logging of events/records.
1337 This includes:
1338 2.1. The capability to insert a new record into the log,
1339 *Examples of a log record could include events, transactions status, usages status or*
1340 *users' activities.*
1341 2.2. The capability to search and view log records, and
1342 2.3. The capability to archive or delete obsolete log records.
1343

1344 In addition, the following key features could be provided to enhance the Functional Element
1345 further:

- 1346 1. The Functional Element MAY also provide the capability to perform conditional search or
1347 viewing of log records.
1348 2. The Functional Element MAY provide the capability to perform basic statistical analysis on
1349 log records. Basic statistical analysis capabilities include:
1350 2.1. Minimum and maximum value calculations on numerical values,
1351 2.2. Mean values calculations on numerical values, and
1352 2.3. Standard deviation calculations on numerical values.

1353 *Note: Report Structure Creation, Generation and Notification are expected to be added in the*
1354 *next Working Draft version under this optional key features.*
1355

1356 **2.4.4 Interdependencies**

1357 None

1358 **2.4.5 Related Technologies and Standards**

1359 None

1360 **2.4.6 Model**

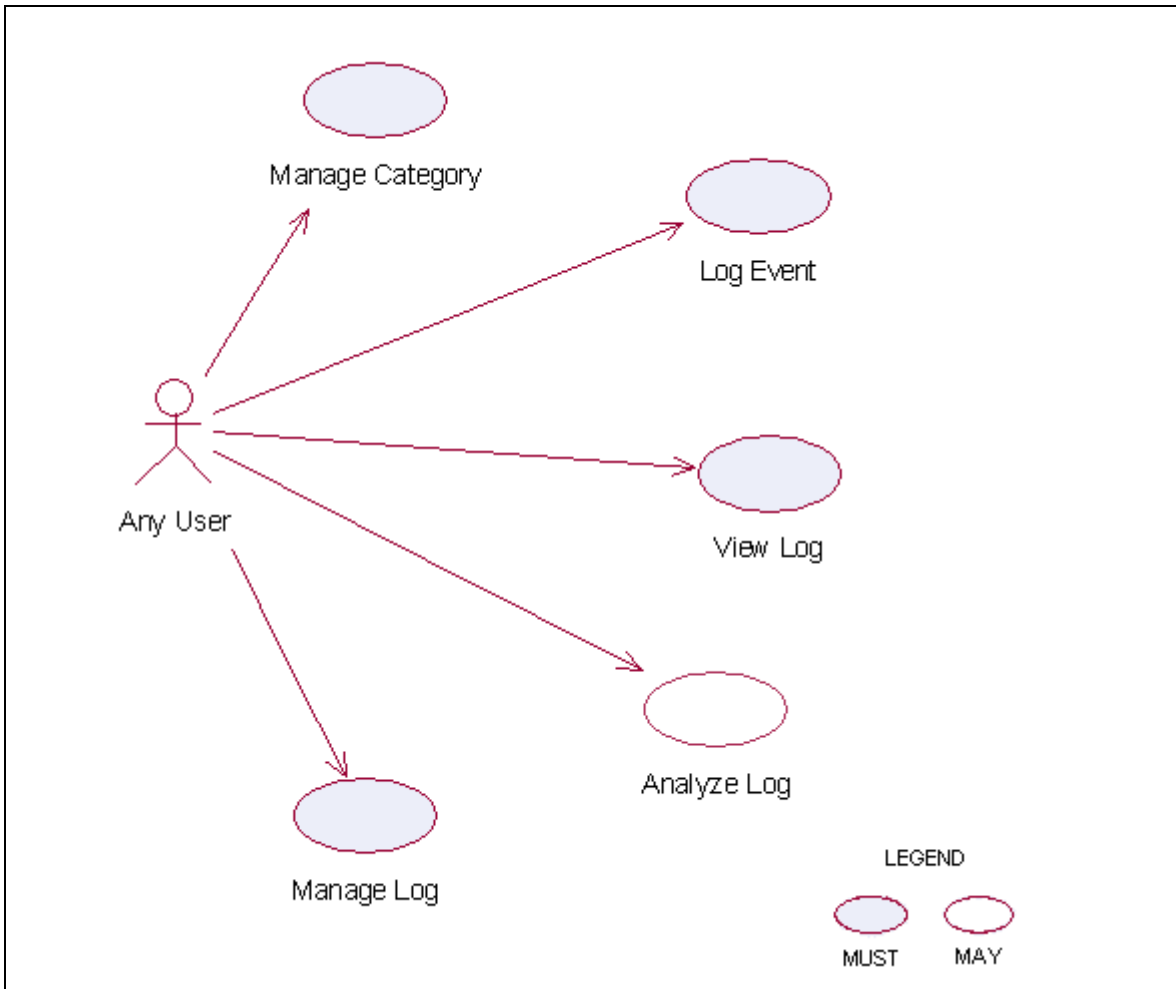


Figure 5: Model Of the Log Utility Functional Element [12]

1361

1362 **2.4.7 Usage Scenarios**

1363 **2.4.7.1 Manage Category**

1364 **2.4.7.1.1 Description**

1365 This use case allows any user to manage log category. Log category defines the data fields that
1366 the user wants to log.

1367 **2.4.7.1.2 Flow of Events**

1368 **2.4.7.1.2.1 Basic Flow**

1369 This use case starts when users wants to manage the log category.

1370 1: The users send the request to the Functional Element. The request contains the operations
1371 the users want to perform.

1372 2: The Functional Element receives the request. Based on the operation specified, one of the
1373 following sub-flows is executed.

- 1374 • If the operation is 'Create Log Category', then sub-flow 2.1 is executed.
- 1375 • If the operation is 'Retrieve Log Category Information', then sub-flow 2.2 is executed.
- 1376 • If the operation is 'Delete Log Category', then sub-flow 2.3 is executed.

1377 2.1: Create Log Category.

1378 2.1.1: The Functional Element gets the following data from the users.

- 1379 • Category name
- 1380 • The definition of category
- 1381 • The data source where the log is located

1382 2.1.2: The Functional Element checks the uniqueness of the category name.

1383 2.1.3: The Functional Element connects to the data source according to the specified
1384 data source.

1385 2.1.4: The Functional Element creates the empty log in the data source.

1386 2.1.5: The Functional Element writes the category name and its definition in its own
1387 category definition record and the use case end.

1388 2.2: Retrieve Log Category Information.

1389 2.2.1: The Functional Element gets the category name.

1390 2.2.2: The Functional Element checks the existence of this category.

1391 2.2.3: The Functional Element retrieves the definition of this category.

1392 2.2.4: The Functional Element returns the definition of this category to the user and the
1393 use case ends.

1394 2.3: Delete Log Category.

1395 2.3.1: The Functional Element gets the category name.

1396 2.3.2: The Functional Element checks the existence of this category.

1397 2.3.3: The Functional Element deletes its own records of category definition and the use
1398 case ends.

1399 **2.4.7.1.2.2 Alternative Flows**

1400 1: Category Already Exists.

1401 1.1: In sub-flow 2.1.2, if the category name is already used by others, the Functional Element
1402 returns an error message and the use case ends.

1403 2: Data Source Not Available.

1404 2.1: In sub-flow 2.1.3, if the data source is not available, the Functional Element returns an
1405 error message and the use case ends.

- 1406 3: Create Log Error.
- 1407 3.1: In sub-flow 2.1.4, if the log cannot be created on the specified data source, the
1408 Functional Element returns an error message and the use case ends.
- 1409 4: Category Does Not Exist.
- 1410 4.1: In sub-flow 2.2.1 and 2.3.1, the category cannot be found in Functional Element category
1411 definition, the Functional Element returns an error message and the use case ends.
- 1412 5: Delete Category Error.
- 1413 5.1: In sub-flow 2.3.3, the log category cannot be deleted, the Functional Element returns an
1414 error message and the use case ends.
- 1415 **2.4.7.1.3 Special Requirements**
- 1416 None
- 1417 **2.4.7.1.4 Pre-Conditions**
- 1418 None.
- 1419 **2.4.7.1.5 Post-Conditions**
- 1420 If the use case was successful, the category definition is saved to the Functional Element and an
1421 empty log is created in the specified data source. Otherwise, the Functional Element's state is
1422 unchanged.
- 1423 **2.4.7.2 Log Event**
- 1424 **2.4.7.2.1 Description**
- 1425 The use case allows any user to log any event.
- 1426 **2.4.7.2.2 Flow of Events**
- 1427 **2.4.7.2.2.1 Basic Flow**
- 1428 This use case starts when users want to write to a log.
- 1429 1: The users provide the event data, category name he/she wants to log to the Functional
1430 Element.
- 1431 2: The Functional Element gets the definition of the category.
- 1432 3: The Functional Element connects the log data source.
- 1433 4: The Functional Element writes the log record into the end of the log file and the use case ends.
- 1434 **2.4.7.2.2.2 Alternative Flows**
- 1435 1: Category Does Not Exist.
- 1436 1.1: If in basic flow 2, the category that the users want to write does not exist, the Functional
1437 Element returns an error message and the use case ends.
- 1438 2: Data Source Not Available.

1439 2.1: If in basic flow 3, the data source is not available, the Functional Element returns an error
1440 message and the use case ends.

1441 3: Data Not Match.

1442 3.1: If in basic flow 4, the data provided by the users for logging does not match with the
1443 category definition in the Functional Element, the Functional Element returns an error
1444 message and the use case ends.

1445 **2.4.7.2.3 Special Requirements**

1446 None.

1447 **2.4.7.2.4 Pre-Conditions**

1448 None.

1449 **2.4.7.2.5 Post-Conditions**

1450 If the use case was successful, the log record is saved to the Functional Element. Otherwise, the
1451 Functional Element's state is unchanged.

1452 **2.4.7.3 View Log**

1453 **2.4.7.3.1 Description**

1454 The use case allows users to retrieve the log content.

1455 **2.4.7.3.2 Flow of Events**

1456 **2.4.7.3.2.1 Basic Flow**

1457 This use case starts when users want to view a log.

1458 1: The users specify the category name and the search criteria, such as searching by event type
1459 or searching by time period (starting time and end time).

1460 2: The Functional Element connects to the data storage where the log records are stored.

1461 3: The Functional Element retrieves the log content and returns to the service users and the use
1462 case ends.

1463 **2.4.7.3.2.2 Alternative Flows**

1464 1: Search Criteria Not Valid.

1465 1.1: If in basic flow 1 and 3, the search criteria specified by the users is invalid for Search
1466 Service, the Functional Element returns an error message and the use case ends.

1467 **2.4.7.3.3 Special Requirements**

1468 None.

1469 **2.4.7.3.4 Pre-Conditions**

1470 None.

1471 **2.4.7.3.5 Post-Conditions**

1472 None.

1473 **2.4.7.4 Analyze Log Data**

1474 **2.4.7.4.1 Description**

1475 The use case allows users to analyze the log data, i.e., to get statistics of certain event. The
1476 service users may get statistical results on the log data, such as the cumulative events and mean
1477 of two numerical values.

1478 **2.4.7.4.2 Flow of Events**

1479 **2.4.7.4.2.1 Basic Flow**

1480 This use case starts when users want to analyze the log data.

1481 1: The users specify the items to analyze, i.e. field name and category name.

1482 2: The users specify the analysis method, option among max, min and mean.

1483 3: The Functional Element retrieves the definition of the category and validates the parameters
1484 provided by the users.

1485 4: The Functional Element connects to the data source and retrieves the log data.

1486 5: The Functional Element analyses the log data and does statistics on the data with respect to
1487 what is specified in Step 1 and 2.

1488 6: The Functional Element returns the analyzed result and the use case ends.

1489 **2.4.7.4.2.2 Alternative Flows**

1490 1: Invalid Item Specified.

1491 1.1: If in basic flow 1, the analyze items specified by the users are invalid, i.e. invalid field and
1492 invalid data source, the Functional Element returns an error message and the use case ends.

1493 2: Category Does Not Exist.

1494 2.1: If in basic flow 3, the category that the users want to write to does not exist, the
1495 Functional Element returns an error message and the use case ends.

1496 3: Data Source Not Available.

1497 3.1: If in basic flow 4, the data source is not available, the Functional Element returns an error
1498 message and the use case ends.

1499 **2.4.7.4.3 Special Requirements**

1500 **2.4.7.4.3.1 Supportability**

1501 Only basic statistic methods of numerical value are supported.

1502 **2.4.7.4.4 Pre-Conditions**

1503 None.

1504 **2.4.7.4.5 Post-Conditions**

1505 None.

1506 **2.4.7.5 Manage Log**

1507 **2.4.7.5.1 Description**

1508 The use case allows users to drop log and backup log.

1509 **2.4.7.5.2 Flow of Events**

1510 **2.4.7.5.2.1 Basic Flow**

1511 The use case starts when the users want to drop and backup a log of a specific data source.

1512 1: The users specify the function name to the Functional Element.

1513 2: Based on the operation specified, one of the following sub-flows is executed.

1514 • If the operation is '**Delete Log**', then sub-flow 2.1 is executed.

1515 • If the operation is '**Backup Log**', then sub-flow 2.2 is executed.

1516 2.1: Delete Log

1517 2.1.1: The Functional Element gets category name from the users.

1518 2.1.2: The Functional Element retrieves the definition of the category.

1519 2.1.3: The Functional Element connects to the corresponding data source.

1520 2.1.4: The Functional Element deletes the log from the data source.

1521 2.2: Backup Log

1522 2.2.1: The Functional Element gets the category name and the destination file name from
1523 the users.

1524 2.2.2: The Functional Element retrieves the definition of the category.

1525 2.2.3: The Functional Element connects to the corresponding data source.

1526 2.2.4: The Functional Element read the original log and writes it to the destination file.

1527 **2.4.7.5.2.2 Alternative Flows**

1528 1: Category Does Not Exist.

1529 1.1: If in basic flow 2.1.2 and 2.2.2 the category that the users want to write does not exist,
1530 the Functional Element returns an error message and the use case ends.

1531 2: Data Source Not Available.

1532 2.1: If in basic flow 2.1.4 and 2.2.4, the data source is not available, the Functional Element
1533 returns an error message and the use case ends.

1534 **2.4.7.5.3 Special Requirements**

1535 None.

1536 **2.4.7.5.4 Pre-Conditions**

1537 None.

1538 **2.4.7.5.5 Post-Conditions**

1539 None.

1540

1541 2.5 Notification Functional Element

1542 2.5.1 Motivation

1543 In a Web Service-enabled implementation, timely information is crucial for the management of
1544 resources that it encompasses. Other uses of this Functional Element include broadcasting of
1545 information to other services and this could span across both the wired and wireless medium. In
1546 order to fulfill these needs, this Functional Element will cover the following aspects which include:

- 1547 • Providing the capability to configure and link with the various gateways so as to enable
1548 messages dissemination, and
- 1549 • Providing the capability to send instantaneous or scheduled messages to the intended
1550 audiences.

1551

1552 This Functional Element fulfills the following requirements from the Functional Elements
1553 Requirements, Working Draft 01a:

- 1554 • Primary Requirements
 - 1555 • DELIVERY-003, and
 - 1556 • PROCESS-118.
- 1557 • Secondary Requirements
 - 1558 • MANAGEMENT-205,
 - 1559 • PROCESS-005,
 - 1560 • PROCESS-102,
 - 1561 • PROCESS-107, and
 - 1562 • PROCESS-110.

1563

1564 2.5.2 Terms Used

Terms	Description
Default Notification Channel	Default Notification Channel refers to the particular channel setting or value that is assigned automatically by the Functional Element and remains in effect unless canceled or overridden.
Device Type	Device Type refers to devices such as Mobile Phone, Numeric Pager, Alphanumeric Numeric Pager and Desktop etc.
Notification Channel	Notification Channel refers to the various messaging channels such as SMS (Short Message Service), Numeric Message, Alpha-numeric Message and E-mail Message etc.
Schedule Type	Schedule Type refers to the various types of Scheduling format such as ONCE, DAILY, WEEKLY, and MONTHLY.
SMS	Short Message Service
SMS Gateway	A device that enable sending of numeric, alpha-numeric and SMS messages.

SMTP	Simple Mail Transfer Protocol
SMTP Server	SMTP server supports email notifications.

1565

1566 2.5.3 Key Features

1567 Implementations of the Notification Functional Element are expected to provide the following key
1568 features:

- 1569 1. The Functional Element MUST support notifications using both the SMS and SMTP
1570 protocols.
- 1571 2. The Functional Element MUST provide the capability to configure supported SMS
1572 gateway(s) and the SMTP servers where applicable.
1573 *Example: The capability to configure the username and password for SMTP server's*
1574 *authentication.*
- 1575 3. The Functional Element MUST provide the capability to send notifications to single and
1576 multiple recipients.
- 1577 4. The Functional Element MUST provide the capability to structure a notification based on the
1578 selected protocol(s).

1579

1580 In addition, the following key features could be provided to enhance the Functional Element
1581 further:

- 1582 1. The Functional Element MAY provide the capability to send notifications either instantly or
1583 based on a pre-defined schedule.
- 1584 2. If Key Feature (1) is provided, the Functional Element MAY also provide the capability to
1585 send scheduled messages in the following manner:
 - 1586 2.1. Hourly,
 - 1587 2.2. Daily,
 - 1588 2.3. Weekly, and
 - 1589 2.4. Monthly (based on a particular date or particular day of the week).

1590 *Note: The next working draft version will attempt to look at other available protocols.*

1591

1592 2.5.4 Interdependencies

1593 None

1594 2.5.5 Related Technologies and Standards

Technologies	Description
Short Message Service (SMS)	Short Message Service is a feature available with some wireless phones that allow users to send and/or receive short alphanumeric messages. This Functional Element is heavily reliance on this for transmission of messages to a pager and hand phone.

Simple Mail Transfer Protocol (SMTP)	A protocol used to send e-mail on the Internet. SMTP is a set of rules regarding the interaction between a program sending e-mail and a program receiving e-mail. This Functional Element is heavily reliance on this for transmission of messages to the designated email account.
--------------------------------------	---

1595

1596 **2.5.6 Model**

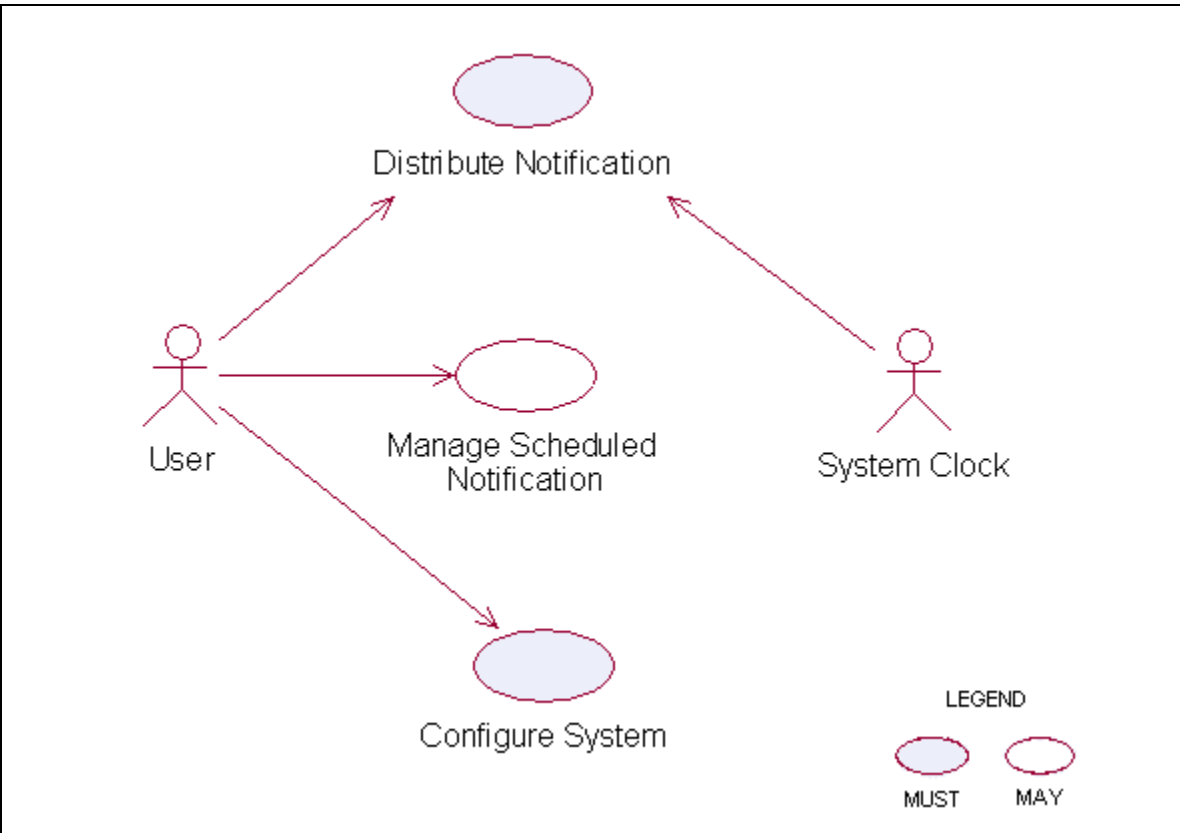


Figure 6: Model Of the Notification Functional Element [13]

1597 **2.5.7 Usage Scenarios**

1598 **2.5.7.1 Distribute Notification**

1599 **2.5.7.1.1 Description**

1600 This use case allows the Functional Element to distribute messages to intended recipients.

1601 **2.5.7.1.2 Flow of Events**

1602 **2.5.7.1.2.1 Basic Flow**

1603 This use case starts when the service user or system clock wishes to send message to recipient.

1604 1: The Functional Element decides to send messages to recipients. Based on the operation
1605 specified, one of the following sub-flows is executed.

- 1606 • If the request is '**Initiated By The User**', then sub-flow 1.1 is executed.
- 1607 • If the request is '**Initiated By The System Clock**' then sub-flow 1.2 is executed.

1608 1.1: Initiated By The User

1609 1.1.1: The Functional Element receives the request from the service user.

1610 1.1.2: The Functional Element validates passed parameters such as message type,
1611 recipient address, and message key and message length.

1612 1.1.3: The Functional Element checks the availability of the connection.

1613 1.1.4: The Functional Element sends message to recipient(s) and the use case end

1614 1.2 : Initiated By The System Clock

1615 1.2.1: The Functional Element checks scheduled message(s) and end date for scheduled
1616 message.

1617 1.2.2: Once the Functional Element detects scheduled messages, one of the sub-flows is
1618 executed.

- 1619 • If the Functional Element detects the scheduled notification is once, the '**Activate**
1620 **Once Notification**' sub-flow 1.2.2.1 is executed.
- 1621 • If the Functional Element detects the scheduled notification is daily, the '**Activate**
1622 **Daily Notification**' sub-flow 1.2.2.2 is executed.
- 1623 • If the Functional Element detects the scheduled notification is weekly, the
1624 '**Activate Weekly Notification**' sub-flow 1.2.2.3 is executed.
- 1625 • If the Functional Element detects the scheduled notification is Monthly, the
1626 '**Activate Monthly Notification**' sub-flow 1.2.2.4 is executed.

1627 1.2.2.1: Activate Once Notification.

1628 1.2.2.1.1: The Functional Element compares the system time with the scheduled
1629 message's time and gets notification details if both times are match.

1630 1.2.2.2: Activate Daily Notification.

1631 1.2.2.2.1: The Functional Element compares the system time with the scheduled
1632 message's time and gets notification details if both times are match.

1633 1.2.2.3: Activate Weekly Notification.

1634 1.2.2.3.1: The Functional Element compares the system date and time with the
1635 scheduled message's date and time and gets notification details if both date &
1636 time are match.

1637 1.2.2.4: Activate Monthly Notification.

1638 1.2.2.4.1: The Functional Element compares the system date and time with the
1639 scheduled message's date and time and gets notification ID if both date & time
1640 are match.

1641 1.2.3: The Functional Element extracts the list of recipient(s) and message(s).

1642 1.2.4: The Functional Element checks the availability of connection.

1643 1.2.5: The Functional Element sends message to recipient(s) and the use case ends.

1644 **2.5.7.1.2.2 Alternative Flows**

1645 1: Unsupported Message Type/Recipient Address/Message.

1646 1.1: If in basic flow 1.1.2, Functional Element detects unsupported message type, recipient
1647 address or message, the Functional Element returns an error message and the use case
1648 ends.

1649 2: Connection Fail.

1650 2.1: If in basic flow 1.1.3 and 1.2.4, the Functional Element is unable to detect connection
1651 type, the Functional Element returns an error message and the use case ends

1652 3: Delete Scheduled Message.

1653 3.1: If in basic flow 1.2.1, if the Functional Element detects that the scheduled message has
1654 expired, the Functional Element will proceed to delete those messages.

1655 **2.5.7.1.3 Special Requirements**

1656 **2.5.7.1.3.1 Supportability**

1657 None

1658 **2.5.7.1.4 Pre-Conditions**

1659 None.

1660 **2.5.7.1.5 Post-Conditions**

1661 None.

1662 **2.5.7.2 Manage Scheduled Notification**

1663 **2.5.7.2.1 Description**

1664 This use case allows the service user to maintain the notification information. This includes
1665 adding, changing and deleting notification information from the Functional Element.

1666 **2.5.7.2.2 Flow of Events**

1667 **2.5.7.2.2.1 Basic Flow**

1668 This use case starts when the service user wishes to schedule notification message(s).

1669 1: The Functional Element requests the service user to specify the function he/she would like to
1670 perform (such as create, update and delete notification message).

1671 2: Once the Functional Element user provides the requested information, one of the sub-flows is
1672 executed.

1673

- If the service user provides '**Create Notification**', then sub-flow 2.1 is executed.

1674

- If the service user provides '**Delete Notification**', then sub-flow 2.2 is executed.

- 1675 2.1 Create Notification
- 1676 2.1.1: The Functional Element receives the request from the service user.
- 1677 2.1.2: The Functional Element validates passed parameters such as schedule type,
1678 message type, recipient address, message key and the message length.
- 1679 2.1.3: The Functional Element generates and assigns a unique Notification ID and adds
1680 the notification information to the Functional Element and ends use case.
- 1681 2.2: Delete Notification
- 1682 2.2.1: The Functional Element requests the service user to provide the Notification
1683 information.
- 1684 2.2.2: The Functional Element retrieves the existing Notification information.
- 1685 2.2.3: The Functional Element deletes the Notification record and use case ends.
- 1686 **2.5.7.2.2.2 Alternative Flows**
- 1687 1: Invalid Parameters.
- 1688 1.1: If in basic flow 2.1.2, if the Functional Element detects invalid parameters such as
1689 schedule type, date & time, recipient address, message key and message, the Functional
1690 Element returns an error message and the use case ends.
- 1691 **2.5.7.2.3 Special Requirements**
- 1692 None.
- 1693 **2.5.7.2.4 Pre-Conditions**
- 1694 None.
- 1695 **2.5.7.2.5 Post-Conditions**
- 1696 If the use case was successful, the schedule message information is added to Functional
1697 Element. Otherwise, the Functional Element's state is unchanged.
- 1698 **2.5.7.3 Configure System**
- 1699 **2.5.7.3.1 Description**
- 1700 This use case allows the service user to maintain the notification Functional Element behaviors.
1701 This includes configuration of supported Notification Channel, Default Notification Channel,
1702 Schedule Types, and SMS and SMTP Gateway.
- 1703 **2.5.7.3.2 Flow of Events**
- 1704 **2.5.7.3.2.1 Basic Flow**
- 1705 1: The Functional Element requests the service user to specify or configure the function he/she
1706 would like to perform (such as create, update and delete configuration parameters).
- 1707 2: Once the Functional Element user provides the requested information, one of the sub-flows is
1708 executed.

- 1709 • If user wishes to configure '**Notification Channel**', then sub-flow 2.1 is executed.
- 1710 • If user wishes to configure '**Default Notification Channel**', then sub-flow 2.2 is executed.
- 1711 • If user wishes to configure '**Schedule Types**', then sub-flow 2.3 is executed.
- 1712 • If user wishes to configure '**SMTP server and SMS Gateway**', then sub-flow 2.4 is
- 1713 executed.
- 1714 2.1 Notification Channel.
- 1715 2.1.1: The Functional Element receives the request from the service user.
- 1716 2.1.2: The Functional Element validates passed parameters such as Notification Channel
- 1717 information.
- 1718 2.1.3: The Functional Element generates and assigns a unique Notification Channel ID
- 1719 and adds the notification information to the Functional Element and the use case ends.
- 1720 2.2: Default Notification Channel.
- 1721 2.2.1: The Functional Element requests the service user to provide the Default
- 1722 Notification information.
- 1723 2.2.2: The Functional Element validates passed parameters such as Default Notification
- 1724 Channel information.
- 1725 2.2.3: The Functional Element updates existing Default Notification or create new Default
- 1726 Notification information and the use case ends.
- 1727 2.3 Schedule Types.
- 1728 2.3.1: The Functional Element receives the request from the service user.
- 1729 2.3.2: The Functional Element validates passed parameters such as Schedule Type.
- 1730 2.3.3: The Functional Element generates and assigns a unique Schedule Type ID and
- 1731 adds the Schedule Type information to the Functional Element and the use case ends.
- 1732 2.4: SMTP server and SMS Gateway.
- 1733 2.4.1: The Functional Element requests the service user to provide the SMTP server and
- 1734 SMS Gateway information.
- 1735 2.4.2: The Functional Element validates passed parameters such as SMTP server and
- 1736 SMS Gateway information.
- 1737 2.4.3: The Functional Element updates existing SMTP server and SMS Gateway or
- 1738 create new SMTP server and SMS Gateway information and the use case ends.

1739 **2.5.7.3.2.2 Alternative Flows**

- 1740 1: Invalid Parameters.
- 1741 1.1: If in sub-flow 2.1.2, 2.2.2, 2.3.2 and 2.4.2, if the Functional Element detects invalid
- 1742 parameters such as Notification Channel, Default Notification Channel, and SMTP server,
- 1743 Schedule Types and SMS Gateway information, the Functional Element returns an error
- 1744 message and the use case ends

1745 **2.5.7.3.3 Special Requirements**

1746 None.

1747 **2.5.7.3.4 Pre-Conditions**

1748 None.

1749 **2.5.7.3.5 Post-Conditions**

1750 None

1751 2.6 Phase and Lifecycle Management Functional Element

1752 2.6.1 Motivation

1753 The Phase and Lifecycle Management Functional Element is expected to be an integral part of
1754 the User Access Management (UAM) functionalities that is expected to be needed by a Web
1755 Service-enabled implementation. This FE is expected to fulfill the needs arising out of managing
1756 the dynamic status of user information across the whole lifecycle. As such it will cover aspects
1757 that include:

- 1758 • Basic lifecycle management facilities,
- 1759 • Basic phase management facilities, and
- 1760 • Management of user information in phases across the whole lifecycle.

1761

1762 This Functional Element fulfills the following requirements from the Functional Elements
1763 Requirements, Working Draft 01a:

- 1764 • Primary Requirements
 - 1765 • MANAGEMENT-070 to MANAGEMENT-078
- 1766 • Secondary Requirements
- 1767 • None

1768

1769 2.6.2 Terms Used

Terms	Description
Group	A Group is a collection of individual users, and are typically grouped together as they have certain commonalities
Namespace	Namespace is use to segregate the instantiation of the application across different application domains. If a company has two separate standalone application, for example, an email application and an equipment booking application, then these two are considered as separate application domains
Phase/lifecycle	Phase/lifecycle refers to the phases a project goes through between when it is conceived and when it is completed. As an example, the software lifecycle. It typically includes the following phases. <ul style="list-style-type: none">• Requirements Analysis• Design, Construction• Testing (Validation)• Installation• Operation• Maintenance• Retirement.

User	A user is loosely defined to include both human and virtual users. Virtual users could include service users and application (or machine) users that are utilising other services in a SOA environment.
User Access Management / Uam	<p>User Access Management or UAM refer to the concept of managing users in a holistic manner, considering all aspect which includes:</p> <ul style="list-style-type: none"> • Defining a set of basic user information that should be stored in any enterprise application. • Providing a means to extend this basic set of user information when needed.. • Simplifying management by grouping related users together through certain criteria. • Having the flexibility of adopting both coarse/fine grain access control.

1770

1771 **2.6.3 Key Features**

1772 Implementations of the Phase and Lifecycle Management Functional Element are expected to
 1773 provide the following key features:

- 1774 1. The Functional Element MUST provide basic structures based on a set of pre-defined
 1775 attributes for Lifecycle and Phase.
- 1776 2. The Functional Element MUST provide the capability to manage the creation and deletion of
 1777 instances of Lifecycle and Phase based on the pre-defined structures.
- 1778 3. The Functional Element MUST provide a means to manage the lifecycles and phases
 1779 contained within. This includes:
- 1780 3.1. The capability to retrieve and update a lifecycle or phase
- 1781 3.2. The capability to add/remove phases from a lifecycle
- 1782 4. The Functional Element MUST provide a mechanism to manage the collection of users in a
 1783 Phase. This includes:
- 1784 4.1. The capability to assign and un-assign users belonging to a Phase.
- 1785 4.2. The users could be individual Users or Groups.
- 1786 5. The Functional Element MUST provide a mechanism for managing Groups across different
 1787 application domains.

1788 *Example: Namespace control mechanism*

1789

1790 **2.6.4 Interdependencies**

Direct Dependency	
Group Management Functional Element	The Group Management Functional Element is used to achieve effective and efficient management of user's information in each of the different phases..

1791

1792 **2.6.5 Related Technologies and Standards**

1793 None.

1794 **2.6.6 Model**

1795

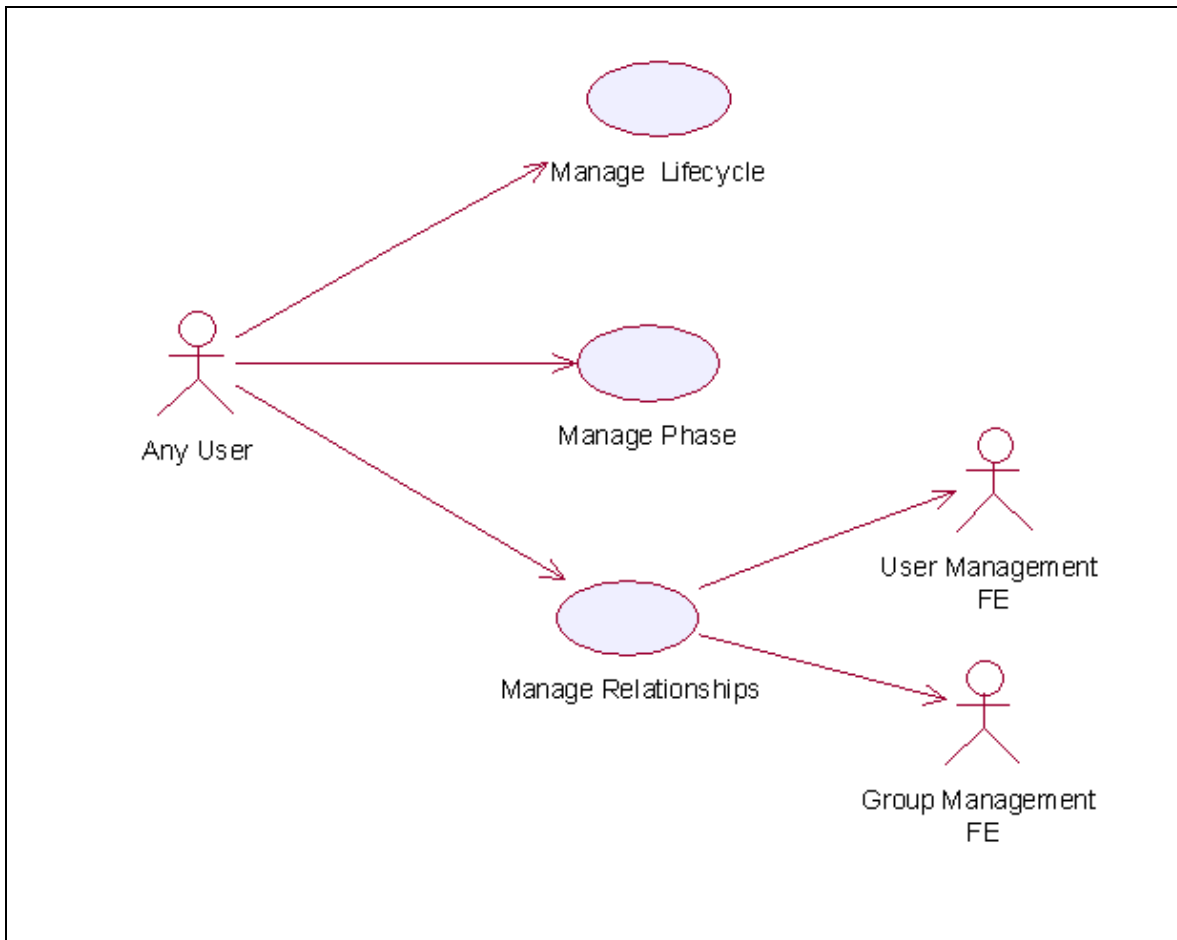


Figure 7: Model Of the Phase and Lifecycle Functional Element [14]

1796 **2.6.7 Usage Scenarios**

1797 **2.6.7.1 Manage Lifecycle**

1798 **2.6.7.1.1 Description**

1799 This use case is used to create, update, retrieve and delete the lifecycle.

1800 **2.6.7.1.2 Flow of Events**

1801 **2.6.7.1.2.1 Basic Flow**

1802 This use case starts when the user wants to manage phase in lifecycle.

- 1803 • If user wants to '**Create Lifecycle**', then basic flow 1 is executed.
- 1804 • If user wants to '**Retrieve Lifecycle**', then basic flow 2 is executed.
- 1805 • If user wants to '**Update Lifecycle**', then basic flow 3 is executed.

- 1806 • If user wants to '**Delete Lifecycle**', then basic flow 4 is executed.
- 1807 1: Create Lifecycle.
- 1808 1.1: User provides information to create lifecycle.
- 1809 1.2: Functional Element creates lifecycle and the use case ends.
- 1810 2: Retrieve Lifecycle
- 1811 2.1: User provides the lifecycle name, lifecycle namespace.
- 1812 2.2: Functional Element returns the lifecycle information and the use case ends.
- 1813 3: Update Lifecycle.
- 1814 3.1: User provides the lifecycle information.
- 1815 3.2: Functional Element updates the lifecycle-phase and the use case ends.
- 1816 4: Delete Lifecycle.
- 1817 4.1: User provides lifecycle name and lifecycle namespace.
- 1818 4.2: Functional Element deletes the lifecycle and the use case ends.
- 1819 **2.6.7.1.2.2 Alternative Flows**
- 1820 1: Lifecycle Does Not Exist.
- 1821 1.1: In basic flow 2.1, 3.1 and 4.1, if lifecycle can not be found based on lifecycle name and
1822 lifecycle namespace provided by user, Functional Element returns an error message and the
1823 use case ends.
- 1824 2: Creation Of Lifecycle Fails.
- 1825 2.1: In basic flow 1.2, if lifecycle cannot be created, the Functional Element returns an error
1826 message and the use case ends
- 1827 **2.6.7.1.3 Special Requirements**
- 1828 None.
- 1829 **2.6.7.1.4 Pre-Conditions**
- 1830 None.
- 1831 **2.6.7.1.5 Post-Conditions**
- 1832 None.
- 1833 **2.6.7.2 Manage Phase**
- 1834 **2.6.7.2.1 Description**
- 1835 This use case describes the management of different phases in a project.

1836 **2.6.7.2.2 Flow of Events**

1837 **2.6.7.2.2.1 Basic Flow**

1838 This use case starts when the user wants to manage phase.

- 1839 • If user wants to 'Create Phase', then basic flow 1 is executed.
- 1840 • If user wants to 'Retrieve Phase', then basic flow 2 is executed.
- 1841 • If user wants to 'Update Phase', then basic flow 3 is executed.
- 1842 • If user wants to 'Delete Phase', then basic flow 4 is executed.

1843 1: Create Phase.

1844 1.1: User provides information to create phase.

1845 1.2: Functional Element creates phase and the use case ends.

1846 2: Retrieve Phase.

1847 2.1: User provides phase name, lifecycle name and lifecycle namespace.

1848 2.2: Functional Element returns the phase information and the use case ends.

1849 3: Update Phase.

1850 3.1: User provides the phase information.

1851 3.2: Functional Element updates the phase and the use case ends.

1852 4: Delete Phase.

1853 4.1: User provides phase name, lifecycle name and lifecycle namespace

1854 4.2: Functional Element deletes phase and the use case ends.

1855 **2.6.7.2.2.2 Alternative Flows**

1856 1: Phase Does Not Exist.

1857 1.1: In basic flow 2.1, 3.1 and 4.1 if phase cannot be found based on phase name, lifecycle
1858 name and lifecycle namespace provided by user, Functional Element returns an error
1859 message and the use case ends.

1860 2: Creation of phase fails.

1861 2.1: In basic flow 1.2, if phase cannot be created, the Functional Element returns an error
1862 message and the use case ends

1863 **2.6.7.2.3 Special Requirements**

1864 None.

1865 **2.6.7.2.4 Pre-Conditions**

1866 None.

1867 **2.6.7.2.5 Post-Conditions**

1868 None.

1869 **2.6.7.3 Manage Relationship**

1870 **2.6.7.3.1 Description**

1871 This use case describes the management of the relationship between user/group and phase in a
1872 lifecycle.

1873 **2.6.7.3.2 Flow of Events**

1874 **2.6.7.3.2.1 Basic Flow**

1875 This use case starts when the user wants to manage the relationship between the user/group and
1876 phase.

- 1877 • If user refers to '**Create Relationship**', basic flow 1 is executed.
- 1878 • If user refers to '**Update Relationship**', basic flow 2 is executed.
- 1879 • If user wants to '**Retrieve Relationship**', basic flow 3 is executed.
- 1880 • If user refers to '**Delete Relationship**', basic flow 4 is executed.

1881 1: Create Relationship.

1882 1.1: User provides user/group, phase and phase information.

1883 1.2: Functional Element creates relationship and the use case ends.

1884 2: Update Relationship.

1885 2.1: User provides user/group name and user/group namespace.

1886 2.2: Functional Element updates the relationship and the use case ends.

1887 3: Retrieve Relationship.

1888 3.1: User provides user/group name and user/group namespace.

1889 3.2: Functional Element returns the relationship and the use case ends.

1890 4: Delete Relationship.

1891 4.1: User provides user/group name and user/group namespace.

1892 4.2: Functional Element deletes relationship between phases and users/groups and the use
1893 case ends.

1894 **2.6.7.3.2.2 Alternative Flows**

1895 1: Phase Does Not Exist.

1896 1.1: In basic flow 1,2, 2.2, 3.2 and 4.2, if the phase does not exist, the Functional Element
1897 returns an error message and the use case ends.

1898 2: User/Group Does Not Exist.

1899 1.1: In basic flow 1,2, 2.2, 3.2 and 4.2, if the user/group does not exist, the Functional
1900 Element returns an error message and the use case ends.

1901 **2.6.7.3.3 Special Requirements**

1902 None.

1903 **2.6.7.3.4 Pre-Conditions**

1904 None.

1905 **2.6.7.3.5 Post-Conditions**

1906 None.

1907

1908 **2.7 Presentation Transformer Functional Element**

1909 **2.7.1 Motivation**

1910 In a Web Service implementation, there exists the need to render the eventual presentation
1911 layout to different consumers depending on their receiving capabilities. As such, there is a need
1912 to dynamically generate the appropriate output at runtime.

1913

1914 This Functional Element fulfills the following requirements from the Functional Elements
1915 Requirements, Working Draft 01a:

- 1916
- Primary Requirements
 - 1917 • DELIVERY-001, and
 - 1918 • DELIVERY-005 to DELIVERY-007.
 - Secondary Requirements
 - 1919 • None
 - 1920

1921 **2.7.2 Terms Used**

Terms	Description
XSL	Extensible Stylesheet Language

1922 **2.7.3 Key Features**

1923 Implementations of the Presentation Transformer Functional Element are expected to provide the
1924 following key features:

- 1925 6. The Functional Element MUST be able to transform an XML document into a required
1926 presentation format (output).
- 1927 7. The Functional Element MUST be able to understand a XSL document for the specifications
1928 of the required presentation format.
- 1929

1930 **2.7.4 Interdependencies**

1931 None

1932 **2.7.5 Related Technologies and Standards**

1933 None

1934

1935 **2.7.6 Model**

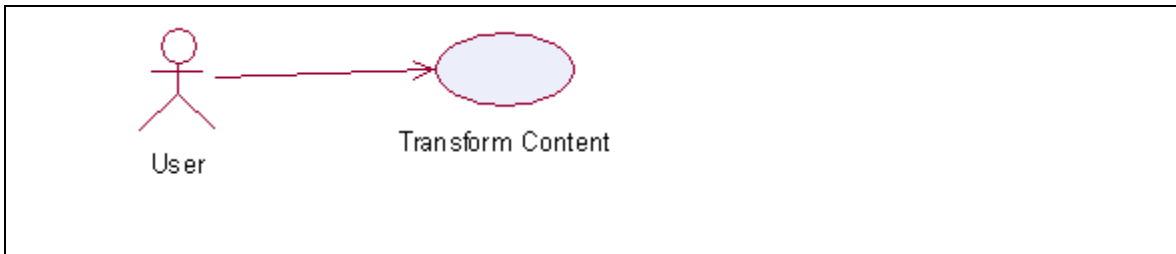


Figure 8: Model Of the Presentation Functional Element [15]

1936

1937 **2.7.7 Usage Scenario**

1938 **2.7.7.1 Transform Content**

1939 **2.7.7.1.1 Description**

1940 This use case allows the service user to transform the content to the appropriate mark-up
1941 language.

1942 **2.7.7.1.2 Flow of Events**

1943 **2.7.7.1.2.1 Basic Flow**

1944 This use case starts when the service user wishes to transform the content to the appropriate
1945 mark-up language.

1946 1: The Functional Element receives the request from the service user to transform the content to
1947 the appropriate mark-up language.

1948 2: The Functional Element detects the type of mark-up language to be transformed.

1949 3: The Functional Element extracts mark-up language type from either the XML document or
1950 parameters.

1951 4: The Functional Element retrieves appropriate mark-up language style sheet and transform it
1952 into appropriate mark-up language.

1953 5: The Functional Element returns transformed appropriate result and the use case ends.

1954 **2.7.7.1.2.2 Alternative Flows**

1955 1: Unsupported Content.

1956 1.1: If in basic flow 2, the Functional Element is unable to detect the content type, the
1957 Functional Element returns an error message and the use case ends.

1958 2: Unsupported Mark-up Language.

1959 2.1: If in basic flow 2, the Functional Element is unable to detect the supported mark-up
1960 language type, the Functional Element returns an error message and the use case ends.

1961 **2.7.7.1.3 Pre-Conditions**

1962 None.

1963 **2.7.7.1.4 Post-Conditions**

1964 None.

1965

1966 **2.8 Role and Access Management Functional Element**

1967 **2.8.1 Motivation**

1968 The Role and Access Management Functional Element is expected to be an integral part of the
1969 User Access Management (UAM) functionalities that is expected to be needed by a Web Service-
1970 enabled implementation. This Functional Element is expected to fulfill the needs arising out of
1971 managing access to resources within an application, based on role-based access control
1972 mechanism. As such it will cover aspects that include:

- 1973 • Management of roles and access privileges, and
1974 • Assignment of roles to entities that will be accessing the resources that is being
1975 managed.

1976

1977 This Functional Element fulfills the following requirements from the Functional Elements
1978 Requirements, Working Draft 01a:

- 1979 • Primary Requirements
1980 • MANAGEMENT-030 to MANAGEMENT-034, and
1981 • MANAGEMENT-200 to MANAGEMENT-205.
1982 • Secondary Requirements
1983 • SECURITY-040 to SECURITY-041.

1984

1985 **2.8.2 Terms Used**

Terms	Description
Access Control	Access Control refers to the process of ensuring that only an authorized user can access the resources within a computer system.
Lifecycle	A lifecycle refers to the sequence of phases in the lifetime of a resource.
Phase	A phase refers to the different stages that a resource may be in when viewed from a lifecycle perspective
Resource	A resource in an application is defined to encompass data/information in a system. Examples of this information include users information, transaction information and security information.
Role	A role is typically assigned to a user to define or indicate the job or responsibility of the said user in a particular context.

Role Based Access Control	<p>Role Based Access Control is a model of access management mechanism. In this model, the access control is enabled in the following manner:</p> <ul style="list-style-type: none"> • Determine who (user) is requesting access. • Determine the role(s) of the user • Determine the type of access that is allowed based on the role(s) of the user <p>It is the task of the access control mechanism to ensure that only processes, which are explicitly authorized, perform the operation by these objects.</p>
User	<p>A user is loosely defined to include both human and virtual users. Virtual users could include service users and application (or machine) users that are utilising other services in a SOA environment.</p>
User Access Management / UAM	<p>User Access Management or UAM refer to the concept of managing users in a holistic manner, considering all aspect which includes:</p> <ul style="list-style-type: none"> • Defining a set of basic user information that should be stored in any enterprise application. • Providing a means to extend this basic set of user information when needed.. • Simplifying management by grouping related users together through certain criteria. • Having the flexibility of adopting both coarse/fine grain access control.

1986

1987 **2.8.3 Key Features**

1988 Implementations of the Group Management Functional Element are expected to provide the
1989 following key features:

- 1990 1. The Functional Element MUST provide the capability to manage the creation and deletion of
1991 instances of the following concepts based on a pre-defined structure:
- 1992 1.1. Role,
 - 1993 1.2. Access, and
 - 1994 1.3. Resource
- 1995 2. The Functional Element MUST provide the capability to manage all the information (attribute
1996 values) stored in such concepts. This includes the capability to retrieve and update
1997 attribute's values belonging to a concept like Role, Access or Resource.
- 1998 3. The Functional Element MUST provide the capability to associate a Role to its access
1999 privileges through the Access structure.
- 2000 4. The Functional Element MUST provide the capability to determine a Role's accessibility to
2001 Resources based on the access privileges that have been assigned.
- 2002 5. The Functional Element MUST provide the ability to manage the association of users to
2003 Roles via assignments of Roles to users. This will include:
- 2004 5.1. Assignment/Un-assignment of Roles to individual Users, and
 - 2005 5.2. Assignment/Un-assignment of Roles to Groups.

2006 This will provide an indirect linkage between the accessibility of specific Users to Resources
2007 through the concept of Role and Access.

2008 6. The Functional Element MUST provide a mechanism for managing the concepts of Role,
2009 Access and Resource across different application domains.

2010 *Example: Namespace control mechanism*

2011

2012 In addition, the following key features could be provided to enhance the Functional Element
2013 further:

2014 1. The Functional Element MAY provide a mechanism to enable different Access instances to
2015 be related to one another.

2016 2. The Functional Element MAY also provide a mechanism to enable hierarchical
2017 relationships between Access instances.

2018 *Example: Parent and Child Relationship*

2019 3. The Functional Element MAY provide the ability for Roles to be temporal sensitive.

2020 *Example: A Role is assigned to a particular Phase in a Lifecycle.*

2021

2022 **2.8.4 Interdependencies.**

Direct Dependencies	
Phase and Lifecycle Management Functional Element	The key abstraction, phases and lifecycle, in the Phase and Lifecycle Management Functional Element is used as a target for the assignment of roles and access privileges.
User Management Functional Element	The key abstraction, user, in the User Management Functional Element is used as a target for the assignment of roles and access privileges.
Group Management Functional Element	The key abstraction, group, in the Group Management Functional Element is used as a target for the assignment of roles and access privileges.

2023 **2.8.5 Related Technologies and Standards**

2024 None

2025

2.8.6 Model

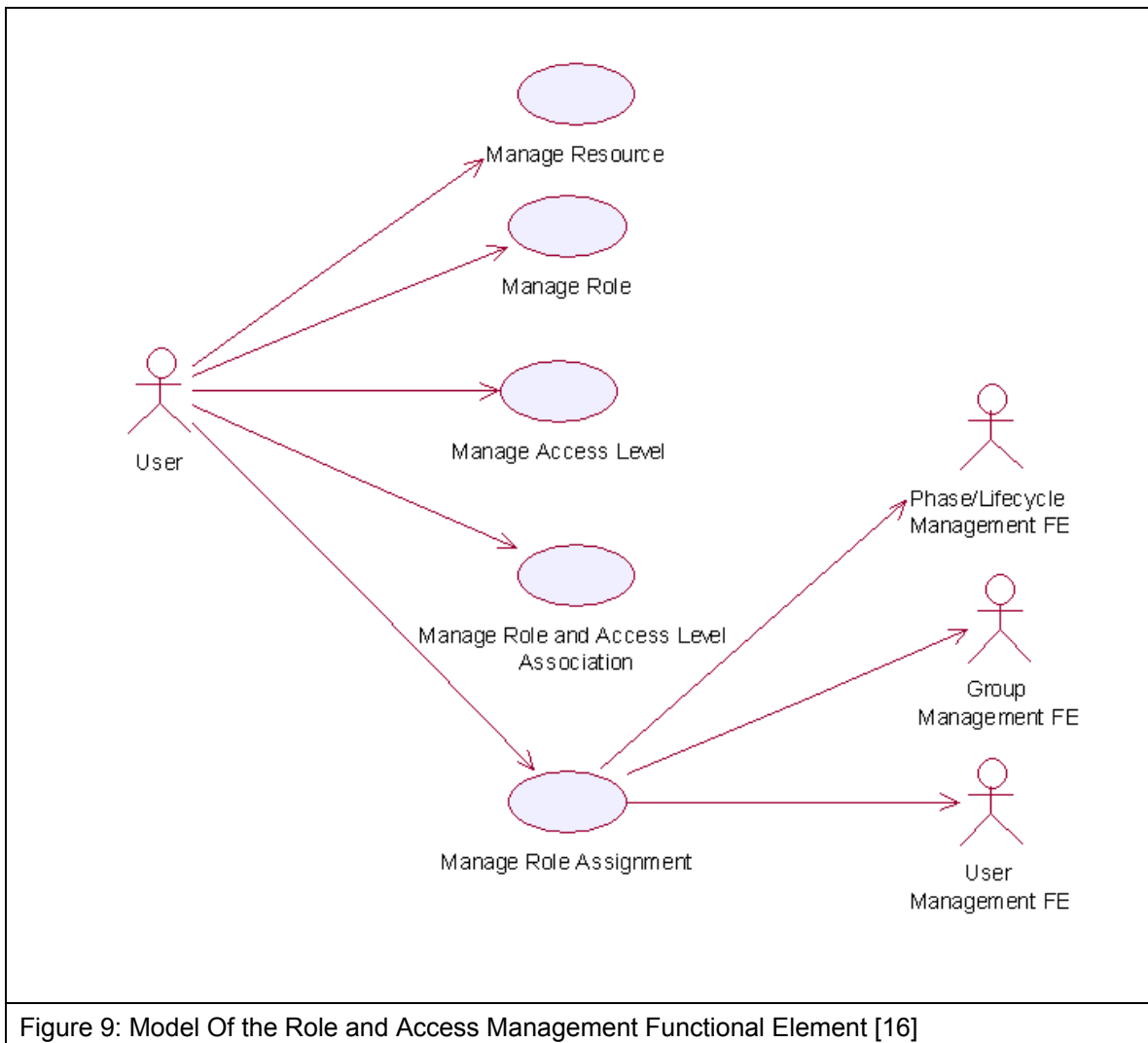


Figure 9: Model Of the Role and Access Management Functional Element [16]

2027

2028 2.8.7 Usage Scenario

2029 2.8.7.1 Manage Role

2030 2.8.7.1.1 Description

2031 This use case allows the service user to manipulate the role information such as adding,
 2032 changing and deleting role information in the Functional Element.

2033 2.8.7.1.2 Flow of Events

2034 2.8.7.1.2.1 Basic Flow

2035 This use case starts when any user wants to create, change or delete a role.

- 2036 1: Service user specifies the function it would like to perform (either create a role, update a role or
2037 delete a role).
- 2038 2: Once the service user provides the requested information, one of the sub-flows is executed.
- 2039 • If the service user provides '**Create a Role**', then sub-flow 2.1 is executed.
- 2040 • If the service user provides '**Retrieve a Role**', then sub-flow 2.2 is executed.
- 2041 • If the service user provides '**Update a Role**', then sub-flow 2.3 is executed.
- 2042 • If the service user provides '**Delete a Role**', then sub-flow 2.4 is executed.
- 2043 2.1: Create a Role.
- 2044 2.1.1: The service user specifies role information such as the role name and description.
- 2045 2.1.2: The Functional Element connects to the data storage.
- 2046 2.1.3: The Functional Element checks whether the role exists in the Functional Element
2047 or not, saves the role information in the data storage and the use case ends.
- 2048 2.2: Retrieve a Role.
- 2049 2.2.1: The service user specifies the role name for retrieval.
- 2050 2.2.2: The Functional Element connects to the data storage.
- 2051 2.2.3: The Functional Element retrieves the role information in the data storage and the
2052 use case ends.
- 2053 2.3: Update a Role.
- 2054 2.3.1: The service user specifies the role name to update.
- 2055 2.3.2: The service user specifies the target field name and value of the role.
- 2056 2.3.3: The Functional Element connects to the data storage.
- 2057 2.3.4: The Functional Element updates the role information in the data storage and the
2058 use case ends.
- 2059 2.4: Delete a Role.
- 2060 2.4.1: The service user specifies the role name to delete.
- 2061 2.4.2: The Functional Element connects to the data storage.
- 2062 2.4.3: The Functional Element removes the record of the role in the data storage and the
2063 use case ends.

2064 **2.8.7.1.2.2 Alternative Flows**

- 2065 1: Data Storage Not Available.
- 2066 1.1: If in basic flow 2.1.2, 2.2.2, 2.3.3 and 2.4.2, the data storage of the role information is not
2067 available, an error message is returned and the use case ends.
- 2068 2: Role Already Exists.

2069 2.1: If in basic flow 2.1.3, the Functional Element checks that the role already exists in the
2070 data storage, an error message is returned and the use case ends.

2071 3: Role Does Not Exist.

2072 3.1: If in basic flow 2.2.3, 2.3.4 and 2.4.3, the Functional Element checks that the role does
2073 not exist in the data storage, an error message is returned and the use case ends.

2074 4: Role Cannot Be Deleted.

2075 4.1: If in basic flow 2.4.3, the other information associated with the role, such as any access
2076 level assigned, still exists, the role information may not be removed. An error message is
2077 returned and the use case ends.

2078 **2.8.7.1.3 Special Requirements**

2079 None

2080 **2.8.7.1.4 Pre-Conditions**

2081 None.

2082 **2.8.7.1.5 Post-Conditions**

2083 If the use case was successful, the role is saved/updated/removed in the Functional Element.
2084 Otherwise, the Functional Element state is unchanged.

2085 **2.8.7.2 Manage Resource**

2086 **2.8.7.2.1 Description**

2087 This use case allows the service user to manipulate the resource information such as adding,
2088 changing and deleting resource information in the Functional Element.

2089 **2.8.7.2.2 Flow of Events**

2090 **2.8.7.2.2.1 Basic Flow**

2091 This use case starts when any user wants to create, change or delete a resource.

2092 1: The user specifies the function it would like to perform.

2093 2: The user provides the requested information, one of the sub-flows is executed.

2094 • If the user provides '**Create a Resource**', then sub-flow 2.1 is executed.

2095 • If the user provides '**Retrieve a Resource**', then sub-flow 2.2 is executed.

2096 • If the user provides '**Update a Resource**', then sub-flow 2.3 is executed.

2097 • If the user provides '**Delete a Resource**', then sub-flow 2.4 is executed.

2098 2.1: Create a Resource.

2099 2.1.1: The user specifies resource information such as the resource name and
2100 description.

2101 2.1.2: The Functional Element connects to the data storage.

2102 2.1.3: The Functional Element checks whether the resource exists in the Functional
2103 Element, saves the resource information in the data storage and the use case ends.

2104 2.2: Retrieve a Resource.

2105 2.2.1: The service user specifies the resource name for retrieval.

2106 2.2.2: The Functional Element connects to the data storage.

2107 2.2.3: The Functional Element retrieves the resource information in the data storage and
2108 the use case ends.

2109 2.3: Update a Resource.

2110 2.3.1: The service user specifies the resource name to update.

2111 2.3.2: The Functional Element connects to the data storage.

2112 2.3.3: The Functional Element updates the resource information in the data storage and
2113 the use case ends.

2114 2.4: Delete a Resource.

2115 2.4.1: The service user specifies the resource name to delete.

2116 2.4.2: The Functional Element connects to the data storage.

2117 2.4.3: The Functional Element removes the record of the resource in the data storage
2118 and the use case ends.

2119 **2.8.7.2.2 Alternative Flows**

2120 1: Data Storage Not Available.

2121 1.1: If in basic flow 2.1.2, 2.2.2, 2.3.2 and 2.4.2, the data storage of the resource information
2122 is not available, an error message is returned and the use case ends.

2123 2: Resource Already Exists.

2124 2.1: If in basic flow 2.1.3, the Functional Element checks that the resource already exists in
2125 the data storage, an error message is returned and the use case ends.

2126 3: Resource Does Not Exist.

2127 3.1: If in basic flow 2.2.3, 2.3.3 and 2.4.3, the Functional Element checks that the resource
2128 does not exist in the data storage, an error message is returned and the use case ends.

2129 **2.8.7.2.3 Special Requirements**

2130 None

2131 **2.8.7.2.4 Pre-Conditions**

2132 None.

2133 **2.8.7.2.5 Post-Conditions**

2134 None

2135 **2.8.7.3 Manage Access Level**

2136 **2.8.7.3.1 Description**

2137 This use case allows service user to manage the creation/retrieval/modification/deletion of access
2138 level.

2139 **2.8.7.3.2 Flow of Events**

2140 **2.8.7.3.2.1 Basic Flow**

2141 This use case starts when service user wants to manage the access levels.

2142 1: The service user specifies the function it would like to perform (add, update or delete an
2143 access level).

2144 2: Once the service user provides the requested information, one of the sub-flows is executed.

- 2145 • If the service user provides '**Add an Access Level**', then sub-flow 2.1 is executed.
- 2146 • If the service user provides '**Retrieve an Access Level**', then sub-flow 2.2 is activated.
- 2147 • If the service user provides '**Update an Access Level**', then sub-flow 2.3 is activated.
- 2148 • If the service user provides '**Delete an Access Level**', then sub-flow 2.4 is executed.

2149 2.1: Add an Access Level.

2150 2.1.1: The service user specifies the access level information, which includes: name,
2151 description, name of parent access level and group of resources that the access level is
2152 associated with.

2153 2.1.2: The Functional Element connects to the data storage.

2154 2.1.3: The Functional Element check whether the access level and its parent access level
2155 exist in the Functional Element, saves the access level information in the data storage
2156 and the use case ends.

2157 2.2: Retrieve an Access Level.

2158 2.2.1: The service user specifies the access level name to retrieve.

2159 2.2.2: The Functional Element connects to the data storage.

2160 2.2.3: The Functional Element gets access level information from the data storage and
2161 returns to the service user and the use case ends.

2162 2.3: Update an Access Level.

2163 2.3.1: The service user specifies the access level name.

2164 2.3.2: The service user specifies the field(s) and new value(s) to update.

2165 2.3.3: The Functional Element connects to the data storage.

2166 2.3.4: The Functional Element updates the access level information in the data storage
2167 with the value specified in 2.3.2 and the use case ends.

2168 2.4: Delete an Access Level.

2169 2.4.1: The service user specifies the access level name to delete.
2170 2.4.2: The Functional Element connects to the data storage.
2171 2.4.3: The Functional Element removes the record of the access level in the data storage
2172 and the use case ends.

2173 **2.8.7.3.2 Alternative Flows**

2174 1: Data Storage Not Available.

2175 1.1: If in basic flow 2.1.2, 2.2.2, 2.3.3 and 2.4.2, the data storage of the access level
2176 information is not available, an error message is returned and the use case ends.

2177 2: Access Level Already Exists.

2178 2.1: If in basic flow 2.1.3, the Functional Element checks that the access level already exists
2179 in the data storage, an error message is returned and the use case ends.

2180 3: Access Level Cannot Be Deleted.

2181 3.1: If in basic flow 2.4.3, the other information associated with the Access Level, such as
2182 roles to which the access level is assigned and the parent access level still exists, the access
2183 level information may not be removed. An error message is returned and the use case ends.

2184 4: Parent Access Level Not Exist.

2185 4.1: If in basic flow 2.1.3, the parent access level does not exist, an error message is returned
2186 and the use case ends.

2187 **2.8.7.3.3 Special Requirements**

2188 None

2189 **2.8.7.3.4 Pre-Conditions**

2190 None.

2191 **2.8.7.3.5 Post-Conditions**

2192 None

2193 **2.8.7.4 Manage Role and Access Level Association**

2194 **2.8.7.4.1 Description**

2195 This use case allows service user to assign, update and remove the access level assigned to
2196 role.

2197 **2.8.7.4.2 Flow of Events**

2198 **2.8.7.4.2.1 Basic Flow**

2199 This use case starts when service user wants to manage the relationship between access level
2200 and role.

2201 1: The service user specifies a role and the function he/she would like to perform on the role
2202 (either assign an access level to role, update role access level, or delete role access level).

- 2203 2: Once the service user provides the requested information, one of the sub-flows is executed.
- 2204 • If the user provides '**Assign an Access Level to Role**', then sub-flow 2.1 is executed.
- 2205 • If the user provides '**Update Access Level for Role**', then sub-flow 2.2 is executed.
- 2206 • If the user provides '**Delete Access Level for Role**', then sub-flow 2.3 is executed.
- 2207 • If the user provides '**Retrieve Access Level for Role**', then sub-flow 2.4 is executed.
- 2208 • If the service user provides '**Retrieve Role for Access Level**', then sub-flow 2.5 is
- 2209 executed.
- 2210 2.1: Assign an Access Level to Role.
- 2211 2.1.1: The service user specifies access level that will be assigned to the role.
- 2212 2.1.2: The Functional Element connects to the data storage.
- 2213 2.1.3: The Functional Element checks whether the access level has been assigned to the
- 2214 role. Functional Element saves the access level reference in the role record in the data
- 2215 storage and the use case ends.
- 2216 2.2: Update Access Level for Role.
- 2217 2.2.1: The service user specifies the access level to update and the new access level
- 2218 information.
- 2219 2.2.2: The Functional Element connects to the data storage.
- 2220 2.2.3: The Functional Element updates the access level reference in the role record in the
- 2221 data storage and the use case ends.
- 2222 2.3: Delete Access Level to Role.
- 2223 2.3.1: The service user specifies the access level to delete.
- 2224 2.3.2: The Functional Element connects to the data storage.
- 2225 2.3.3: The Functional Element removes the access level reference from the record of the
- 2226 role in the data storage and the use case ends.
- 2227 2.4: Retrieve Access Level for Role.
- 2228 2.4.1: The service user specifies the role to retrieve the access levels associated with it.
- 2229 2.4.2: The Functional Element connects to the data storage.
- 2230 2.4.3: The Functional Element retrieves the access level assigned to the role in the data
- 2231 storage and the use case ends.
- 2232 2.5: Retrieve Role for Access Level.
- 2233 2.5.1: The service user specifies the access level to retrieve roles associated to it.
- 2234 2.5.2: The Functional Element connects to the data storage.
- 2235 2.5.3: The Functional Element retrieves roles associated to the access level in the data
- 2236 storage and the use case ends.

2237 **2.8.7.4.2 Alternative Flows**

2238 1: Data Storage Not Available.

2239 1.1: If in basic flow 2.1.2, 2.2.2 and 2.3.2, the data storage of the access level information is
2240 not available, an error message is returned and the use case ends.

2241 2: Access Level Assignment Already Exists.

2242 2.1: If in basic flow 2.1.3, the Functional Element checks that the access level already exists
2243 in the role record in the data storage, an error message is returned and the use case ends.

2244 3: Access Level Assignment Not Exist.

2245 3.1: If in basic flow 2.3.3, the access level assignment does not exist, an error message is
2246 returned and the use case ends.

2247 4: Access Level Not Exist.

2248 4.1: If in basic flow 2.1.3, 2.2.3, 2.3.3, 2.4.3 and 2.5.3, the access level does not exist, an
2249 error message is returned and the use case ends.

2250 5: Role Not Exist.

2251 5.1: If in basic flow 2.1.3, 2.2.3, 2.3.3, 2.4.3 and 2.5.3, the role does not exist, an error
2252 message is returned and the use case ends.

2253 **2.8.7.4.3 Special Requirements**

2254 None.

2255 **2.8.7.4.4 Pre-Conditions**

2256 None.

2257 **2.8.7.4.5 Post-Conditions**

2258 None.

2259 **2.8.7.5 Manage Role Assignment**

2260 **2.8.7.5.1 Description**

2261 The use case allows service user to assign a role to a user, a group, a phase in a lifecycle, to
2262 change or to delete such assignment.

2263 **2.8.7.5.2 Flow of Events**

2264 **2.8.7.5.2.1 Basic Flow**

2265 This use case starts when the service user wants to manage the assignment of a role. This role
2266 can be assigned to a user, group, phase and lifecycle.

2267 1: Service user specifies a role and an operation to perform on the role.

2268 2: Once the service user provides the requested information, one of the sub-flows is executed.

2269 • If the user provides '**Assign Role**', then sub-flow 2.1 is executed.

- 2270 • If the user provides '**Retrieve Role**', then sub-flow 2.2 is executed.
- 2271 • If the user provides '**Un-assign Role**', then user sub-flow 2.3 is executed.
- 2272 2.1: Assign Role.
- 2273 2.1.1: The service user specifies a user/group/phase/lifecycle to which the role will be
2274 assigned.
- 2275 2.1.2: Depending of target of the assignment, the Functional Element will check for the
2276 presence of one of the following Functional Elements.
- 2277 • User Management Functional Element
- 2278 • Group Management Functional Element
- 2279 • Phase and Lifecycle Management Functional Element
- 2280 2.1.3: The Functional Element checks whether the role has been assigned to the
2281 intended target
- 2282 2.1.4: The Functional Element saves the relationship between the role and the target and
2283 the use case ends.
- 2284 2.2: Retrieve Role.
- 2285 2.2.1: The service user specifies a user/group/phase/lifecycle to retrieve all roles
2286 assigned
- 2287 2.2.2: Depending of target of the assignment, the Functional Element will check for the
2288 presence of one of the following Functional Elements.
- 2289 • User Management Functional Element
- 2290 • Group Management Functional Element
- 2291 • Phase and Lifecycle Management Functional Element
- 2292 2.2.3: The Functional Element gets the roles that are assigned to the target.
- 2293 2.2.4: The Functional Element returns the results to the service user and the use case
2294 ends.
- 2295 2.3: Un-assign Role.
- 2296 2.3.1: The service user specifies a user/group/phase/lifecycle and the role that is to be
2297 un-assigned.
- 2298 2.3.2: Depending of target of this un-assignment, the Functional Element will check for
2299 the presence of one of the following Functional Elements.
- 2300 • User Management Functional Element
- 2301 • Group Management Functional Element
- 2302 • Phase and Lifecycle Management Functional Element
- 2303 2.3.3: The Functional Element checks if the roles have been assigned to the target in the
2304 first place.
- 2305 2.3.4: The Functional Element removes the role assigned and the use case ends.

2306 **2.8.7.5.2 Alternative Flows**

2307 1: Dependent Functional Element not available.

2308 1.1: If in basic flow 2.1.2, 2.2.2 and 2.3.2, the dependent Functional Elements are not
2309 available, an error message is returned and the use case ends.

2310 2: Invalid User/Group/Phase/Lifecycle Account.

2311 2.1: If in basic flow 2.1.2, 2.2.2 and 2.3.2, the dependent Functional Elements are available
2312 but an invalid account is provided, an error message is returned and the use case ends.

2313 3: Data Storage Not Available.

2314 3.1: If in basic flow 2.1.2, 2.2.2 and 2.3.2, the Functional Element is unable to access the data
2315 storage, an error message is provided and the use case ends.

2316

2317 **2.8.7.5.3 Special Requirements**

2318 None.

2319 **2.8.7.5.4 Pre-Conditions**

2320 None.

2321 **2.8.7.5.5 Post-Conditions**

2322 None.

2323 2.9 Search Functional Element

2324 2.9.1 Motivation

2325 In a Web Service-enabled implementation, information is distributed across different sites and this
2326 makes searching and collating information difficult. Against this backdrop, this Functional
2327 Element is expected to fulfill the needs identified within an application by covering the following
2328 aspects.

- 2329 • Providing the capability for configuration of different types of data sources for information
2330 search,
- 2331 • Providing the facility to provide a concrete definition of data source classification for
2332 information search,
- 2333 • Providing the ability to define different search scopes for various data source
2334 classification,
- 2335 • Performing information search on those pre-configured different types of data sources
2336 and
- 2337 • Providing the provision to consolidate the return result arising from the search operation.

2338

2339 This Functional Element fulfills the following requirements from the Functional Elements
2340 Requirements, Working Draft 01a:

- 2341 • Primary Requirements
 - 2342 • MANAGEMENT-009,
 - 2343 • PROCESS-030 to PROCESS-031, and
 - 2344 • PROCESS-034.
- 2345 • Secondary Requirements
 - 2346 • None

2347

2348 2.9.2 Terms Used

Terms	Description
Data source	Data source refers to any kind of information storage and retrieval databases like RDBMS, LDAP, ODBMS, XMLDB, XML Files, TEXT Files, etc.
Search Category	A Search Category refers to some logical grouping of the data sources on the basis of purpose of various data source purpose like NEWS, EMAIL, USERS, GROUPS, TRANSACTIONS, etc.
Data Source Type	Data Source Type refers to the various kinds of data storage format or structure like XML, HTML, TEXT, Databases, Tables, Rows, Columns in RDBMS, Collections, Nodes, Files & Tags in XMLDB, that are used to store and retrieve information from different data sources
RDBMS	Relational Database Management Systems

XMLDB	eXtensible Markup Language (XML) Database
LDAP	Lightweight Directory Access Protocol
XML	eXtensible Markup Language
HTML	HyperText Markup Language

2349 **2.9.3 Key Features**

2350 Implementations of the Search Functional Element are expected to provide the following key
2351 features:

- 2352 1. The Functional Element MUST provide a mechanism to define and manage Search
2353 Categories.
- 2354 2. The Functional Element MUST provide the capability to configure and store information
2355 about targeted data sources for a particular Search Category.
2356 *Example: Some of the stored information would include Location, Type, Name, Data Fields*
2357 *(of interest to the search) and access control (typically username and password) of the*
2358 *targeted data source.*
- 2359 3. As part of Key Feature (2), the Functional Element MUST also provide the ability to
2360 configure the scope of search and returned results.
- 2361 4. The Functional Element MUST also provide a mechanism to link the Search Categories to
2362 configured target data sources.
- 2363 5. The Functional Element MUST provide the ability to search multiple data sources for a
2364 defined Search Category.
2365 *Example: Some of the common data sources would include RDBMS, XML DB, LDAP*
2366 *servers and flat files like XML files, text files and HTML files*
- 2367 6. The Functional Element MUST provide the ability to perform searches based on a given set
2368 of keyword(s).

2369

2370 In addition, the following key features could be provided to enhance the Functional Element
2371 further:

- 2372 1. The Functional Element MAY also provide the ability to perform conditional and parametric
2373 searches.
- 2374 2. The Functional Element MAY also provide the ability to restrict the scope of a search.
2375 *Example: By providing a particular Search Category or types of data sources for the*
2376 *search.*

2377

2378 **2.9.4 Interdependencies**

2379 None

2380

2381 **2.9.5 Related Technologies and Standards**

2382 None

2383 **2.9.6 Model**

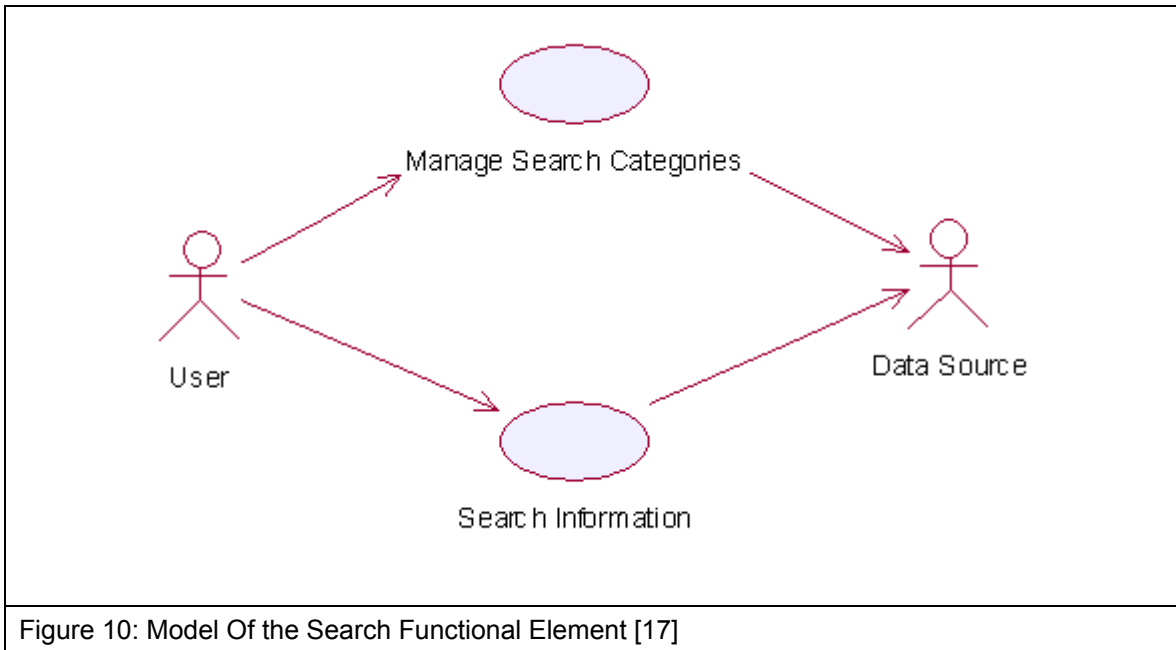


Figure 10: Model Of the Search Functional Element [17]

2384 **2.9.7 Usage Scenario**

2385 **2.9.7.1 Manage Search Categories**

2386 **2.9.7.1.1 Description**

2387 This use case allows the users to manage the different search categories.

2388 **2.9.7.1.2 Flow of Events**

2389 **2.9.7.1.2.1 Basic Flow**

2390 This use case starts when the user wishes to manage the different data sources for search to be
2391 performed on it.

2392 1: The users initiates a request to configure data source(s) and type(s) by providing the data
2393 source information and type to be added, removed or retrieved.

2394 2: The Functional Element checks whether the data source configuration file exists.

2395 3: The Functional Element checks the request. Based on the type of request, one of the sub-
2396 flows is executed.

2397 • If the request is to **'Create Data Source And Type'**, then sub-flow 3.1 is executed.

2398 • If the request is to **'View Data Sources And Types'**, then sub-flow 3.2 is executed.

2399 • If the request is to **'Delete Data Source And Type'**, then sub-flow 3.3 is executed.

2400 3.1: Create Data Source and Type.

2401 3.1.1: The Functional Element checks whether the same data source and type has been
2402 created.

2403 3.1.2: The Functional Element appends the new data source and type in the data source
2404 configuration file specified.

2405 3.2: View Data Source and Type.

2406 3.2.1: The Functional Element retrieves all the data source and type information from the
2407 data source configuration file.

2408 3.2.2: The Functional Element returns the data source(s) and type(s).

2409 3.3: Delete Data Source and Type.

2410 3.3.1: The Functional Element checks whether the data source and type exist in the data
2411 source configuration based on data source id from the data source configuration file.

2412 3.3.2: The Functional Element removes the old data source and type from the data
2413 source configuration file.

2414 4: The Functional Element returns a success or failure flag indicating the status of the operation
2415 being performed and use case ends.

2416 **2.9.7.1.2.2 Alternative Flows**

2417 1: Data Source Configuration File Not Found.

2418 1.1: If in basic flow 2, the data source configuration file does not exist, the Functional Element
2419 creates an empty data source configuration file.

2420 2: Duplicate Data Source and Type.

2421 2.1: If in basic flow 3.1.1, the same data source and type have been configured, the
2422 Functional Element returns an error message and the use case end.

2423 3: Data Source and Type Do Not Exist.

2424 3.1: If in basic flow 3.2.1 and 3.3.1, a particular data source and type cannot be found in the
2425 specified data source configuration file, the Functional Element returns an error message and
2426 the use case end.

2427 **2.9.7.1.3 Special Requirements**

2428 None.

2429 **2.9.7.1.4 Pre-Conditions**

2430 None.

2431 **2.9.7.1.5 Post-Conditions**

2432 None.

2433 **2.9.7.2 Search Information**

2434 **2.9.7.2.1 Description**

2435 This use case allows any users to perform search on various disparate data sources and types
2436 configured to be searched and returns the matching results.

2437 **2.9.7.2.2 Flow of Events**

2438 **2.9.7.2.2.1 Basic Flow**

2439 This use case starts when users wishes to perform information search on a data source.

2440 1: Users initiates a request to perform information search on a given data source by providing
2441 information to be searched, location of the data source(s) and the data source type(s).

2442 2: The Functional Element checks for the existence of the specified data source(s).

2443 3: The Functional Element validates the data source type(s) against the set of supported data
2444 type(s) configured within the Functional Element that are available for information search.

2445 4: The Functional Element performs information search based on the search parameters given by
2446 the users or the other Functional Elements.

2447 5: The Functional Element returns the result of the information search performed to the users or
2448 other Functional Elements and use case ends.

2449 **2.9.7.2.2.2 Alternative Flows**

2450 1: Data Source(s) Are Not Available.

2451 1.1: In basic flow 2, if the identified data source is not available, the Functional Element
2452 returns an error message and the use case ends.

2453 2: Invalid Configuration Instructions

2454 2.1: In basic flow 2, if the input inform by the user is incomplete, the Functional Element
2455 returns an error message and the use case ends.

2456 3: Invalid Data Source Type.

2457 3.1: In basic flow 3, if the data source type is invalid, the Functional Element returns an error
2458 message and the use case ends.

2459 4: No Matching Result.

2460 4.1: In basic flow 4, if the search results in no matching results, the Functional Element
2461 returns an error message and the use case ends..

2462 **2.9.7.2.3 Special Requirements**

2463 None

2464 **2.9.7.2.4 Pre-Conditions**

2465 None.

2466 **2.9.7.2.5 Post-Conditions**

2467 None.

2468

2469 **2.10 Secure SOAP Management Functional Element**

2470 **2.10.1 Motivation**

2471 In a Web Services implementation, it is envisaged that confidential information is being exchanged
2472 all the time. Against this backdrop, it is imperative that an application in such an environment is
2473 equipped with the capability to guard sensitive information from prying eyes. Secure SOAP
2474 Management fulfills this need by covering the following areas.

- 2475 • The facility of digitally signing SOAP message,
- 2476 • The facility of encrypting SOAP message, and
- 2477 • The capability to generate the original SOAP message after signing or encrypting the
2478 message.

2479

2480 This Functional Element fulfills the following requirements from the Functional Elements
2481 Requirements, Working Draft 01a:

- 2482 • Primary Requirements
 - 2483 • SECURITY-003 (SECURITY-003-3 only),
 - 2484 • SECURITY-020 (all), and
 - 2485 • SECURITY-022, and
 - 2486 • SECURITY-026.
- 2487 • Secondary Requirements
 - 2488 • None

2489

2490 **2.10.2 Terms Used**

Terms	Description
Digital Signature	An electronic signature that can be used to authenticate the identity of the sender of a message, or of the signer of a document. It can also be used to ensure that the original content of the message or document that has been conveyed is unchanged
Encryption	A method of scrambling or encoding data to prevent unauthorized users from reading or tampering with the data. Only individuals with access to a password or key can decrypt and use the data.
PKCS#11	The cryptographic token interface standards. Defines a technology independent programming interface for cryptographic devices such as smart cards.
Public Key Cryptography Specification (PKCS)#12	The personal information exchange syntax standard. Defines a portable format for storage and transportation of user private keys, certificates etc.

2491

2492

2.10.3 Key Features

2493

Implementations of the Group Management Functional Element are expected to provide the following key features:

2494

2495

1. The Functional Element MUST provide the capability to digitally sign SOAP messages completely or partially using XML-Signature Syntax and Processing, W3C Recommendation 12 February 2002.

2496

2497

2498

2. The Functional Element MUST provide the capability to validate a signed SOAP message.

2499

2500

2501

3. The Functional Element MUST provide the capability to encrypt SOAP messages completely or partially using XML-Encryption Syntax and Processing, W3C Recommendation 10 December 2002.

2502

4. The Functional Element MUST provide the capability to decrypt encrypted SOAP messages.

2503

5. The Functional Element MUST support PKCS12 compatible digital certificates.

2504

2505

6. The Functional Element MUST be able to verify the validity and authenticity of digital certificates used.

2506

2507

In addition, the following key features could be provided to enhance the Functional Element further:

2508

2509

1. The Functional Element MAY also support PKCS11 compatible tokens.

2510

2. The Functional Element MAY also provide log support as part of the audit trails for its transaction records.

2511

2512

2513

2.10.4 Interdependencies

Direct Dependency

Log Utility Functional Element

The Log Utility Functional Element is being used for logging and creation of audit trails.

2514

2.10.5 Related Technologies and Standards

Standards / Specifications	Specific References
Public Key Infrastructure (PKI)	PKI is a system of digital certificates, Certificate Authorities, and other registration authorities that verify and authenticate the validity of each party involved in an Internet transaction In this Functional Element, the private key and public key are generated for the Functional Element to sign and encrypt SOAP messages. The Functional Element uses the session key to encrypt the SOAP message. The digital certificate is attached to the SOAP message after the Functional Element has signed the SOAP message.
XML-Signature Syntax and Processing, W3C Recommendation 12 th Feb 2002 [18]	This specification addresses authentication, non-repudiation and data-integrity issues. In addition, it also specifies the XML syntax and processing rules for creating and representing digital signatures. In this Functional Element, both the digital signature on the SOAP message and validation of the signed SOAP message is done based on this specification.

XML-Encryption Syntax and Processing, W3C Recommendation 10 th Dec 2002 [19]	This specification addresses data privacy by defining a process for encrypting data and representing the result in XML document. In this Functional Element, the encryption and decryption of SOAP messages are done based on this specification.
---	--

2515

2516

2517 **2.10.6 Model**

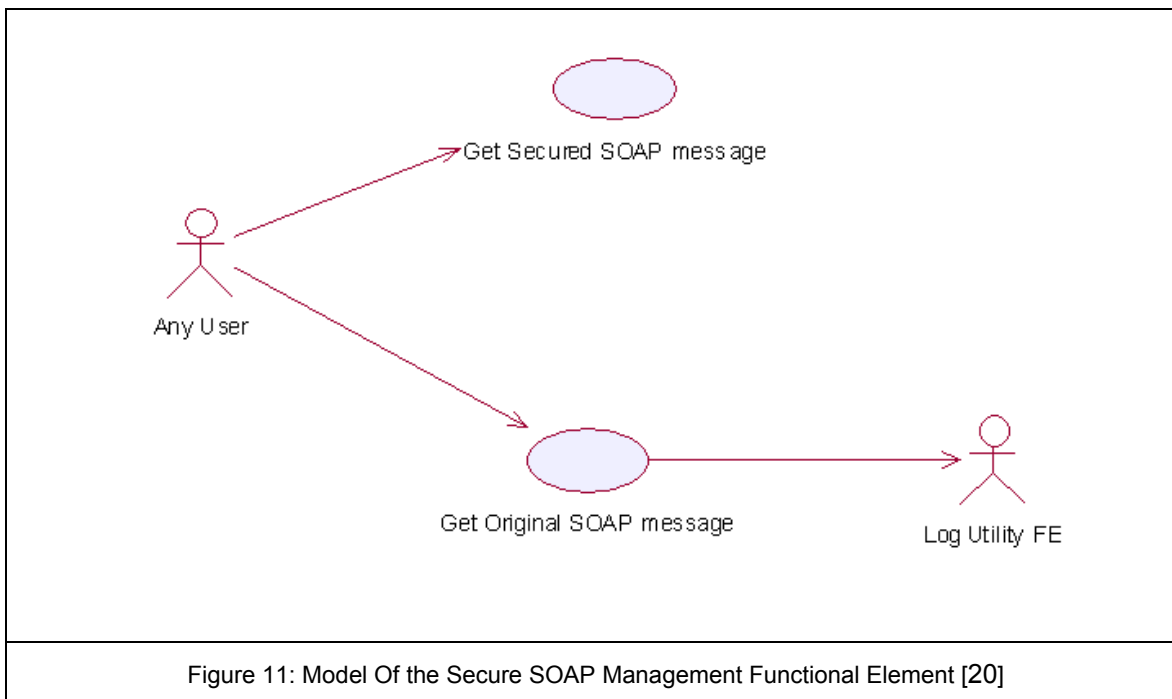


Figure 11: Model Of the Secure SOAP Management Functional Element [20]

2518 **2.10.7 Usage Scenarios**

2519 **2.10.7.1 Get Secured SOAP message**

2520 **2.10.7.1.1 Description**

2521 This Functional Element describes the process to generate secured SOAP message.

2522 **2.10.7.1.2 Flow of Events**

2523 **2.10.7.1.2.1 Basic Flow**

2524 This use case starts when the user wants to secure the SOAP message.

- 2525 • If user wants to 'Sign SOAP message', then basic flow 1 is executed.
- 2526 • If user wants to 'Encrypt and Sign the SOAP message', then basic flow 2 is executed.

2527 1: Sign SOAP Message.

2528 1.1: User sends the SOAP message, digital certificate and specifies the element name that
2529 needs to be signed.

2530 1.2: Functional Element gets the key information from the digital certificate.

2531 *Note: The private key will be used to sign the SOAP message and the public key will be*
2532 *added to the SOAP message after the signing.*

2533 1.3: Functional Element signs the element.

2534 *Note: The digital signature format is expected to be based on XML-Digital Signature Syntax*
2535 *mentioned in section 3.10.5.*

2536 1.4: Functional Element parses the secure SOAP message and regenerates the SOAP
2537 message.

2538 1.5: Functional Element returns the secured SOAP message to user and the use case ends.

2539 2: Encrypt And Sign SOAP Message.

2540 2.1: User sends the SOAP message, digital certificate and specify the element name that
2541 needs to be encrypted.

2542 2.2: User sends the receiver's public key information to Functional Element.

2543 *Note: Receiver's public key will be used to encrypt the session key, which was then used to*
2544 *encrypt the content of the element in the SOAP message.*

2545 2.3: Functional Element gets key information from the user's digital certificate.

2546 *Note: Private key is used to sign the SOAP message and public key is used to add into the*
2547 *SOAP message after the signing.*

2548 2.4: Functional Element generates the session key.

2549 *Note: Session key is used to encrypt the content of the element.*

2550 2.5: Functional Element encrypts the content of element with the session key.

2551 2.6: Functional Element encrypts session key with the receiver's public key.

2552 2.7: Functional Element signs the SOAP message after encryption.

2553 2.8: Functional Element regenerates the SOAP message.

2554 *Note: Functional Element adds the encrypted content of the element, encrypted session key*
2555 *information, the receiver's public key information and the signature to the SOAP message.*

2556 2.9: Functional Element returns the SOAP message and the use case ends.

2557 **2.10.7.1.2.2 Alternative Flows**

2558 1: Cannot Get Key.

2559 1.1: In basic flow 1.2 and 2.3, Functional Element cannot get the key information from the
2560 digital certificate. The Functional Element returns an error message and the use case ends.

2561 2: Cannot Sign

2562 2.1: In basic flow 1.3, Functional Element cannot sign the SOAP message. The Functional
2563 Element returns an error message and the use case ends.

2564 3: Cannot Encrypt

2565 3.1: In basic flow 2.5, Functional Element cannot encrypt the SOAP message. The Functional
2566 Element returns an error message and the use case ends.

2567 **2.10.7.1.3 Special Requirements**

2568 None.

2569 **2.10.7.1.4 Pre-Conditions**

2570 None.

2571 **2.10.7.1.5 Post-Conditions**

2572 None.

2573 **2.10.7.2 Get Original SOAP Message**

2574 **2.10.7.2.1 Description**

2575 This use case allows users to get original SOAP message.

2576 **2.10.7.2.2 Flow of Events**

2577 **2.10.7.2.2.1 Basic Flow**

2578 This use case starts when the user wants to get the original SOAP message.

2579

- If the user wants to ‘**Verify the SOAP message**’, then basic flow 1 is executed.

2580

- If the user wants to ‘**Decrypt and Verify the SOAP message**’, then basic flow 2 is

2581 executed.

2582 1: Verify SOAP Message.

2583 1.1: User sends the SOAP message and sender’s digital certificate.

2584 1.2: Functional Element verifies the SOAP message.

2585 *Note: The sender’s certificate information will be used to verify the signature.*

2586 1.3: Functional Element gets the original SOAP message, returns to user and the use case
2587 ends.

2588 2: Decrypt And Verify The SOAP Message.

2589 2.1: User sends the SOAP message, user’s digital certificate and sender’s certificate.

2590 2.2: Functional Element verifies the SOAP message.

2591 *Note: The sender’s certificate information will be used to verify the signature.*

2592 2.3: Functional Element gets the user’s key information from the user’s digital certificate.

2593 *Note: The user’s private key will be used to decrypt the session key.*

- 2594 2.4: Functional Element decrypts the session key.
- 2595 2.5: Functional Element decrypts the content of the element with the session key.
- 2596 2.6: Functional Element regenerates the SOAP message.
- 2597 *Note: Functional Element removes the session key information and the digital signature*
2598 *information from the SOAP message and gets the original one.*
- 2599 2.7: Functional Element returns the original SOAP message to user and the use case ends.

2600 **2.10.7.2.2.2 Alternative Flows**

- 2601 1: Verification Fails.
- 2602 1.1: In basic flow 1.3 and 2.3, if verification fails, the Functional Element returns an error
2603 message and the use case ends.
- 2604 2: Decryption of Content Fails.
- 2605 2.1: In basic flow 2.5, the Functional Element cannot decrypt the content of the element. The
2606 Functional Element returns an error message and the use case ends.

2607 **2.10.7.2.3 Special Requirements**

2608 None

2609 **2.10.7.2.4 Pre-Conditions**

2610 None.

2611 **2.10.7.2.5 Post-Conditions**

2612 None.

2613 **2.11 Sensory Functional Element**

2614 **2.11.1 Motivation**

2615 In a Web Service implementation where the presentation capabilities of clients differ, there is a
2616 need to determine the exact ability of the end devices so that the appropriate contents may be
2617 forwarded. The Sensory Functional Element can help to play this role by covering the following
2618 aspects within an application:

- 2619
- Determining the presentation capabilities by inspecting incoming headers, and
 - Determining the presentation capabilities by extracting MIME information from the relevant headers.
- 2620
2621

2622

2623 This Functional Element fulfills the following requirements from the Functional Elements
2624 Requirements, Working Draft 01a:

- 2625
- Primary Requirements
 - DELIVERY-001,
 - DELIVERY-005 to DELIVERY-006, and
 - DELIVERY-009.
 - Secondary Requirements
 - MANAGEMENT-011, and
 - MANAGEMENT-096.
- 2626
2627
2628
2629
2630
2631
2632

2633 **2.11.2 Terms Used**

Terms	Description
HTTP	Hyper Text Transport Protocol [HTTP] refers to the protocol for moving hypertext files across the Internet. Requires a HTTP client program on one end, and an HTTP server program on the other end. HTTP is the most important protocol used in the World Wide Web (WWW).
MIME	Multipurpose Internet Mail Extensions (MIME) refers to a standard that allows the embedding of arbitrary documents and other binary data of known types (images, sound, video, and so on) into e-mail handled by ordinary Internet electronic mail interchange protocols
Location Based Services (LBS)	Location-based services (LBS) refer to the services that provides users of mobile devices personalized services tailored to their current location.

2634

2635 **2.11.3 Key Features**

2636 Implementations of the Sensory Functional Element are expected to provide the following key
2637 features:

- 2638 1. The Functional Element MUST intercept requests from client and determines existing
2639 supportability of the request's MIME type.

2640 2. The Functional Element MUST provide the mechanism to manage MIME types, including
 2641 the ability to add, delete and retrieve supported MIME types.

2642

2643 In addition, the following key features could be provided to enhance the Functional Element
 2644 further:

2645 1. The Functional Element MAY provide a mechanism to enable Location Based Services
 2646 (LBS).

2647 **2.11.4 Interdependencies**

Interaction Dependency	
Presentation Transformer Functional Element	The Presentation Transformer Functional Element may be used to generate the appropriate output for the targeted devices.

2648 **2.11.5 Related Technologies and Standards**

2649 None.

2650

2651 **2.11.6 Model**

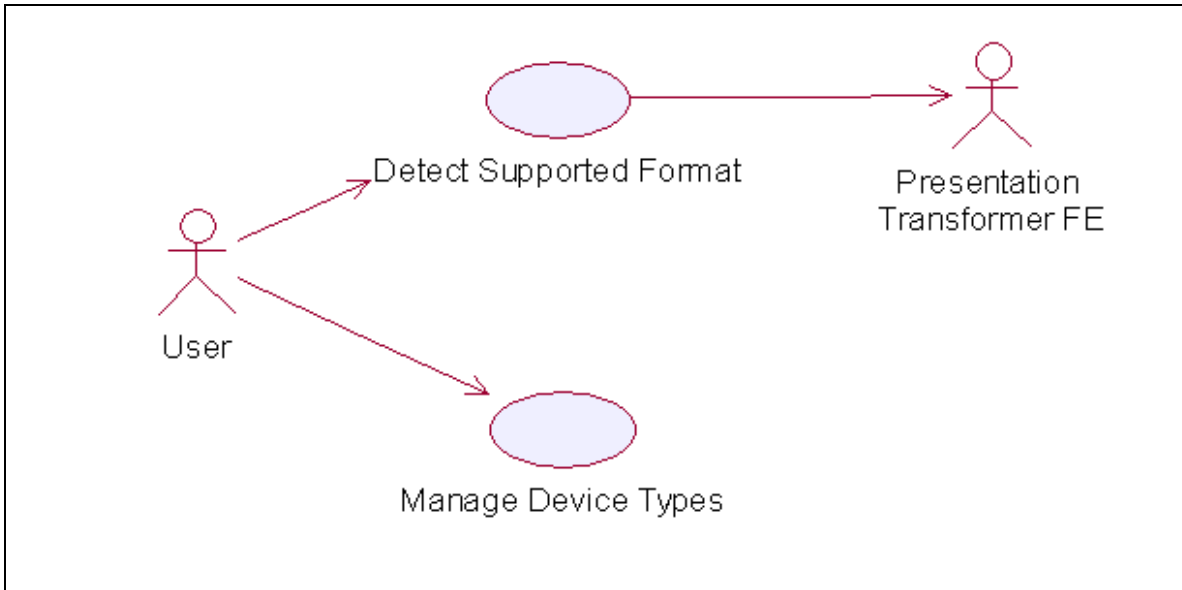


Figure 12: Model Of the Sensory Functional Element [21]

2652 **2.11.7 Usage Scenarios**

2653 **2.11.7.1 Detect Supported Format**

2654 **2.11.7.1.1 Description**

2655 This use case allows the service user (user/other service) to make request and based on that
 2656 request it detects service user's device capabilities.

2657 **2.11.7.1.2 Flow of Events**

2658 **2.11.7.1.2.1 Basic Flow**

2659 This use case starts when the service user wishes to use any service provided by the service
2660 provider.

2661 1: The Functional Element receives the request from the service user.

2662 2: The Functional Element extracts MIME name and MIME type from the service user's HTTP
2663 request (even form SOAP request).

2664 3: The Functional Element uses MIME name and MIME TYPE to check with the registered MIME
2665 type.

2666 4: The Functional Element sends device capabilities to service user and ends the use case.

2667 **2.11.7.1.2.2 Alternative Flows**

2668 1: Unsupported Device.

2669 1.1 If in the basic flow 2, the Functional Element is unable to detect the service user' device
2670 capability, the Functional Element returns a error message and the use case ends.

2671 **2.11.7.1.3 Special Requirements**

2672 None

2673 **2.11.7.1.3.1 Supportability**

2674 The edge devices must be able to support the HTTP request.

2675 **2.11.7.1.4 Pre-Conditions**

2676 None.

2677 **2.11.7.1.5 Post-Conditions**

2678 None.

2679 **2.11.7.2 Manage Device Types**

2680 **2.11.7.2.1 Description**

2681 This use case allows the service user to maintain the device (MIME Type information). This
2682 includes adding, changing and deleting device information from the Functional Element.

2683 **2.11.7.2.2 Flow of Events**

2684 **2.11.7.2.2.1 Basic Flow**

2685 This use case starts when the service user wishes to add or delete either device or service
2686 information from the Functional Element.

2687 1: The Functional Element requests that the service user specify the function he/she would like to
2688 perform (either add, update or delete device or service).

- 2689 2: Once the service user provides the requested information, one of the sub-flows is executed.
- 2690 • If the service user provides '**Add Device Types**', then sub-flow 2.1 is executed.
- 2691 • If the service user provides '**Delete Device Types**', then sub-flow 2.2 is executed.
- 2692 2.1: Add Device Type.
- 2693 2.1.1: The Functional Element requests that the service user provide the device
2694 information. This includes: MIME Name, MIME Description, Supported MIME type.
- 2695 2.1.2: Once the service user provides the requested information, the Functional Element
2696 generates and assigns a unique MIME Id number to the device.
- 2697 2.2: Delete Device Type.
- 2698 2.2.1: The Functional Element requests that the service user provide the Device ID.
- 2699 2.2.2: The Functional Element retrieves the existing device information based on the
2700 Device ID.
- 2701 2.2.3: The service user provides the delete device information and the Functional
2702 Element deletes the device record from the Functional Element.
- 2703 3: The use case ends when the service user provides the requested information or decided to
2704 end use case.

2705 **2.11.7.2.2.2 Alternative Flows**

- 2706 1: Invalid Device Information.
- 2707 1.1: If in the sub-flow 2.1.2, the requested information provided by the user is invalid, the
2708 Functional Element returns an error message and the use case ends
- 2709 2: Device Not Found.
- 2710 2.1 If in the basic flows 2.2.2, the device information with the specified device is not found or
2711 does not exist, the Functional Element returns an error message and the use case ends.

2712 **2.11.7.2.3 Special Requirements**

2713 **2.11.7.2.3.1 Supportability**

2714 Manage Device Types supports the most widespread MIME types used today.

2715 **2.11.7.2.4 Pre-Conditions**

2716 None.

2717 **2.11.7.2.5 Post-Conditions**

2718 If the use case was successful, the device information is added, updated or deleted from the
2719 Functional Element. Otherwise, the Functional Element's state is unchanged.

2720 **2.12 Service Management Functional Element**

2721 **2.12.1 Motivation**

2722 The ability to monitor Web Services invocation is crucial towards the adoption of this technology
2723 from the security and performance standpoints. A security framework should incorporate an
2724 authentication and authorisation mechanism together with an audit trail. These twin
2725 considerations will serve to discourage resource misuse and in addition, will help to promote the
2726 “pay-as-you-use” concept. Service throughput on the server end is another important parameter
2727 that must be monitored. Administrators of services, which are sluggish, should be notified
2728 immediately via any electronic means.

2729

2730 This Functional Element fulfills the following requirements from the Functional Elements
2731 Requirements, Working Draft 01a:

- 2732 • Primary Requirements
 - 2733 • MANAGEMENT-090, and
 - 2734 • MANAGEMENT-093 to MANAGEMENT-096.
- 2735 • Secondary Requirements
 - 2736 • None

2737 **2.12.2 Terms Used**

Terms	Description
Management Domain	Management Domain refers to the set of servers that needs to be monitored.
Performance Parameters	Performance Parameters refers to the set of attributes that should be track for the purpose of evaluating the performance of the Web Services.
Monitoring	Monitoring refers to the logging and tracking of the Web Service’s

2738

2739 **2.12.3 Key Features**

2740 Implementations of the Service Management Functional Element are expected to provide the
2741 following key features:

- 2742 1. The Functional Element MUST provide the capability to configure the Management Domain.
Example: All Servers that falls under a certain IP range (192.168.20.3 to 192.168.20.22)
- 2743 2. The Functional Element MUST provide the capability to discover services that are under the
2744 Management Domain.
- 2745 3. The Functional Element MUST provide the capability to configure Performance Parameters
2746 that are of interest for Monitoring purposes.

Example: The following are some of the Performance Parameter that may be of interest:

- *The time at which a Web Service request came.*
- *The time at which the corresponding response was sent.*
- *The name of the Web Service that was invoked.*

2747 4. The Functional Element MUST provide a means to log Performance Parameters.

2748

2749 In addition, the following key feature could be provided to enhance the Functional Element
2750 further:

2751 1. The Functional Element MAY provide the capability to configure additional attributes that is
2752 tagged along with a particular Web Service.

Example: The access permission for invoking the service.

2753 2. The Functional Element MAY provide verification services to block unauthorized Web
2754 Service's usage.

Example: The header information that accompanies the request may be extracted for relevant client's credential. This could then be compared to the access permission for the service.

2755 2.12.4 Interdependencies

Direct Dependency

Log Utility Functional Element

The Log Utility Functional Element helps to log the Performance Parameter into the appropriate data sources

2756

Interaction Dependencies

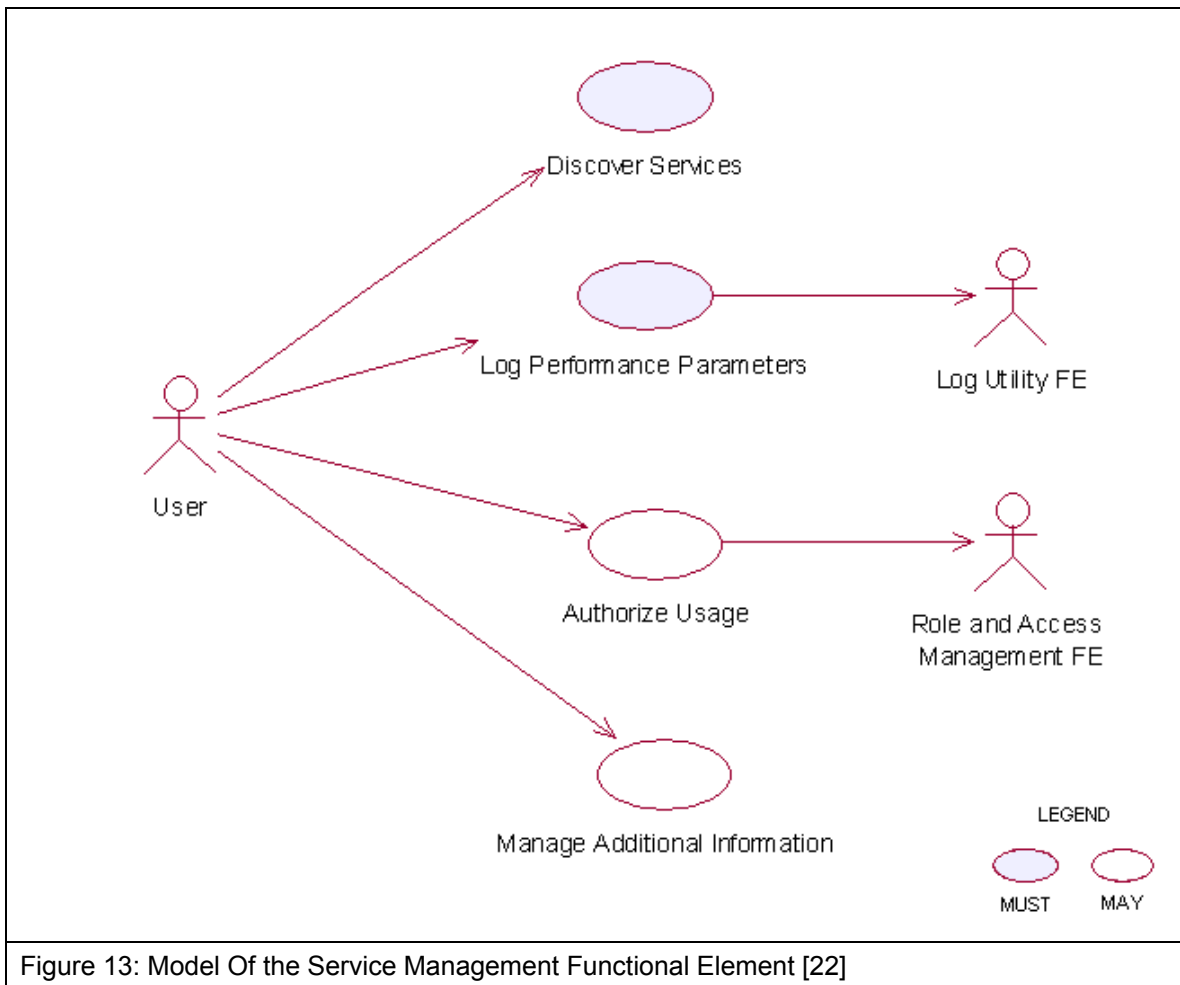
Role and Access Management Functional Element

In the event when authentication is required before invocation of a particular service is allowed, the Service Management Functional Element may extract authentication information from the header of the incoming request and use the Role and Access Management Functional Element to extract the relevant role information before deciding if a user has the privilege to access a particular Web Service.

2757 2.12.5 Related Technologies and Standards

2758 None

2.12.6 Model



2760 2.12.7 Usage Scenarios

2761 2.12.7.1 Discover Services

2762 2.12.7.1.1 Description

2763 This use case describes the scenario surrounding the automatic discovery of services hosted in
2764 the Management Domain.

2765 2.12.7.1.2 Flow of Events

2766 2.12.7.1.2.1 Basic Flow

2767 The use case begins when the user wants to retrieve a list of services URLs from the
2768 Management Domain.

2769 1: The user sends a request to retrieve the list of services URLs from the Management Domain.

2770 2: The Functional Element reads from a configuration file to so as to determine the exact
2771 boundaries of the Management Domain.

2772 3: The Functional Element retrieves from each of the servers as stated in the configuration file a
2773 list of service URLs that it is hosting

2774 4: The Functional Element returns the list of service URLs back to the user and the use case
2775 ends.

2776 **2.12.7.1.2.2 Alternative Flows**

2777 1: Configuration File Does Not Exist

2778 1.1: In basic flow 2, the Functional Element fails to read boundaries from the configuration
2779 file. The Functional Element in turn return an error message and the use case end.

2780 2: Fail To Communicate With the Server

2781 2.1: In basic flow 3, the Functional Element fails to communicate with the servers hosting the
2782 services. The Functional Element in turn return an error message and the use case end.

2783 **2.12.7.1.3 Special Requirements**

2784 The protocol of communicating with a server hosting the services is not standardized. Each
2785 server may offer different mechanism for retrieving the list of services hosted and as such, the
2786 extensibility this approach is severely limited.

2787 **2.12.7.1.4 Pre-Conditions**

2788 None.

2789 **2.12.7.1.5 Post-Conditions**

2790 None

2791 **2.12.7.2 Log Performance Parameters**

2792 **2.12.7.2.1 Description**

2793 This use case allows the user to log the performance parameters of all the Web Services that is
2794 being hosted by an application that contains the Service Management Functional Element.

2795 **2.12.7.2.2 Flow of Events**

2796 **2.12.7.2.2.1 Basic Flow**

2797 The use case begins when the user wants to log the performance parameters of all the Web
2798 Services that is being hosted by an application that contains the Service Management Functional
2799 Element.

2800 1: The user sends a request to log the performance parameters of all the Web Services hosted.

2801 2: The Functional Element reads from a configuration file the performance parameter to be
2802 logged.

2803 3: The Functional Element extracts the performance parameters from the incoming message and
2804 stores them into the data store

2805 4: The Functional Element next extracts the performance parameters from the outgoing message
2806 and stores them into the data store

2807 5: The Functional Element stores the necessary information into the data store.

2808 **2.12.7.2.2 Alternative Flows**

2809 1: No Performance Parameter Found.

2810 1.1: In basic flow 2, the Functional Element discovers that the performance parameter to be
2811 logged is not configured. The Functional Element returns an error message and the use case
2812 ends.

2813 2: Data Store Not Available.

2814 2.1: In basic flow 5, the Functional Element detects that the data store is not available. The
2815 Functional Element returns an error message and the use case ends.

2816 **2.12.7.2.3 Special Requirements**

2817 None.

2818 **2.12.7.2.4 Pre-Conditions**

2819 None.

2820 **2.12.7.2.5 Post-Conditions**

2821 None.

2822 **2.12.7.3 Authorize Usage**

2823 **2.12.7.3.1 Description**

2824 This use case describes the authentication process for invoking a Web Service that is being
2825 hosted by an application that contains the Service Management Functional Element.

2826 **2.12.7.3.2 Flow of Events**

2827 **2.12.7.3.2.1 Basic Flow**

2828 The use case starts when a user accesses a service.

2829 1: The user sends a request to invoke a particular Web Service.

2830 2: The Functional Element extracts the following information from the incoming message

2831 2.1: The username attribute that resides in the header of the incoming message

2832 3: The Functional Element extracts the access privilege associated with the service from the data
2833 store

2834 4: The Functional Element uses the Role and Access Management Functional Element to retrieve
2835 the role of the user.

2836 5: The Functional Element looks up the data store to determine if the user is authorized to access
2837 the service

2838 6: The Functional Element allows the request to be process and the use case ends.

2839 **2.12.7.3.2.2 Alternative Flow**

2840 1: Username header not found.

2841 1.1: In basic flow 2, the username attribute is not found in the header.

2842 1.2: The Functional Element denies access to the requested Web Service and returns an
2843 error message.

2844 2: Web Service access privilege not set.

2845 2.1: In basic flow 3, the Functional Element could not find the access privilege for the Web
2846 Service.

2847 2.2: The Functional Element denies access to the requested Web Service and returns an
2848 error message.

2849 3: Role and Access Management Functional Element not available

2850 3.1: In basic flow 4, the Functional Element could not find the Role and Access Management
2851 Functional Element.

2852 3.2: The Functional Element denies access to the requested Web Service and returns an
2853 error message.

2854 4: User not authorize

2855 4.1: In basic flow 5, the Functional Element looks up the data source and determines that the
2856 user does not have the required privilege to access the service.

2857 4.2: The Functional Element denies access to the requested Web Service and returns an
2858 error message.

2859 **2.12.7.3.3 Special Requirements**

2860 None.

2861 **2.12.7.3.4 Pre-Conditions**

2862 None.

2863 **2.12.7.3.5 Post-Conditions**

2864 None.

2865 **2.12.7.4 Manage Additional Information**

2866 **2.12.7.4.1 Description**

2867 This use case helps to maintain the following attributes of a Web Service that is useful in
2868 determining if a particular user has the privilege to invoke it.

- 2869
- Service Name. This is the name of the service to monitor
- 2870
- Access level. This refers to the access level of the Web Services hosted
- 2871
- Role Names. If a user's role matches any of the roles contained here, then he/she has
- 2872 the privilege to access the Web Service.

2873 **2.12.7.4.2 Flow of Events**

2874 **2.12.7.4.2.1 Basic Flow**

2875 This use case starts when user wants to manage services.

2876 1: The user specifies the additional information that he wants to create/update/delete/retrieve.

2877 2: Once the user provides the requested information, one of the sub-flows is executed.

- 2878 • If the user provides '**Create Service**', then sub-flow 2.1 is executed.
- 2879 • If the user provides '**Update Service**', then sub-flow 2.2 is executed.
- 2880 • If the user provides '**Delete Service**', then sub-flow 2.3 is executed.
- 2881 • If the user provides '**Retrieve Service**', then sub-flow 2.4 is executed.

2882 2.1: Create Service.

2883 2.1.1: The user specifies the service to create with the appropriate additional information.

2884 2.1.2: The Functional Element connects to the data store.

2885 2.1.3: The Functional Element saves the new service in the data store and the use case
2886 ends.

2887 2.2: Update Service.

2888 2.2.1: The user specifies the service to update with the appropriate additional information.

2889 2.2.2: The Functional Element connects to the data store.

2890 2.2.3: The Functional Element updates the service in the data store and the use case
2891 ends.

2892 2.3: Delete Service.

2893 2.3.1: The user specifies the service to delete.

2894 2.3.2: The Functional Element connects to the data store.

2895 2.3.3: The Functional Element deletes the service in the data store and the use case
2896 ends.

2897 2.4: Retrieve Service.

2898 2.4.1: The user specifies the service to retrieve.

2899 2.4.2: The Functional Element connects to the data store.

2900 2.4.3: The Functional Element retrieves the service from the data store and the use case
2901 ends.

2902 **2.12.7.4.2.2 Alternative Flows**

2903 1: Data Store Not Available.

2904 1.1: If in basic flow 2.1.2, 2.2.2, 2.3.2 and 2.4.2, the data store is not available, an error message
2905 is returned and the use case ends.

2906 **2.12.7.4.3 Special Requirements**

2907 None.

2908 **2.12.7.4.4 Pre-Conditions**

2909 None.

2910 **2.12.7.4.5 Post-Conditions**

2911 None.

2912 **2.13 Service Registry Functional Element**

2913 **2.13.1 Motivation**

2914 In a Web Service-enabled implementation, there exist the needs to maintain a central repository
2915 of all the services that are available. This facilitates service lookups as well as management of
2916 Web Services within the application that contains the Functional Element. In order to achieve
2917 these expectations, the Functional Element will cover the following aspects.

- 2918 • Simplify management of information in a XML registry server like UDDI and ebXML, and
- 2919 • Simplify information publish and query from a XML registry server like UDDI and ebXML.

2920

2921 This Functional Element fulfills the following requirements from the Functional Elements
2922 Requirements, Working Draft 01a:

- 2923 • Primary Requirements
 - 2924 • PROCESS-031 to PROCESS-032,
 - 2925 • PROCESS-035, and
 - 2926 • MANAGEMENT-097 to MANAGEMENT-100
- 2927 • Secondary Requirements
 - 2928 • PROCESS-014.

2929

2930 **2.13.2 Terms Used**

Terms	Description
Classification / Taxonomy	Classification / Taxonomy refers to a taxonomy that may be used to classify or categorize any registry object instances like Organizations, Web Services, Service Bindings, etc.
Concept / tModel	Concept / tModel is used to represent taxonomy elements and their structural relationship with each other in order to describe an internal taxonomy.
Organization	Organization provides information on organizations such as a Submitting Organization. Each Organization may have a reference to a parent Organization. In addition it may have a contact attribute defining the primary contact within the organization. An Organization also has an address attribute.
Registry Server	Registry Server refers to a registry that offers a mechanism for users or software applications to advertise and discover Web Services. An XML registry is an infrastructure that enables the building, deployment, and discovery of Web Services.
Service Binding	Service Binding represent technical information on a specific way to access a specific interface offered by a service.
UUID	Universally Unique Identifier

2931

2.13.3 Key Features

2932

Implementations of the Group Management Functional Element are expected to provide the following key features:

2933

2934

1. The Functional Element MUST provide the capability to facilitate the management of the following information in a UDDI or an ebXML compliant registry server.

2935

2936

1.1. Organisation

2937

1.2. Classification / Taxonomy

2938

1.3. Web Service

2939

1.4. tModel / Concept

2940

1.5. Service Binding

2941

The management of this information includes registering, updating, deleting and searching.

2942

2. As part of Key Feature (1), the Functional Element MUST provide the ability to perform the operations specified across multiple registry servers.

2943

2944

2945

In addition, the following key feature could be provided to enhance the Functional Element further:

2946

2947

1. The Functional Element MAY provide a mechanism enable single step publishing of services into registry servers.

2948

2949

2950

2.13.4 Interdependencies

2951

None

2952

2.13.5 Related Technologies and Standards

Specifications	Description
UDDI Data Structure and API Specification v2.0	UDDI Data Structure Specification v2.0 describes in detail the data structure models of organizations, web services, service categories, service bindings, and tModels. [23] UDDI API Specification v2.0 describes in detail the publishing, deleting, and querying API(s) to manipulate the information stored in XML registry server like UDDI. [24]
ebXML Registry Information Model (RIM) Specification v2.0 [25]	ebXML Registry Information Model Specification v2.0 describes in detail the data structure models of organizations, web services, service categories, service bindings, and tModels.
ebXML Registry Services (RS) Specification v2.0 [26]	ebXML Registry Services Specification v2.0 describes in detail the publishing, deleting, and querying API(s) to manipulate the information stored in XML registry server like UDDI.

2953

2954 **2.13.6 Model**

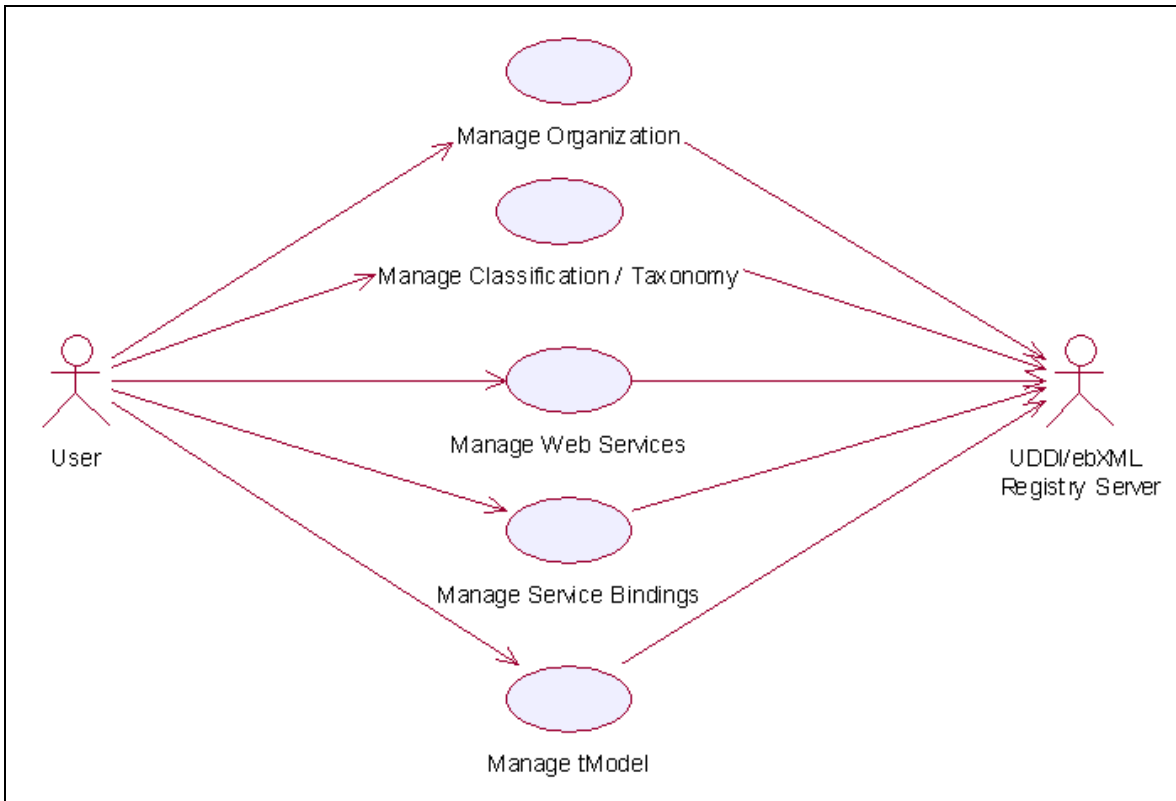


Figure 14: Model Of the Service Registry Functional Element [27]

2955 **2.13.7 Usage Scenario**

2956 **2.13.7.1 Manage Classification / Taxonomy**

2957 **2.13.7.1.1 Description**

2958 This use case allows any users to create, remove and view classification/taxonomy in the
2959 registry.

2960 **2.13.7.1.2 Flow of Events**

2961 **2.13.7.1.2.1 Basic Flow**

2962 This use case starts when the users of registry server wishes to create, remove or view the
2963 classification/taxonomy in the registry server.

2964

2965 1: User initiates a request type to the Functional Element stating whether to create, remove or
2966 view classification/taxonomy.

2967 2: The Functional Element checks whether the registry server exists.

2968 3: The Functional Element checks the request. Based on the type of request, one of the sub-
2969 flows is executed.

- 2970 • If the request is to '**Create Classification/Taxonomy**', then sub-flow 3.1 is executed.
 - 2971 • If the request is to '**View Classification/Taxonomy**', then sub-flow 3.2 is executed.
 - 2972 • If the request is to '**Remove Classification/Taxonomy**', then sub-flow 3.3 is executed.
- 2973 3.1: Create Classification/Taxonomy.
- 2974 3.1.1: Other Functional Element provides username, password and registry server URL
2975 to the Functional Element for authentication.
- 2976 3.1.2: The Functional Element checks for the user validity in the identified registry server.
- 2977 3.1.3: Other Functional Element provides classification/taxonomy information to be
2978 created in the registry server.
- 2979 3.1.4: The Functional Element checks for the duplicate classification/taxonomy name.
- 2980 3.1.5: The Functional Element creates the classification/taxonomy information in the
2981 private (default) or the public UDDI registry server according to the URL provided by
2982 other Functional Element, if it does not exist.
- 2983 3.2: View Classification/Taxonomy.
- 2984 3.2.1: The Functional Element retrieves all the classification/taxonomy from the identified
2985 registry server, which may be private (default) or public.
- 2986 3.2.2: The Functional Element returns the classification/taxonomy information from the
2987 identified registry server to other Functional Element.
- 2988 3.3: Remove Classification/Taxonomy.
- 2989 3.3.1: Other Functional Element provides username, password and registry server URL
2990 to the Functional Element for authentication.
- 2991 3.3.2: The Functional Element checks for the user validity in the identified registry server.
- 2992 3.3.3: Other Functional Element provides classification/taxonomy key (i.e. UUID) to be
2993 removed from the identified registry server.
- 2994 3.3.4: The Functional Element removes the classification/taxonomy information from the
2995 private (default) or the public UDDI registry server according to the URL provided by the
2996 user.
- 2997 4: The Functional Element returns the status of the operation and the use case ends.

2998 **2.13.7.1.2.2 Alternative Flows**

- 2999 1: Registry Server Down.
- 3000 1.1: In the basic flow 2, if the identified registry server is down, the Functional Element
3001 returns an error message and the use case ends.
- 3002 2: Invalid Username And Password.
- 3003 2.1: In the basic flow 3.1.2 and 3.3.2, if the username or password is invalid, the Functional
3004 Element returns an error message and the use case ends.
- 3005 3: Classification/Taxonomy Key Not Found.
- 3006 3.1: In the basic flow 3.3.3, if the classification/taxonomy key cannot be found in the
3007 specified registry server, the Functional Element returns an error message and the use
3008 case ends.

3009 4: Duplicate Classification/Taxonomy.
3010 4.1: In the basic flow 3.1.4, If the same classification/taxonomy name has been defined in
3011 the registry server, the Functional Element returns an error message and the use case
3012 ends.

3013 **2.13.7.1.3 Special Requirements**

3014 None

3015 **2.13.7.1.4 Pre-Conditions**

3016 In order to manage the classification/taxonomy in the registry server, users must be registered
3017 with the registry server. Username and password will be given when a user registers with a
3018 registry server.

3019 **2.13.7.1.5 Post-Conditions**

3020 None.

3021 **2.13.7.2 Manage Web Services**

3022 **2.13.7.2.1 Description**

3023 This use case allows any users to register, remove and view Web Services in the private (default)
3024 as well as the public UDDI Registry Server.

3025 **2.13.7.2.2 Flow of Events**

3026 **2.13.7.2.2.1 Basic Flow**

3027 This use case starts when the users of registry server wishes to create, remove and view Web
3028 Services.

3029 1: User initiates a request type to the Functional Element stating whether to create, remove or
3030 view Web Services in the identified private or public registry server.

3031 2: The Functional Element checks whether the registry server exists.

3032 3: The Functional Element checks the request. Based on the type of request, one of the sub-
3033 flows is executed.

- 3034 • If the request is to '**Create Web Service**', then sub-flow 3.1 is executed.
- 3035 • If the request is to '**View Web Services**', then sub-flow 3.2 is executed.
- 3036 • If the request is to '**Remove Web Service**', then sub-flow 3.3 is executed.

3037 3.1: Create Web Service.

3038 3.1.1: User provides username, password and registry server URL to the Functional
3039 Element for authentication.

3040 3.1.2: The Functional Element checks for the user validity in the identified registry server.

3041 3.1.3: Other Functional Element provides Web Service information to be created in the
3042 registry server.

3043 3.1.4: The Functional Element creates the Web Service information in the private
3044 (default) or the public UDDI registry server according to the URL provided by other
3045 Functional Element.

3046 3.2: View Web Services.

3047 3.2.1: The Functional Element retrieves all the Web Services from the identified registry
3048 server for specific stated conditions like service name search, business name search,
3049 etc.

3050 3.2.2: The Functional Element displays the Web Services information search results from
3051 the identified registry server to other Functional Element.

3052 3.3: Remove Web Service

3053 3.3.1 User provides username, password and registry server URL to the Functional
3054 Element for authentication.

3055 3.3.2: The Functional Element checks for the user validity in the identified registry server.

3056 3.3.3: Other Functional Element provides Web Service key (i.e. UUID) to be removed
3057 from the identified registry server.

3058 3.3.4: The Functional Element removes the Web Service information from the private
3059 (default) or the public UDDI registry server according to the URL provided by other
3060 Functional Element.

3061 4: The Functional Element returns the results of the operation and the use case ends.

3062 **2.13.7.2.2.2 Alternative Flows**

3063 1: Registry Server Down.

3064 1.1: In the basic flow 2, if the identified registry server is down, the Functional Element
3065 returns an error message and the use case ends.

3066 2: Invalid Username And Password.

3067 2.1: In the basic flow 3.1.2 and 3.3.2, if the username or password is invalid, the Functional
3068 Element returns an error message and the use case ends.

3069 3: Web Service Key Not Found.

3070 3.1: In the basic flow 3.3.3, if the Web Service key cannot be found in the specified registry
3071 server, the Functional Element returns an error message and the use case ends.

3072 **2.13.7.2.3 Special Requirements**

3073 **2.13.7.2.4 Pre-Conditions**

3074 In order to manage Web Services in the registry server, the users must be registered with the
3075 registry server. Username and password will be given when a user registers with a registry
3076 server.

3077 **2.13.7.2.5 Post-Conditions**

3078 None.

3079 **2.13.7.3 Manage Organization**

3080 **2.13.7.3.1 Description**

3081 This use case allows any users to create, remove and view organization in the registry.

3082 **2.13.7.3.2 Flow of Events**

3083 **2.13.7.3.2.1 Basic Flow**

3084 This use case starts when the users of registry server wishes to create, remove or view
3085 Organization.

3086 1: User initiates a request type to the Functional Element stating whether to create, remove or
3087 view Organization.

3088 2: The Functional Element checks whether the registry server exists.

3089 3: The Functional Element checks the request. Based on the type of request, one of the sub-
3090 flows is executed.

- 3091 • If the request is to '**Create Organization**', then sub-flow 3.1 is executed.
- 3092 • If the request is to '**View Organizations**', then sub-flow 3.2 is executed.
- 3093 • If the request is to '**Remove Organization**', then sub-flow 3.3 is executed.

3094 3.1: Create Organization.

3095 3.1.1: Other Functional Element provides username, password and registry server URL
3096 to the Functional Element for authentication.

3097 3.1.2: The Functional Element checks for the user validity in the identified registry server.

3098 3.1.3: Other Functional Element provides organization information to be created in the
3099 registry server.

3100 3.1.4: The Functional Element checks for the duplicate organization name.

3101 3.1.5: The Functional Element creates the organization information in the private (default)
3102 or the public UDDI registry server according to the URL provided by other Functional
3103 Element, if it does not exist.

3104 3.2: View Organizations.

3105 3.2.1: The Functional Element retrieves all the organizations from the identified registry
3106 server for specific stated conditions like organization name, key, etc.

3107 3.2.2: The Functional Element returns the organization information from the identified
3108 registry server to other Functional Element.

3109 3.3: Remove Organization.

3110 3.3.1: Other Functional Element provides username, password and registry server URL
3111 to the Functional Element for authentication.

3112 3.3.2: The Functional Element checks for the user validity in the identified registry server.

3113 3.3.3: Other Functional Element provides Organization key (i.e. UUID) to be removed
3114 from the identified registry server.

3115 3.3.4: The Functional Element removes the Organization information from the private
3116 (default) or the public UDDI registry server according to the URL provided by the user.

3117 4: The Functional Element returns the status of the operation and the use case ends.

3118 **2.13.7.3.2.2 Alternative Flows**

3119 1: Registry Server Down.

3120 1.1: In the basic flow 2, if the identified registry server is down, the Functional Element
3121 returns an error message and the use case ends.

3122 2: Invalid Username And Password.

3123 2.1: In the basic flow 3.1.2 and 3.3.2, if the username or password is invalid, the Functional
3124 Element returns an error message and the use case ends.

3125 3: Organization Key Not Found.

3126 3.1: In the basic flow 3.3.3, if the Organization key cannot be found in the specified registry
3127 server, the Functional Element returns an error message and the use case ends.

3128 4: Duplicate Organization.

3129 4.1: In the basic flow 3.1.4, If the same Organization name has been defined in the registry
3130 server the Functional Element returns an error message and the use case ends.

3131 **2.13.7.3.3 Special Requirements**

3132 None

3133 **2.13.7.3.4 Pre-Conditions**

3134 In order to manage Organization in the registry server, users must be registered with the registry
3135 server. Username and password will be given when a user registers with a registry server.

3136 **2.13.7.3.5 Post-Conditions**

3137 None.

3138 **2.13.7.4 Manage Service Binding**

3139 **2.13.7.4.1 Description**

3140 This use case allows any users to register, remove and view Service Binding in the private
3141 (default) as well as the public UDDI Registry Server.

3142 **2.13.7.4.2 Flow of Events**

3143 **2.13.7.4.2.1 Basic Flow**

3144 This use case starts when the users of registry server wishes to create, remove and view Service
3145 Binding.

3146 1: User initiates a request type to the Functional Element stating whether to create, remove or
3147 view Service Binding in the identified private or public registry server.

3148 2: The Functional Element checks whether the registry server exists.

3149 3: The Functional Element checks the request. Based on the type of request, one of the sub-
3150 flows is executed.

- 3151 • If the request is to '**Create Service Binding**', then sub-flow 3.1 is executed.
 - 3152 • If the request is to '**View Service Bindings**', then sub-flow 3.2 is executed.
 - 3153 • If the request is to '**Remove Service Binding**', then sub-flow 3.3 is executed.
- 3154 3.1: Create Service Binding.
- 3155 3.1.1: User provides username, password and registry server URL to the Functional
3156 Element for authentication.
- 3157 3.1.2: The Functional Element checks for the user validity in the identified registry server.
- 3158 3.1.3: Other Functional Element provides Service Binding information to be created in the
3159 registry server.
- 3160 3.1.4: The Functional Element creates the Service Binding information in the private
3161 (default) or the public UDDI registry server according to the URL provided by other
3162 Functional Element.
- 3163 3.2: View Service Bindings.
- 3164 3.2.1: The Functional Element retrieves all the Service Bindings from the identified
3165 registry server for specific stated conditions like service binding key search, etc.
- 3166 3.2.2: The Functional Element displays the Service Bindings information search results
3167 from the identified registry server to other Functional Element.
- 3168 3.3: Remove Service Binding
- 3169 3.3.1 User provides username, password and registry server URL to the Functional
3170 Element for authentication.
- 3171 3.3.2: The Functional Element checks for the user validity in the identified registry server.
- 3172 3.3.3: Other Functional Element provides Service Binding key (i.e. UUID) to be removed
3173 from the identified registry server.
- 3174 3.3.4: The Functional Element removes the Service Binding information from the private
3175 (default) or the public UDDI registry server according to the URL provided by other
3176 Functional Element.
- 3177 4: The Functional Element returns the results of the operation and the use case ends.

3178 **2.13.7.4.2.2 Alternative Flows**

- 3179 1: Registry Server Down.
- 3180 1.1: In the basic flow 2, if the identified registry server is down, the Functional Element returns
3181 an error message and the use case ends.
- 3182 2: Invalid Username And Password.
- 3183 2.1: In the basic flow 3.1.2 and 3.3.2, if the username or password is invalid, the Functional
3184 Element returns an error message and the use case ends.
- 3185 3: Service Binding Key Not Found.
- 3186 3.1: In the basic flow 3.3.3, if the Service Binding key cannot be found in the specified registry
3187 server, the Functional Element returns an error message and the use case ends.

3188 **2.13.7.4.3 Special Requirements**

3189 **2.13.7.4.4 Pre-Conditions**

3190 In order to manage Service Binding in the registry server, the users must be registered with the
3191 registry server. Username and password will be given when a user registers with a registry
3192 server.

3193 **2.13.7.4.5 Post-Conditions**

3194 None.

3195 **2.13.7.5 Manage tModel**

3196 **2.13.7.5.1 Description**

3197 This use case allows any users to register, remove and view tModel in the private (default) as
3198 well as the public UDDI Registry Server.

3199 **2.13.7.5.2 Flow of Events**

3200 **2.13.7.5.2.1 Basic Flow**

3201 This use case starts when the users of registry server wishes to create, remove and view tModel.

3202 1: User initiates a request type to the Functional Element stating whether to create, remove or
3203 view tModel in the identified private or public registry server.

3204 2: The Functional Element checks whether the registry server exists.

3205 3: The Functional Element checks the request. Based on the type of request, one of the sub-
3206 flows is executed.

3207 • If the request is to '**Create tModel**', then sub-flow 3.1 is executed.

3208 • If the request is to '**View tModels**', then sub-flow 3.2 is executed.

3209 • If the request is to '**Remove tModel**', then sub-flow 3.3 is executed.

3210 3.1: Create tModel.

3211 3.1.1: User provides username, password and registry server URL to the Functional
3212 Element for authentication.

3213 3.1.2: The Functional Element checks for the user validity in the identified registry server.

3214 3.1.3: Other Functional Element provides tModel information to be created in the registry
3215 server.

3216 3.1.4: The Functional Element creates the tModel information in the private (default) or
3217 the public UDDI registry server according to the URL provided by other Functional
3218 Element.

3219 3.2: View tModels.

3220 3.2.1: The Functional Element retrieves all the tModels from the identified registry server
3221 for specific stated conditions like tModel name search, tModel key search, etc.

- 3222 3.2.2: The Functional Element displays the tModel information search results from the
3223 identified registry server to other Functional Element.
- 3224 3.3: Remove tModel.
- 3225 3.3.1 User provides username, password and registry server URL to the Functional
3226 Element for authentication.
- 3227 3.3.2: The Functional Element checks for the user validity in the identified registry server.
- 3228 3.3.3: Other Functional Element provides tModel key (i.e. UUID) to be removed from the
3229 identified registry server.
- 3230 3.3.4: The Functional Element removes the tModel information from the private (default)
3231 or the public UDDI registry server according to the URL provided by other Functional
3232 Element.
- 3233 4: The Functional Element returns the results of the operation and the use case ends.
- 3234 **2.13.7.5.2.2 Alternative Flows**
- 3235 1: Registry Server Down.
- 3236 1.1: In the basic flow 2, if the identified registry server is down, the Functional Element returns
3237 an error message and the use case ends.
- 3238 2: Invalid Username And Password.
- 3239 2.1: In the basic flow 3.1.2 and 3.3.2, if the username or password is invalid, the Functional
3240 Element returns an error message and the use case ends.
- 3241 3: tModel Key Not Found.
- 3242 3.1: In the basic flow 3.3.3, if the tModel key cannot be found in the specified registry server,
3243 the Functional Element returns an error message and the use case ends.
- 3244 **2.13.7.5.3 Special Requirements**
- 3245 **2.13.7.5.4 Pre-Conditions**
- 3246 In order to manage tModel in the registry server, the users must be registered with the registry
3247 server. Username and password will be given when a user registers with a registry server.
- 3248 **2.13.7.5.5 Post-Conditions**
- 3249 None.

3250 **2.14 Service Tester Functional Element**

3251 **2.14.1 Motivation**

3252 In a Web Service environment where the lifecycle of services may be rather dynamic, there exist
3253 a need for a client to dynamically discover the capabilities of the hosted service and bind to these
3254 services dynamically. The later is the main motivation of this Functional Element.

3255

3256 This Functional Element fulfills the following requirements from the Functional Elements
3257 Requirements, Working Draft 01a:

- 3258
 - Primary Requirements
 - 3259
 - MANAGEMENT-091,
 - 3260
 - MANAGEMENT-094, and
 - 3261
 - PROCESS-130 to PROCESS-132.
 - 3262 • Secondary Requirements
 - 3263
 - PROCESS-133.

3264 **2.14.2 Terms Used**

Terms	Description
WSDL	Web Services Description Language

3265 **2.14.3 Key Features**

3266 Implementations of the Service Tester Functional Element are expected to provide the following
3267 key features:

- 3268 1. The Functional Element MUST provide the capability to generate a Web Service client from
3269 a WSDL file.
- 3270 2. The Functional Element MUST provide the capability to test the availability of Web Services
3271 based on the generated Web Service client.

3272 *Example: To retrieve the response time of a particular user-specified Web Service*
3273 *operation to test the availability of a Web Service.*

3274 **2.14.4 Interdependencies**

3275 None

3276 **2.14.5 Related Technologies and Standards**

Specifications	Description
WSDL 1.1 [28]	The ability to parse the WSDL document and generate a client is heavily dependent on it being a conforming WSDL document.

3277 **2.14.6 Model**

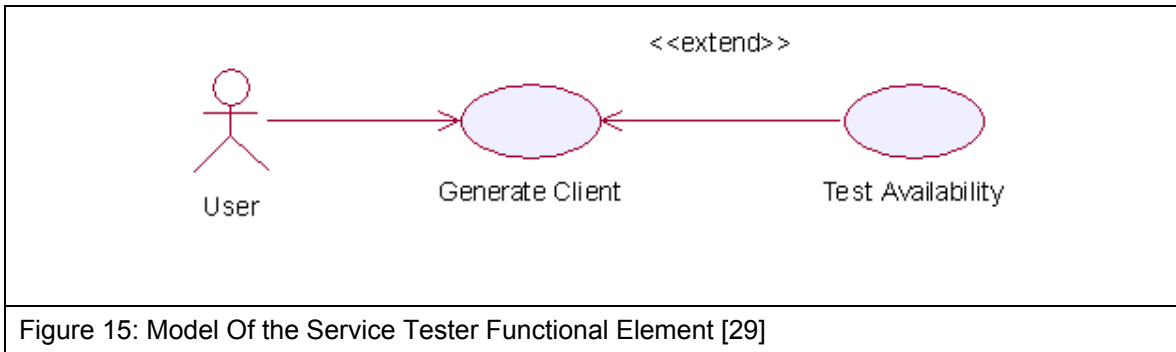


Figure 15: Model Of the Service Tester Functional Element [29]

3278

3279 **2.14.7 Usage Scenarios**

3280 **2.14.7.1 Generate Client**

3281 **2.14.7.1.1 Description**

3282 This use case describes the steps to generate a dynamic client

3283 **2.14.7.1.2 Flow of Events**

3284 **2.14.7.1.2.1 Basic Flow**

3285 This use case starts when the user wants to create a dynamic client

3286 1: User submits the WSDL of the Web Service.

3287 2: Functional Element parses the WSDL document and extracts the necessary information.

3288 3: Functional Element generates the client base on the available parameters.

3289 **2.14.7.1.2.2 Alternative Flows**

3290 1: Invalid WSDL

3291 11: In basic flow 2, if the structure of the WSDL does not comply with the standard, the
3292 Functional Element returns an error message and the use case ends.

3293 **2.14.7.1.3 Special Requirements**

3294 None.

3295 **2.14.7.1.4 Pre-Conditions**

3296 None.

3297 **2.14.7.1.5 Post-Conditions**

3298 None.

3299 **2.14.7.2 Test Availability**

3300 **2.14.7.2.1 Description**

3301 This use case allows the user to test the availability of a Web Service.

3302 **2.14.7.2.2 Flow of Events**

3303 **2.14.7.2.2.1 Basic Flow**

3304 This use case starts when a user wants to test the availability of a Web Service.

3305 1: User forms the dynamic client as describe in the use case 'Generate Client'.

3306 2: User inputs the acceptable response time for the purpose of testing the service.

3307 3: Functional Element invokes the web service and waits for the response. The response time is
3308 then compared with the stipulated time and the result is subsequently returned to the user.

3309 **2.14.7.2.2.2 Alternative Flows**

3310 1: Failure to Generate the Client

3311 1.1: In basic flow 1, if the Functional Element fails to generate the client, the Functional
3312 Element returns an error message and the use case ends.

3313 2: Time Out

3314 2.1: In basic flow 3, if the response if not returned within the stipulated time, the Functional
3315 Element returns an error message and the use case ends.

3316

3317 **2.14.7.2.3 Special Requirements**

3318 None.

3319 **2.14.7.2.4 Pre-Conditions**

3320 None.

3321 **2.14.7.2.5 Post-Conditions**

3322 None.

3323 2.15 User Management Functional Element

3324 2.15.1 Motivation

3325 The User Management Functional Element is expected to be an integral part of the user access
3326 management (UAM) functionalities that is expected to be needed by a Web Service-enabled
3327 implementation. This FE is expected to fulfill the needs arising out of managing resources within
3328 an application, with a user-centric viewpoint. As such it will cover aspects that include:

- 3329 • Basic user accounts management facilities,
- 3330 • Ability to extend dynamically from the set basic set of account information,
- 3331 • Capability for configurable policies governing account management,
- 3332 • Providing log trails for user activities, and
- 3333 • Management of user authentication means, either directly or indirectly.

3334

3335 This Functional Element fulfills the following requirements from the Functional Elements
3336 Requirements, Working Draft 01a:

- 3337 • Primary Requirements
 - 3338 • MANAGEMENT-001 to MANAGEMENT-003,
 - 3339 • MANAGEMENT-005,
 - 3340 • MANAGEMENT-008,
 - 3341 • MANAGEMENT-012, and
 - 3342 • SECURITY-002 (all).
- 3343 • Secondary Requirements
 - 3344 • SECURITY-001.

3345

3346 2.15.2 Terms Used

Terms	Description
Namespace	Namespace is use to segregate the instantiation of the application across different application domains. If a company has two separate standalone application, for example, an email application and an equipment booking application, then these two are considered as separate application domains.
User	A user is loosely defined to include both human and virtual users. Virtual users could include service users and application (or machine) users that are utilising other services in a SOA environment.

User Access Management / UAM	<p>User Access Management or UAM refer to the concept of managing users in a holistic manner, considering all aspect which includes:</p> <ul style="list-style-type: none"> • Defining a set of basic user information that should be stored in any enterprise application. • Providing a means to extend this basic set of user information when needed. • Simplifying management by grouping related users together through certain criteria. • Having the flexibility of adopting both coarse/fine grain access controls.
User Repository	User Repository is where the user information is stored. It can be a database or a flat file.

3347

3348 **2.15.3 Key Features**

3349 Implementations of the User Management Functional Element are expected to provide the
3350 following key features:

- 3351 1. The Functional Element MUST provide a User Repository.
- 3352 2. The Functional Element MUST be able to control access to such a User Repository.
- 3353 3. The Functional Element MUST provide a basic User structure with a set of pre-defined
3354 attributes.
- 3355 4. The Functional Element MUST provide the capability to extend on the basic User structure
3356 dynamically.
- 3357 5. As part of Key Feature (4), this dynamic extension MUST be definable and configurable at
3358 runtime implementation of the Functional Element.
- 3359 6. The Functional Element MUST provide the capability to manage the creation and deletion of
3360 instances of Users based on defined structure.
- 3361 7. The Functional Element MUST provide the capability to manage all the information (attribute
3362 values) stored in such Users. This includes the capability to:
 - 3363 7.1. Retrieve and update attribute's values belonging to a User,
 - 3364 7.2. Generate a random password,
 - 3365 7.3. Encrypt sensitive user information, and
 - 3366 7.4. Authenticate a user.
- 3367 8. As part of Key Feature (7.4), the authentication of a User MUST be achieved at least
3368 through the use of a password.
- 3369 9. The Functional Element MUST provide a mechanism for managing Users across different
3370 application domains.

3371 *Example: Namespace control mechanism*

3372

3373 In addition, the following key features could be provided to enhance the Functional Element
3374 further:

- 3375 1. The Functional Element MAY provide a mechanism to control the username format.
3376 *Example: Usernames must be at least 8 characters long.*
- 3377 2. The Functional Element MAY provide additional security mechanisms to enhance the
3378 security of sensitive information like user passwords.

- 3379 *Example: Passwords are stored in security tokens, or a more secure encryption algorithms*
 3380 *for passwords.*
- 3381 3. If Key Feature (2) is provided, the Functional Element MAY also provide a selection of
 3382 selectable encryption algorithms.
- 3383 4. The Functional Element MAY provide additional security policies to ensure that systems are
 3384 not compromised.
- 3385 *Example: Passwords must be changed every 30 days.*
- 3386 5. If Key Feature (4) is provided, the Functional Element MAY also provide a facility to notify
 3387 users before the password expires.
- 3388

3389 **2.15.4 Interdependencies**

Interaction Dependencies	
Group Management Functional Element	The Group Management Functional Element may be used to provide useful aggregation of the users.
Phase and Lifecycle Management Functional Element	The Phase and Lifecycle Management Functional Element may be used to maintain the relationships between various phases of a project lifecycle and the group who is working on it.
Role and Access Management Functional Element	The Role and Access Management Functional Element may be used to manage the user's access rights by virtue of it's association with a group, phase or even the complete lifecycle of the project.

3390

3391 **2.15.5 Related Technologies and Standards**

3392 None

3393 **2.15.6 Model**

3394

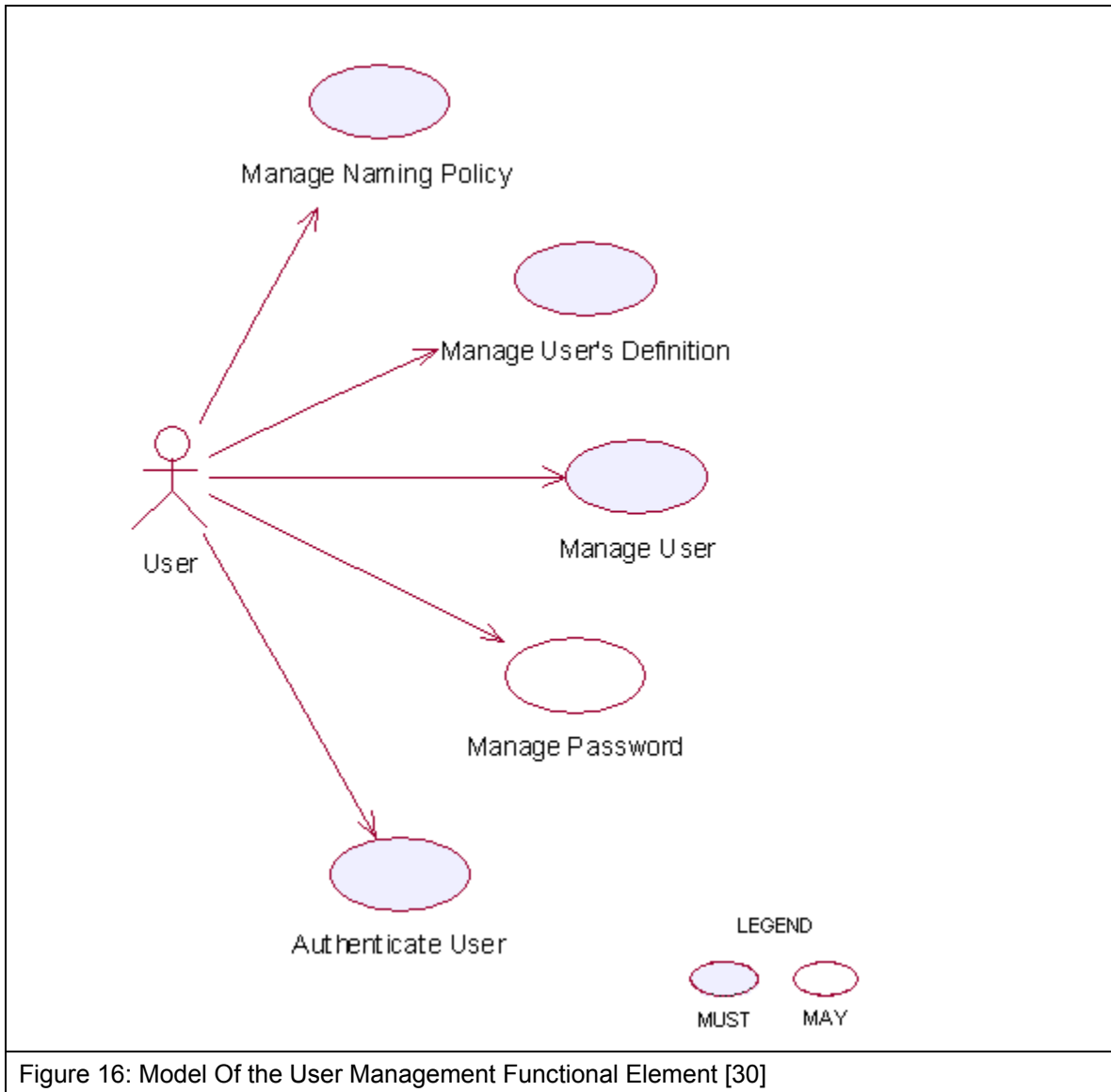


Figure 16: Model Of the User Management Functional Element [30]

3395 **2.15.7 Usage Scenarios**

3396 **2.15.7.1 Manage Naming Policy**

3397 **2.15.7.1.1 Description**

3398 This use case allows any user to manage naming policy when creating/updating user accounts.
 3399 The service user may create, update, retrieve and delete a naming policy.

3400 **2.15.7.1.2 Flow of Events**

3401 **2.15.7.1.2.1 Basic Flow**

3402 This use case starts when any user wants to manage naming policy for creating/updating user
 3403 account.

- 3404 1: The user sends Manage Naming Policy request to the Functional Element together with the
3405 specified operation.
- 3406 2: Functional Element gets the operation. Based on the operation, one of the sub-flows is
3407 executed.
- 3408 • If the service user provides '**Create Naming Policy**', then sub-flow 2.1 is executed.
 - 3409 • If the service user provides '**Update Naming Policy**', then sub-flow 2.2 is executed.
 - 3410 • If the service user provides '**Delete Naming Policy**', then sub-flow 2.3 is executed.
- 3411 2.1: Create Naming Policy.
- 3412 2.1.1: The service user specifies namespace, name and description of the policy to
3413 create, for example, the policy name may be name length, the policy description may be
3414 "=7".
- 3415 2.1.2: The Functional Element checks the existing naming policy.
- 3416 2.1.3: The Functional Element generates naming policy information and adds to the
3417 Functional Element and the use case ends.
- 3418 2.2: Update Naming Policy.
- 3419 2.2.1: The service user specifies the policy to update.
- 3420 2.2.2: The Functional Element retrieves the existing naming policy information.
- 3421 2.2.3: The service user provides the update naming policy information according to the
3422 policy name used in creating a naming policy.
- 3423 2.2.4: The Functional Element updates the naming policy with the updated information
3424 and ends use case.
- 3425 2.3: Retrieve Naming Policy.
- 3426 2.3.1: The service user specifies the policy to retrieve.
- 3427 2.3.2: The Functional Element retrieves the existing naming policy information and ends
3428 the use case.
- 3429 2.4: Delete Naming Policy.
- 3430 2.4.1: The service user specifies the policy to delete.
- 3431 2.4.2: The Functional Element retrieves the existing naming policy information.
- 3432 2.4.3: The Functional Element deletes the naming policy from the Functional Element
3433 and the use case ends.

3434 **2.15.7.1.2.2 Alternative Flows**

- 3435 1: Invalid Policy.
- 3436 1.1: If in the basic flow 2.1.1, Functional Element detects any invalid description, Functional
3437 Element returns general error message and ends the use case.
- 3438 2: Naming Policy already exists.

3439 2.1: If in the basic flow 2.1.2, the Functional Element checks the existing naming policy and
3440 finds the naming policy already exists. The Functional Element returns an error and ends the
3441 use case.

3442 **2.15.7.1.3 Special Requirements**

3443 **2.15.7.1.4 Pre-Conditions**

3444 None.

3445 **2.15.7.1.5 Post-Conditions**

3446 If the use case was successful, the naming policy information is added to the Functional Element.
3447 To do any creating and updating of User information after the naming policy is added must satisfy
3448 the naming policies defined. If unsuccessful, the Functional Element's state is unchanged.

3449 **2.15.7.2 Manage User Definition**

3450 **2.15.7.2.1 Description**

3451 The use case allows any user to manage user definition when more basic user definition can not
3452 satisfied in creating/updating user accounts. The service user may create, update, retrieve and
3453 delete a user definition.

3454 **2.15.7.2.2 Flow of Events**

3455 **2.15.7.2.2.1 Basic Flow**

3456 This use case starts when any user wants to manage user definition for creating/updating user
3457 account.

3458 1: The user sends Manage User Definition request to the Functional Element together with the
3459 specified operation.

3460 2: Functional Element gets the operation. Based on the operation, one of the sub-flows is
3461 executed.

- 3462 • If the service user provides '**Create User Definition**', then sub-flow 2.1 is executed.
- 3463 • If the service user provides '**Update User Definition**', then sub-flow 2.2 is executed.
- 3464 • If the service user provides '**Delete User Definition**', then sub-flow 2.3 is executed.

3465 2.1: Create User Definition.

3466 2.1.1: The service user specifies namespace, name and description of the user definition
3467 fields to create.

3468 2.1.2: The Functional Element checks the existing user definition fields (including basic
3469 ones).

3470 2.1.3: The Functional Element generates user definition information and adds to the
3471 Functional Element and the use case ends.

3472 2.2: Update User Definition.

3473 2.2.1: The service user specifies the user definition field to update.

- 3474 2.2.2: The Functional Element retrieves the existing user definition information.
- 3475 2.2.3: The service user provides the update user definition information.
- 3476 2.2.4: The Functional Element updates the user definition with the updated information
3477 and ends use case.
- 3478 2.3: Retrieve User Definition.
- 3479 2.3.1: The service user specifies the user definition to retrieve.
- 3480 2.3.2: The Functional Element retrieves the existing user definition information and ends
3481 the use case.
- 3482 2.4: Delete User Definition.
- 3483 2.4.1: The service user specifies the user definition to delete.
- 3484 2.4.2: The Functional Element retrieves the existing user definition information.
- 3485 2.4.3: The Functional Element deletes the user definition from the Functional Element
3486 and the use case ends.
- 3487 **2.15.7.2.3 Alternative Flows**
- 3488 1: Invalid User Definition.
- 3489 1.1: If in basic flow 2.1.1, Functional Element detects any invalid description, Functional
3490 Element returns general error message and ends the use case.
- 3491 2: User Definition already exists.
- 3492 2.1: If in basic flow 2.1.2, the Functional Element checks the existing user definition and finds
3493 the user definition already exists. The Functional Element returns an error and ends the use
3494 case.
- 3495 3: User Definition not exists.
- 3496 3.1: If in basic flow 2.2.2, 2.3.2 and 2.4.2, the Functional Element checks the existing user
3497 definition and finds the user definition does not exist. The Functional Element returns an
3498 error and ends the use case.
- 3499 **2.15.7.2.4 Special Requirements**
- 3500 None
- 3501 **2.15.7.2.5 Pre-Conditions**
- 3502 None.
- 3503 **2.15.7.2.6 Post-Conditions**
- 3504 If the use case was successful, the user definition information is added to the Functional Element.
3505 Thereafter, when creating and updating User, the User information must satisfy the user definition
3506 defined earlier. If the use case fails, the Functional Element's state is unchanged.

3507 **2.15.7.3 Manage User**

3508 This use case describes the management of a user, namely the creation, deletion, retrieval and
3509 update of the user.

3510 **2.15.7.3.1 Flow of Events**

3511 **2.15.7.3.1.1 Basic Flow**

3512 This use case starts when the user wants to manage a user.

- 3513 • If user wants to '**Create User**', then basic flow 1 is executed.
- 3514 • If user wants to '**Retrieve User**', then basic flow 2 is executed.
- 3515 • If user wants to '**Update User**', then basic flow 3 is executed.
- 3516 • If user wants to '**Delete User**', then basic flow 4 is executed.

3517 1: Create User.

3518 1.1: User provides the information that is necessary for creating a user.

3519 1.2: The Functional Element validates the user information provided against the naming
3520 policy.

3521 1.3: The Functional Element validates the user information provided against the user's
3522 definition.

3523 1.4: Functional Element creates the user and the use case ends.

3524 2: Retrieve User.

3525 2.1: User provides the necessary information for retrieving the complete user's attributes.

3526 2.2: The Functional Element returns the user's information and the use case ends.

3527 3: Update User.

3528 3.1: User provides the necessary information for updating the group's attributes.

3529 3.2: The Functional Element validates the user's information provided against the naming
3530 policy.

3531 3.3: The Functional Element validates the user information provided against the user's
3532 definition.

3533 3.4: The Functional Element updates the user and the use case ends.

3534 4: Delete User.

3535 4.1: User provides the necessary information for deleting a user group.

3536 4.2: Functional Element deletes the user and the use case ends.

3537 **2.15.7.3.1.2 Alternative Flows**

3538 1: User Exist.

3539 1.1: In basic flow 1.4, if the Functional Element detects an identical user, the Functional
3540 Element returns an error message and the use case ends.

3541 2: User Does Not Exist.

3542 1.1: In basic flow 2.2, 3.4 and 4.2, if the Functional Element cannot find a user that matches
3543 the user's criteria, the Functional Element returns an error message and the use case ends.

3544 **2.15.7.3.2 Special Requirements**

3545 None.

3546 **2.15.7.3.3 Pre-Conditions**

3547 None.

3548 **2.15.7.3.4 Post-Conditions**

3549 None.

3550 **2.15.7.4 Authenticate User**

3551 **2.15.7.4.1 Description**

3552 This use case allows users to authenticate a user.

3553 **2.15.7.4.2 Flow of Events**

3554 **2.15.7.4.2.1 Basic Flow**

3555 This use case starts when users wish to authenticate a user.

3556 1: Users provide user name and password to Functional Element.

3557 2: The Functional Element checks the user name and password.

3558 3: The Functional Element returns the result to users and the use case ends.

3559 **2.15.7.4.2.2 Alternative Flows**

3560 None.

3561 **2.15.7.4.3 Special Requirements**

3562 None.

3563 **2.15.7.4.4 Pre-Conditions**

3564 None.

3565 **2.15.7.4.5 Post-Conditions**

3566 None.

3567 **2.15.7.5 Manage Password**

3568 This use case describes the management of password in this Functional Element.

3569 **2.15.7.5.1 Flow of Events**

3570 **2.15.7.5.1.1 Basic Flow**

3571 This use case starts when the user wants to obtain an encrypted password. This can be
3572 achieved via one of the following basic flow.

- 3573 • If user wants to '**Generate Password**', then basic flow 1 is executed.
3574 • If user wants to '**Encrypt Password**', then basic flow 2 is executed.

3575 1: Generate Password

3576 1.1: The user specifies the option of format of password among available options in the
3577 Functional Element.

3578 1.2: The Functional Element generates clear text password based on the format specified by
3579 the service user.

3580 1.3: The Functional Element includes "Encrypt Password" use case to encrypt the clear text
3581 password.

3582 1.4: The Functional Element returns the clear text password and encrypted password to user
3583 and the use case ends.

3584 2: Encrypt Password

3585 1.1: The user provides clear text password to Functional Element.

3586 1.2: The user specifies the encryption algorithm to be used.

3587 1.3: The Functional Element encrypts the clear text password.

3588 1.4: The Functional Element returns the encrypted password to user and the use case ends.

3589 **2.15.7.5.1.2 Alternative Flows**

3590 None.

3591 **2.15.7.5.2 Special Requirements**

3592 None.

3593 **2.15.7.5.3 Pre-Conditions**

3594 None.

3595 **2.15.7.5.4 Post-Conditions**

3596 None.

3597 **2.16 Web Service Aggregator Functional Element**

3598 **2.16.1 Motivation**

3599 In any Web Service-enabled application, it is expected that complex business functions have to
3600 be realized via aggregation of multiple Web Services. This Functional Element is expected to
3601 fulfill the needs arising out of Web Services composition. As such it will cover aspects that
3602 include:

- 3603 • Facilitating the composition of Web Services, and
3604 • Testing of aggregated Web Services.

3605

3606 This Functional Element fulfills the following requirements from the Functional Elements
3607 Requirements, Working Draft 01a:

- 3608 • Primary Requirements
3609 • PROCESS-010 to PROCESS-014.
3610 • Secondary Requirements
3611 • PROCESS-131

3612

3613 **2.16.2 Terms Used**

Terms	Description
Aggregated Web Service	Aggregated Web Service is single Web Services that invoke multiple Web Services to realize its functionality.
Composition Rule	A Composition Rule is an expression specifying how individual Web Services are invoked to form aggregated Web Services. It includes the name of Web Services that are included in aggregation, specification of aggregation sequence, data dependency among the individual Web Services.

3614

3615 The following diagram shows the meaning of the terms in the context of Web Services
3616 aggregation.

3617

3618

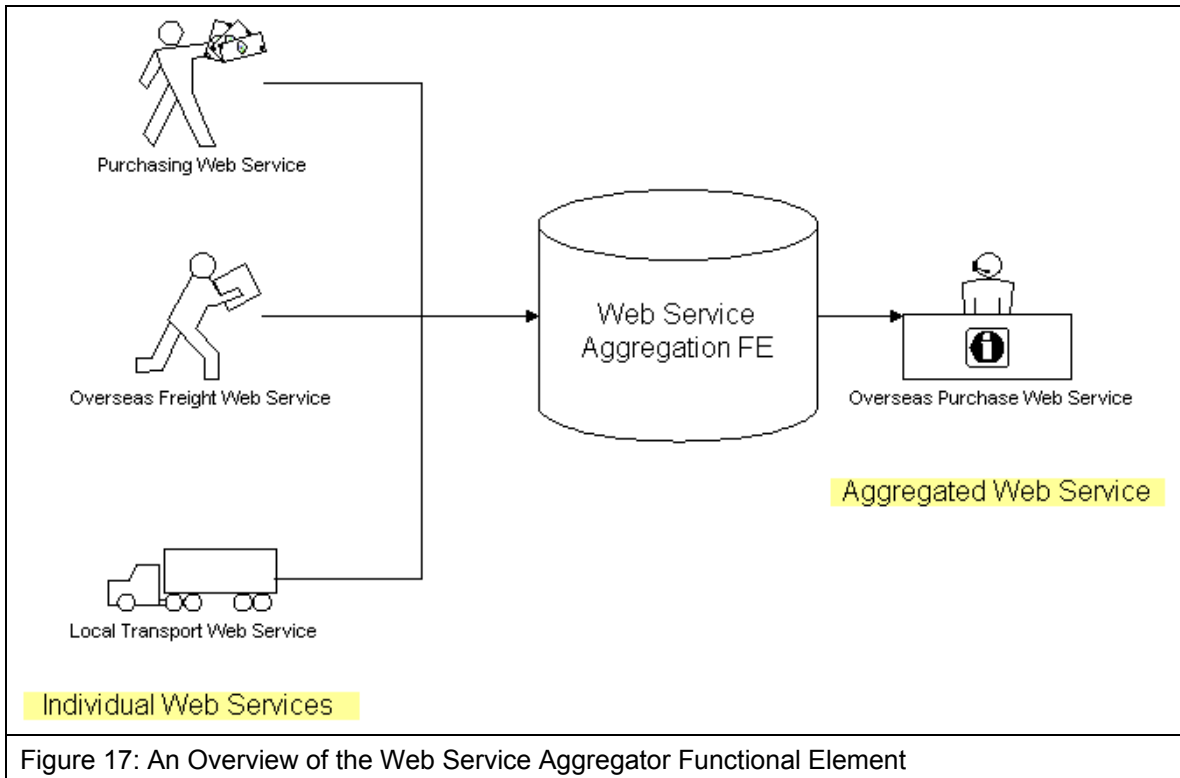


Figure 17: An Overview of the Web Service Aggregator Functional Element

3619

3620 2.16.3 Key Features

3621 Implementations of the Web Service Aggregator Functional Element are expected to provide the
 3622 following key features:

- 3623 1. The Functional Element MUST provide a mechanism for composing any number of Web
 3624 Services into single Web Service according to specified Composition Rule(s).
- 3625 2. Individual web services can reside at any location, but it is expected to be accessible.
- 3626 3. As part of Key Feature (1), the WSDL of a web service used for composition MUST be
 3627 available.
- 3628 4. The Functional Element MUST support the definition, modification and removal of
 3629 Composition Rules.
- 3630 5. The Functional Element MUST encapsulate the composition logic used into an interpretable
 3631 XML-based script based on a particular standard*.

3632 *Example: BPEL or WSCI. The TC will have to decide on which standard to use*

3633

3634 In addition, the following key features could be provided to enhance the Functional Element
 3635 further:

- 3636 1. The Functional Element MAY provide the capability to transform the interpretable XML-based
 3637 script into an executable program.
- 3638 2. If Key Feature (1) is provided, then the Functional Element MAY also have the following
 3639 capabilities:
 - 3640 2.1 The ability to test the functionality of the aggregated Web Service,
 - 3641 2.2 A WSDL to describe the aggregated Web Service, and
 - 3642 2.3 The capability to publish the aggregated Web Service into an UDDI-compliant registry

3643 **2.16.4 Interdependencies**

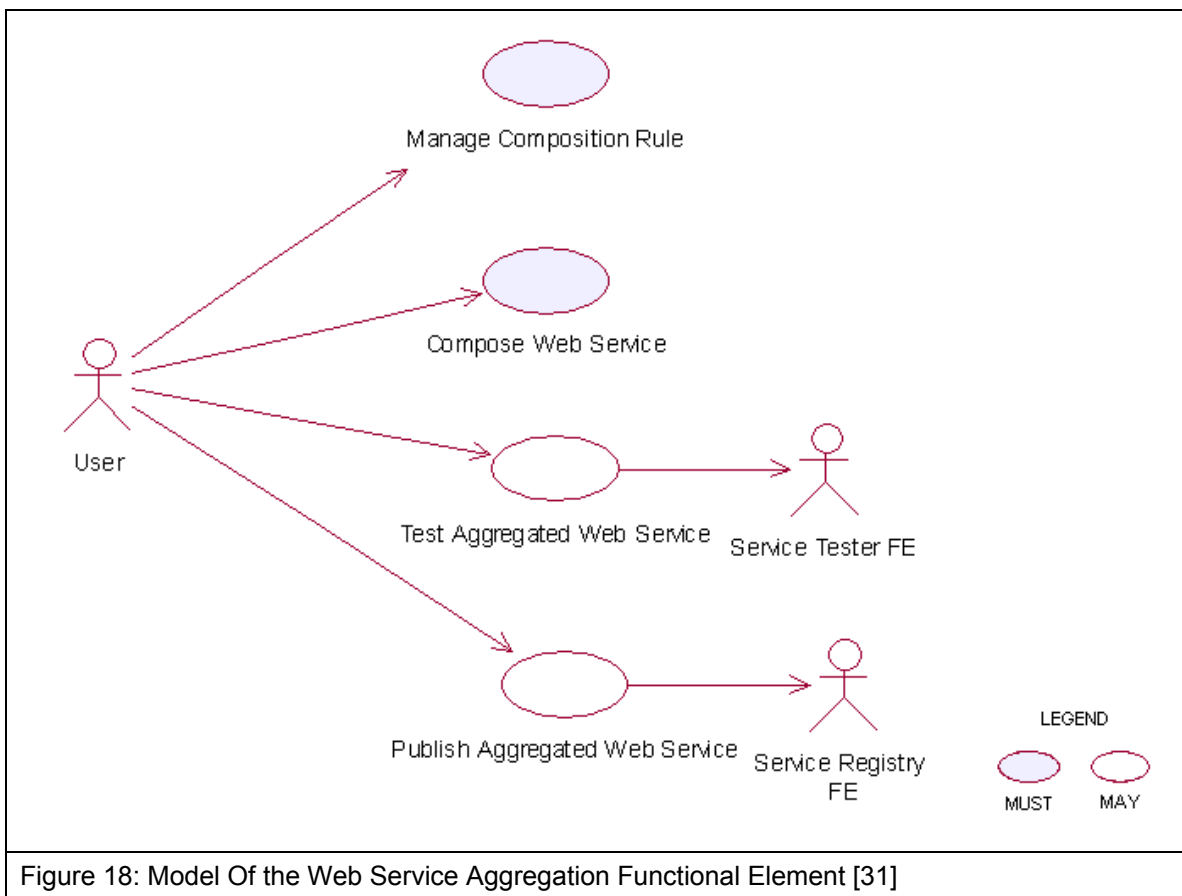
Interaction Dependencies	
Services Tester Functional Element	The Services Tester Functional Element may be used to test the performance of the aggregated web services
Service Registry Functional Element	The Services Registry Functional Element may be used to publish the aggregated web services

3644

3645 **2.16.5 Related Technologies and Standards**

3646 ** The interpretable XML based script should follow the standard identified by TC.*

3647 **2.16.6 Model**



3648

3649 **2.16.7 Usage Scenarios**

3650 **2.16.7.1 Manage composition rule**

3651 **2.16.7.1.1 Description**

3652 This use case allows the user to manage the composition rule used for Web Services
3653 aggregation.

3654 **2.16.7.1.2 Flow of Events**

3655 **2.16.7.1.2.1 Basic Flow**

3656 The use case begins when the user wants to manage a composition rule.

3657 1: The user sends a request to the Functional Element together with the composition rule and
3658 operation.

3659 2: Based on the operation it specified, one of the following sub-flows is executed:

- 3660 • If the operation is '**Define a rule**', then sub-flow 2.1 is executed.
- 3661 • If the operation is '**Update a rule**', then sub-flow 2.2 is executed.
- 3662 • If the operation is '**Retrieve a rule**', then sub-flow 2.3 is executed.
- 3663 • If the operation is '**Remove a rule**', then sub-flow 2.4 is executed.

3664 2.1: Define Rule.

3665 2.1.1: The Functional Element gets the composition rule, i.e. names of all Web Service,
3666 the sequence specification, parameters mapping between Web Services.

3667 2.1.2: The Functional Element verifies the correctness of composition rule.

3668 2.1.3: The Functional Element saves the composition rule to persistent mechanism.

3669 2.2: Update Rule.

3670 2.2.1: The Functional Element gets the name of composition rule.

3671 2.2.2: The Functional Element retrieves the composition rule definition from persistent
3672 mechanism.

3673 2.2.3: The Functional Element verifies the correctness of composition rule.

3674 2.2.4: The Functional Element updates the composition rule.

3675 2.3: Retrieve Rule.

3676 2.3.1: The Functional Element gets the name of composition rule.

3677 2.3.2: The Functional Element retrieves the definition of composition rule.

3678 2.3.3: The Functional Element returns the definition of rule.

3679 2.4: Remove Rule.

3680 2.4.1: The Functional Element gets the name of composition rule.

3681 2.4.2: The Functional Element checks whether the rule exists.

3682 2.4.3: The Functional Element removes the rule.

3683 3: The Functional Element returns the results to indicate the success or failure of this operation to
3684 the user and the use case ends.

3685 **2.16.7.1.2.2 Alternative Flows**

3686 1: Composition Rule Already Created.

3687 1.1: If in the basic flow 2.1.2, the same rule already created, Functional Element will return an
3688 error message to the user and the use case ends.

3689 2: Composition Rule Not Exist.

3690 2.1: If in the basic flow 2.2, 2.3, and 2.4 the specified rule does not exist, Functional Element
3691 will return an error message to the user and the use case ends.

3692 3: Persistency Mechanism Error.

3693 3.1: If in the basic flow 2.1, 2.2, 2.3, and 2.4, the Functional Element cannot perform data
3694 persistency, Functional Element will return an error message to the user and the use case
3695 ends.

3696 **2.16.7.1.3 Special Requirements**

3697 None.

3698 **2.16.7.1.4 Pre-Conditions**

3699 None.

3700 **2.16.7.1.5 Post-Conditions**

3701 None.

3702 **2.16.7.2 Compose Web Services**

3703 **2.16.7.2.1 Description**

3704 This use case will allow users to aggregate several simpler services into a higher-level service.

3705 **2.16.7.2.2 Flow of Events**

3706 **2.16.7.2.2.1 Basic Flow**

3707 This use case begins when any user wants to compose a Web Service.

3708 1: The user passes in a list of parameters for composition, including URLs of the WSDL,
3709 composition rules.

3710 2: Functional Element checks the signature of the Web Services to be composed via accessing
3711 WSDL.

3712 3: Functional Element generates interpretable XML-based script to encapsulate the composition
3713 logic.

3714 4: Functional Element returns the generated script and the use case ends.

3715 **2.16.7.2.2.2 Alternative Flows**

3716 1: Functional Element generates executable program and WSDL.

3717 1.1: At basic flow 3, Functional Element may transform the interpretable XML-based script
3718 into an executable program, if the user requested.

3719 1.2: At basic flow 3, Functional Element may generate WSDL for the executable program, if
3720 the user requested.

3721 1.3: Functional Element returns the code of executable program and WSDL file

3722 2: Functional Element detects ambiguity in Web Services signature.

3723 2.1: At basic flow 2, Functional Element encounters an ambiguity in the Web Services
3724 signature which it cannot resolve.

3725 2.2: Functional Element returns an error message that there is a composition error.

3726 3: Functional Element detects error in Web Services composition.

3727 3.1: At basic flow 3, Functional Element encounters an error in the Web Services
3728 composition.

3729 3.2: Functional Element returns an error message that there is a composition error.

3730 **2.16.7.2.3 Special Requirements**

3731 None.

3732 **2.16.7.2.4 Pre-Conditions**

3733 The composition rule for this Web Services aggregation must be pre-defined.

3734 **2.16.7.2.5 Post-Conditions**

3735 The generated program is ready for deployment in any Web Services container.

3736

3737 **2.16.7.3 Test Aggregated Web Services**

3738 **2.16.7.3.1 Description**

3739 This use case will allow users to test the functionality of aggregate web service.

3740 **2.16.7.3.2 Flow of Events**

3741 **2.16.7.3.2.1 Basic Flow**

3742 This use case begins when any user wants to test aggregated web service.

3743 1: The user passes in a list of parameters for testing, including URLs of the WSDL, values of
3744 parameters for invocation.

3745 2: Functional Element invokes the aggregated web service with parameters.

- 3746 3: Functional Element compares the returned parameter with the expected values.
- 3747 4: Functional Element returns the result of comparison and the use case ends.
- 3748 **2.16.7.3.2 Alternative Flows**
- 3749 1: Functional Element cannot invoke the aggregated web service.
- 3750 1.1: At basic flow 2, Functional Element encounters problems of invoking the aggregated web
3751 services.
- 3752 1.2: Functional Element returns an error message that indicates the invocation error.
- 3753 **2.16.7.3.3 Special Requirements**
- 3754 None.
- 3755 **2.16.7.3.4 Pre-Conditions**
- 3756 The executable program must be generated and deployed in web services hosting environment
3757 and ready for invocation.
- 3758 **2.16.7.3.5 Post-Conditions**
- 3759 None.
- 3760 **2.16.7.4 Publish Aggregated Web Services**
- 3761 **2.16.7.4.1 Description**
- 3762 This use case will allow users to publish the aggregated web services into UDDI registry.
- 3763 **2.16.7.4.2 Flow of Events**
- 3764 **2.16.7.4.2.1 Basic Flow**
- 3765 This use case begins when any user wants to publish the aggregated web services into UDDI
3766 registry.
- 3767 1: The user passes in a list of parameters for publishing, including URLs of the WSDL of
3768 aggregated web services, URL of UDDI and parameters of business and services description.
- 3769 2: Functional Element checks the availability of UDDI.
- 3770 3: Functional Element publishes services description of aggregated web services into UUDI.
- 3771 4: Functional Element returns the publish result and the use case ends.
- 3772 **2.16.7.4.2.2 Alternative Flows**
- 3773 1: UDDI registry server is not available
- 3774 1.1: At basic flow 2, Functional Element cannot connect to UDDI registry if UDDI registry
3775 server is not available.
- 3776 1.2: Functional Element returns the error message that UDDI connection cannot be built.
- 3777 2: Functional Element detects error in Web Services publishing.

3778 2.1: At basic flow 3, Functional Element encounters an error in the publishing Web Services.

3779 2.2: Functional Element returns an error message that there is a publishing error.

3780 **2.16.7.4.3 Special Requirements**

3781 None.

3782 **2.16.7.4.4 Pre-Conditions**

3783 The WSDL of the aggregated web services must exist.

3784 **2.16.7.4.5 Post-Conditions**

3785 None

3786

3 Functional Elements Usage Scenario

3787

The Functional Elements are designed to be building blocks that can be assembled to accelerate web service-enabled applications. From these Functional Elements, a variety of solutions can be built. In this section, the following solutions are provided as examples

3788

3789

3790

- A service monitoring solution for the management of services in a SOA model

3791

- Enabling security through the Secure SOAP Functional Element

3792

- Decoupled User Access Management with support for multi-domain capabilities in a web service environment

3793

3794

3795

3796 **3.1 Service Monitoring**

3797 In a SOA environment, management of services includes the capability to monitor services within
3798 the management domain. These includes:

- 3799 • Monitoring the performance of services invoked
- 3800 • Generating audit trails of services invoked
- 3801 • Monitoring and testing the availability of services on the remote machine (server)

3802 A basic solution can be realised through the aggregation of two Functional Element, namely
3803 Service Management and Service Tester, as shown in Figure 19. This solution can be improved
3804 with notification capabilities, using the Notification Engine, be it to a remote client, a system
3805 administrator or an end user of a particular service. Further enhancement can be added with a
3806 Rule Engine that will have the cognitive ability to make decisions. An example of this
3807 enhancement would be the ability to decide when should notifications or alerts be sent and in
3808 what form.

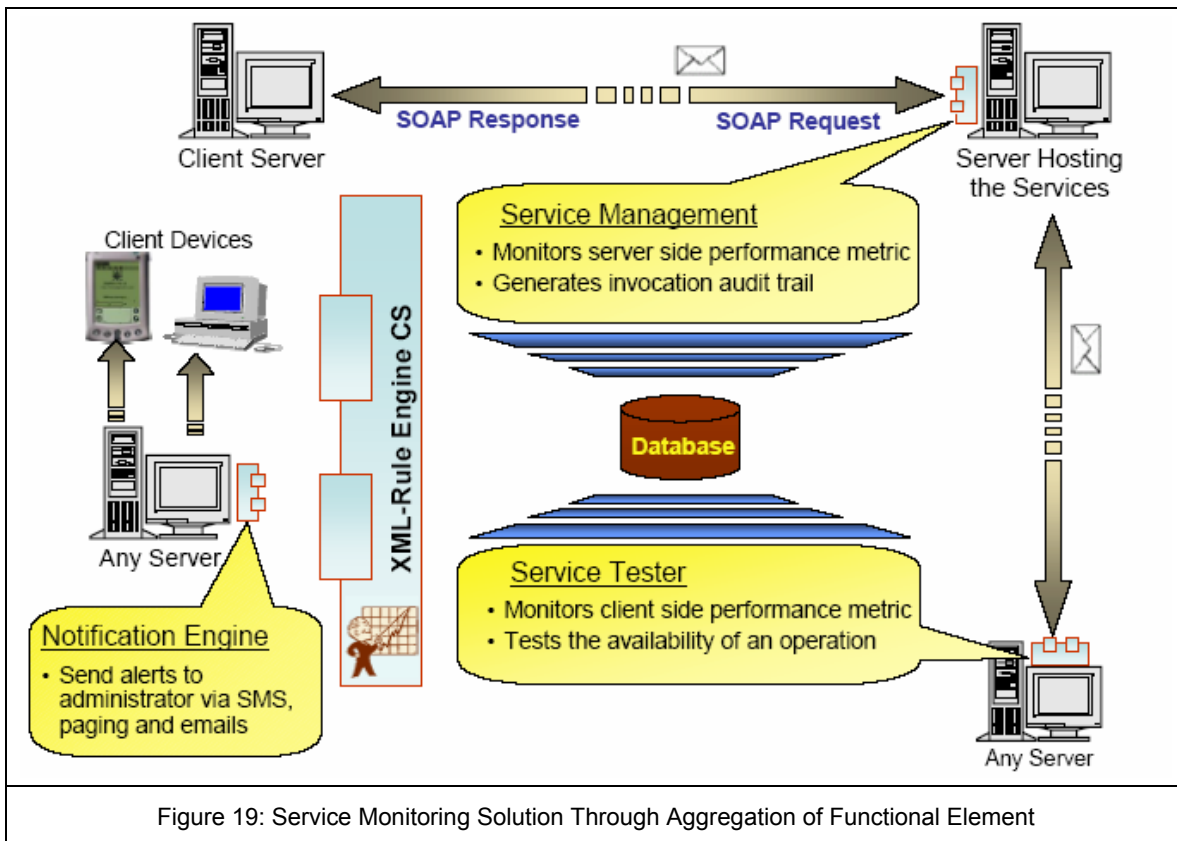
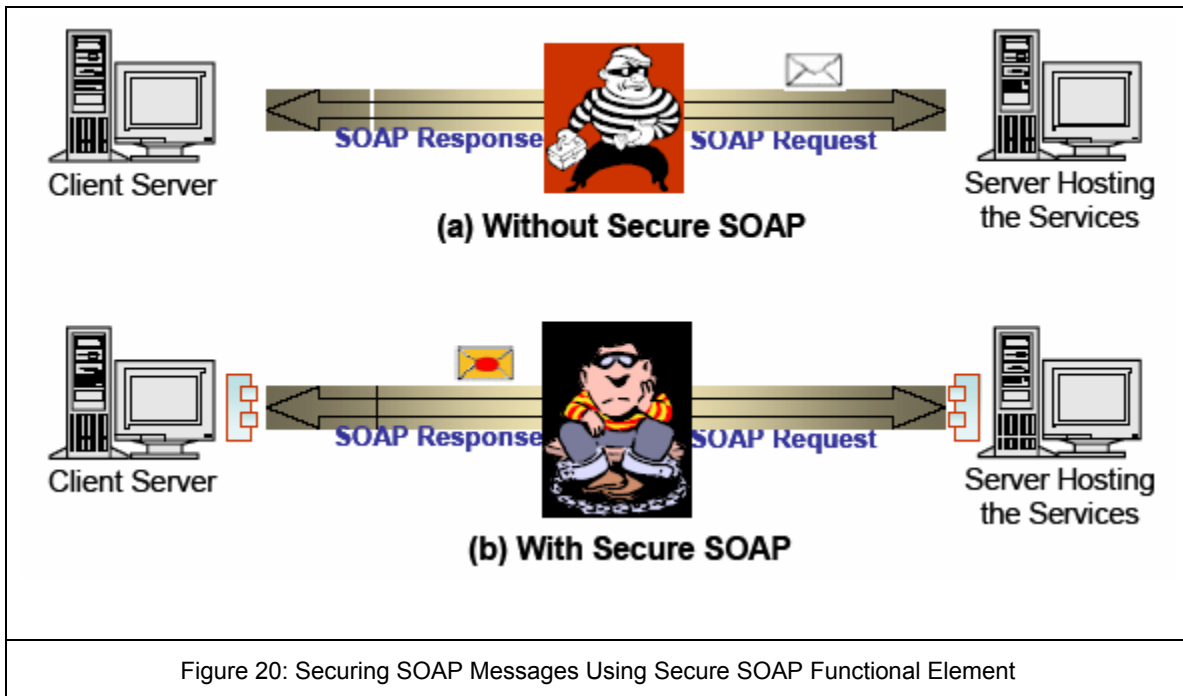


Figure 19: Service Monitoring Solution Through Aggregation of Functional Element

3809

3810 **3.2 Securing SOAP Messages**

3811 SOAP in its pure form does not have any built in security as it is meant to be a simple and
3812 lightweight protocol. As such, where security is needed, additional capabilities must be provided.
3813 Presently, standards like XML Encryption and XML Signature are available. Making use of these
3814 standards, the Secure Soap Functional Element, when deployed on both the sending and
3815 receiving parties, will be able to provide encryption and signing of messages as illustrated in
3816 Figure 20.
3817



3818

3819 3.3 Decoupled User Access Management

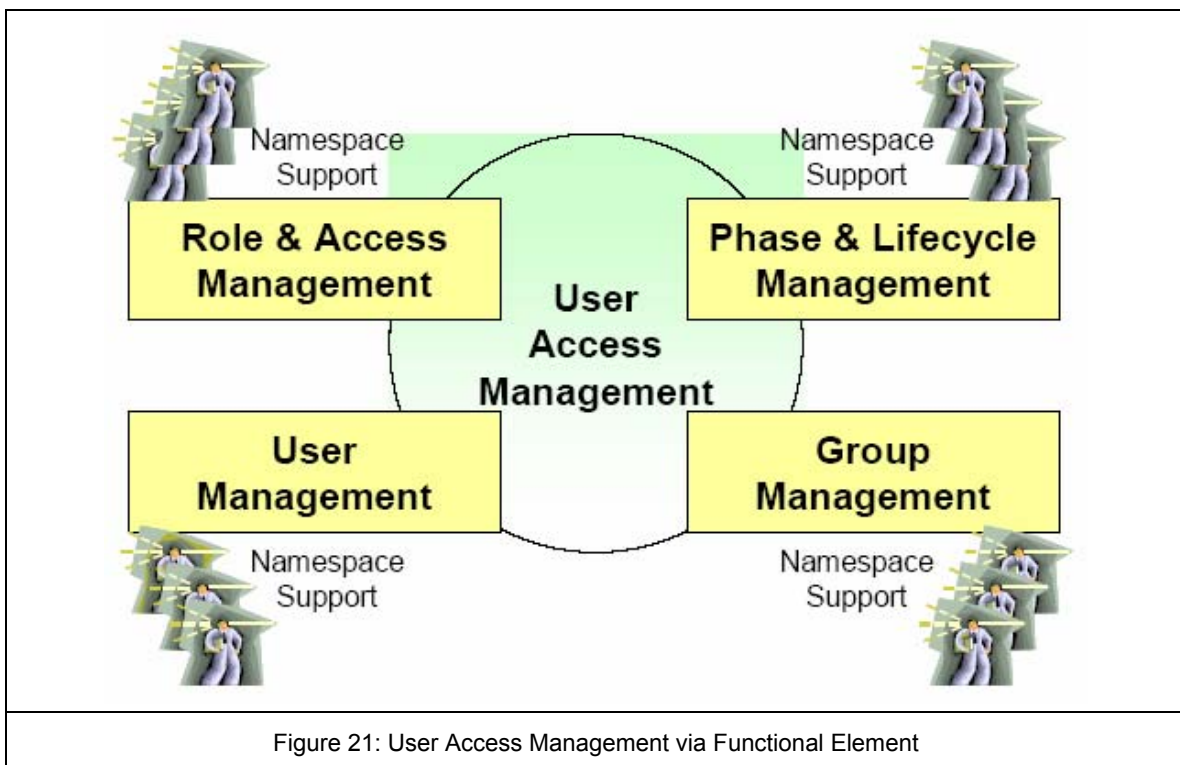
3820 User Access Management (UAM) has been implemented in many forms and in a wide variety of
3821 ways, from the most basic to the most complex. At the most simple form, the functionality would
3822 include username and password support. On the end of the scale, it would include functionalities
3823 like distributed access management, replication capabilities and fine-grain controls just to name a
3824 few.

3825 In this specification, the goal is to provide a set of Functional Element that can be used as
3826 building blocks for UAM, and can be extended when the need arises. It is provided as a
3827 decoupled building blocks consisting of four Functional Elements, namely User Management,
3828 Group Management, Role & Access Management and Phase & Lifecycle Management, as
3829 illustrated in Figure 21. These Functional Elements can be used in a variety of combinatorial
3830 forms, and some of these examples include:

- 3831 • User Management only, or
- 3832 • User Management and Group Management, or
- 3833 • User Management and Role & Access Management, or
- 3834 • User Management, Group Management and Role & Access Management, or
- 3835 • All the four Functional Elements in tandem

3836 On the same token, any of the Functional Elements can be replaced with similar functionality third
3837 party web services. As these services are designed to be in a web service environment, each of
3838 them also supports the concept of namespace. This namespace provision enables each of the
3839 Functional Elements to be used as web services that can be accessed by multiple organisations
3840 or to cater for users from different domains. With this, access control for example, can be defined
3841 for multiple domains without corruption or interferences problems.

3842



4 References

1. FWSI TC, OASIS, **Web Service Implementation Methodology Working Draft 0.1**, <http://www.oasis-open.org/apps/org/workgroup/fwsi/documents.php>, September 2004.
2. FWSI TC, OASIS, **Functional Elements Requirements Working Draft 0.1a**, <http://www.oasis-open.org/apps/org/workgroup/fwsi/documents.php>, July 2004.
3. S. Bradner, **Key words for use in RFCs to Indicate Requirement Levels**, 809, <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.
4. Cheng, Y.S., **WSRA Use Case Specifications - Event Handler**, version 1.0, JSSL of Singapore Institute of Manufacturing Technology, November 2003.
5. Wu, Y.Z., **WSRA Use Case Specifications – Group Management**, version 1.4, JSSL of Singapore Institute of Manufacturing Technology, September 2003.
6. OASIS Web Services Security TC, **Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)**, <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>, March 2004
7. OASIS, **Security Assertion Markup Language (SAML) v1.0**, <http://www.oasis-open.org/committees/download.php/2290/oasis-sstc-saml-1.0.zip>, September 2002.
8. Liberty Alliance, **ID-FF 1.2 Specifications**, version 1.2, http://www.projectliberty.org/specs/index.html#ID-FF_Specs.
9. Liberty Alliance, **ID-WSF 1.0 Specifications**, version 1.0, http://www.projectliberty.org/specs/index.html#ID-WSF_Specs.
10. **Web Services Federation Language (WS-Federation)**, <http://www-106.ibm.com/developerworks/webservices/library/ws-fed/>, July 2003.
11. Chan, L.P., **WSRA Use Case Specifications – Identity Management**, version 0.3, JSSL of Singapore Institute of Manufacturing Technology, December 2003.
12. Yin, Z.L., **WSRA Use Case Specifications – Log Utility**, version 1.2, JSSL of Singapore Institute of Manufacturing Technology, December 2002.
13. Limbu, D.K., **WSRA Use Case Specifications - Notification Engine**, version 1.2, JSSL of Singapore Institute of Manufacturing Technology, December 2002.
14. Wu, Y.Z., **WSRA Use Case Specifications - Phase & LC Management**, version 1.3, JSSL of Singapore Institute of Manufacturing Technology, October 2003.
15. Jebaraj, D., **WSRA Use Case Specifications – Presentation Transformer**, version 1.1, JSSL of Singapore Institute of Manufacturing Technology, November 2002.

-
16. Xu, X.J., **WSRA Use Case Specifications - Role & Access Management**, version 1.3, JSSL of Singapore Institute of Manufacturing Technology, September 2003.
 17. Ramasamy, V., **WSRA Use Case Specifications - Search**, version 1.3, JSSL of Singapore Institute of Manufacturing Technology, June 2004.
 18. W3C, **XML-Signature Syntax and Processing**, W3C Recommendation, <http://www.w3.org/TR/xmlsig-core/>, February 2002.
 19. W3C, **XML-Encryption Syntax and Processing**, W3C Recommendation, <http://www.w3.org/TR/xmlenc-core/>, August 2002.
 20. Wu, Y.Z., **WSRA Use Case Specifications - Secure SOAP Management Private**, version 1.2, JSSL of Singapore Institute of Manufacturing Technology, December 2002
 21. Limbu, D.K., **WSRA Use Case Specifications - Sensory Engine**, version 1.2, JSSL of Singapore Institute of Manufacturing Technology, December 2002.
 22. Cheng, H.K., **WSRA Use Case Specifications - Service Management**, version 1.2, JSSL of Singapore Institute of Manufacturing Technology, December 2002.
 23. OASIS UDDI Specification TC, **Universal Description, Discovery And Integration (UDDI) Data Structure**, OASIS Standard, version 2.03, <http://uddi.org/pubs/DataStructure-V2.03-Published-20020719.pdf>, July 2002.
 24. OASIS UDDI Specification TC, **Universal Description, Discovery And Integration (UDDI) API Specifications**, OASIS Standard, version 2.04, <http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.pdf>, July 2002.
 25. OASIS ebXML Registry TC, **ebXML Registry Information Model Specification**, version 2.0, OASIS Standard, <http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebrim.pdf>, April 2002.
 26. OASIS ebXML Registry TC, **ebXML Registry Services Specification**, version 2.0, <http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebrs.pdf>, April 2002.
 27. Ramasamy, V., **WSRA Use Case Specifications - Service Registry**, version 1.2, JSSL of Singapore Institute of Manufacturing Technology, December 2002.
 28. W3C, **Web Services Description Language**, version 1.1, W3C Note, <http://www.w3.org/TR/wsdl>, March 2001.
 29. Andersson, K., **WSRA Use Case Specifications – Service Tester**, version 1.2, JSSL of Singapore Institute of Manufacturing Technology, December 2002.
 30. Xu, X.J., **WSRA Use Case Specifications – User Management**, version 1.2, JSSL of Singapore Institute of Manufacturing Technology, December 2002.
 31. Cheng, H.K., **WSRA Use Case Specifications – Web Service Aggregator**, version 1.2, JSSL of Singapore Institute of Manufacturing Technology, December 2002.

3844 **Appendix A. Acknowledgments**

3845 Special thanks to the following individuals who contributed significantly towards to the initial draft
3846 of this work during the development of this specification:

- 3847 • Ang Chai Hong, Singapore Institute of Manufacturing Technology
- 3848 • Chan Lai Peng, Singapore Institute of Manufacturing Technology
- 3849 • Cheng Yushi, Singapore Institute of Manufacturing Technology
- 3850 • Dilip Kumar Limbu, Singapore Institute of Manufacturing Technology
- 3851 • V. Ramasamy, Singapore Institute of Manufacturing Technology
- 3852 • Wu Yingzi, Singapore Institute of Manufacturing Technology
- 3853 • Xu Xingjian, Singapore Institute of Manufacturing Technology
- 3854 • Yin Zunliang, Singapore Institute of Manufacturing Technology

3855

3856 The following individuals were members of the committee during the development of this
3857 specification:

- 3858 • Christopher Haddad, Individual
- 3859 • Lee Eng Wah, Singapore Institute of Manufacturing Technology
- 3860 • Lim Kenneth, CrimsonLogic Pte Ltd
- 3861 • Ravi Shankar, CrimsonLogic Pte Ltd
- 3862 • Jagdip Talla, CrimsonLogic Pte Ltd
- 3863 • Andy Tan, Individual
- 3864 • Roberto Pascual, Infocomm Development Authority (IDA) of Singapore

3865 The committee would also like to express its appreciation for the encouragement and guidance
3866 provided by Jamie Clark throughout the course of the TC work.

3867

3868 The committee would also like to record its heartfelt appreciation to IBM Rational (Singapore) Pte.
3869 Ltd. for kindly agreeing to allow the use of the Rational Tools towards the creation of the Use
3870 Case Model used in this document.

3871

3872

3873

3874

3875

Appendix B. Revision History

3876

The following revision of this document represents the major milestones achieved.

3877

Rev	Date	By Whom	What
FWSI-FESC-specifications-01.doc	01-July-2004	Huang Kheng Cheng Puay Siew Tan	First Draft
FWSI-FESC-specifications-02.doc	18-October-2004	Huang Kheng Cheng Puay Siew Tan	Second Draft

3878

Appendix C. Notices

3880 OASIS takes no position regarding the validity or scope of any intellectual property or other rights
3881 that might be claimed to pertain to the implementation or use of the technology described in this
3882 document or the extent to which any license under such rights might or might not be available;
3883 neither does it represent that it has made any effort to identify any such rights. Information on
3884 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS
3885 website. Copies of claims of rights made available for publication and any assurances of licenses
3886 to be made available, or the result of an attempt made to obtain a general license or permission
3887 for the use of such proprietary rights by implementors or users of this specification, can be
3888 obtained from the OASIS Executive Director.

3889 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
3890 applications, or other proprietary rights which may cover technology that may be required to
3891 implement this specification. Please address the information to the OASIS Executive Director.

3892 Copyright © OASIS Open 2004. All Rights Reserved.

3893 This document and translations of it may be copied and furnished to others, and derivative works
3894 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
3895 published and distributed, in whole or in part, without restriction of any kind, provided that the
3896 above copyright notice and this paragraph are included on all such copies and derivative works.
3897 However, this document itself does not be modified in any way, such as by removing the
3898 copyright notice or references to OASIS, except as needed for the purpose of developing OASIS
3899 specifications, in which case the procedures for copyrights defined in the OASIS Intellectual
3900 Property Rights document must be followed, or as required to translate it into languages other
3901 than English.

3902 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
3903 successors or assigns.

3904 This document and the information contained herein is provided on an "AS IS" basis and OASIS
3905 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
3906 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE
3907 ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
3908 PARTICULAR PURPOSE.